Large-Scale Simultaneous Localization and Mapping for Teams of Mobile Robots

Robert George Reid

Thesis presented for the degree of
Doctor of Philosophy

School of Electrical, Electronic and Computer Engineering
The University of Western Australia

July 2016

Last updated: 26th July 2016

The latest version of this manuscript and accompanying videos
are available on-line at http://reid.ai/thesis

In loving memory of my mother

Abstract

Localization and mapping are core requirements for teams of mobile robots to cooperate autonomously in everyday environments. From emergency search and rescue, to precision agriculture and space exploration, there are many applications where it is advantageous to deploy teams of robots without relying on external localization or *a priori* maps, and instead using on-board sensors only. This problem, called Simultaneous Localization And Mapping (SLAM), has been well-studied for individual robots. While many single-robot SLAM solutions have been adapted to teams of robots, highly-centralized architectures are typically proposed that fail to address real-world problems such as intermittent and lossy communications, and particularly in the case of large-scale deployments.

For robust deployments of autonomous teams of robots, a decentralized multi-robot SLAM (MR-SLAM) solution is required; one that allows teams of robots to operate for extended periods independent of a central server by sharing SLAM data and performing loop closures on-board. Several decentralized architectures have been described in the literature, however none have demonstrated MR-SLAM with the mapping fidelity required for both indoor and outdoor deployments at large scales. State-of-the-art large-scale MR-SLAM systems have demonstrated up to 15 robots exploring a 500×500 meter urban environment at the 2010 Multi Autonomous Ground-robotic International Challenge (MAGIC). While these systems were centralized, their Decoupled Centralized Architectures (DCA) allowed individual robots brief periods of limited autonomy. Without the ability to share SLAM data or close loops on-board, however, architectures like DCA are unable to provide teams of robots with extended operations or high-level autonomy independent of a central server.

This thesis contributes in three areas: 1) The design of a MR-SLAM architecture that combines a novel hybrid-decentralized pose-graph SLAM technique with a unique submap-based approach; this architecture distributes pose graph optimization and global map building across all robots, enabling decentralized teams to operate autonomously for extended periods. 2) Highly-parallelized algorithms that enable efficient global occupancy gridmap fusion and efficient submap correlations that generate multimodal constraints; together these algorithms allow the proposed architecture to be realized on commodity hardware. 3) Continuous Mode Blending Optimization (COMBO), a novel technique that enables pose graphs with multimodal constraints to be optimized using traditional nonlinear least squares; this allows complex environments and effects such as perceptual aliasing to be modeled more accurately.

These contributions have been demonstrated on-line at the MAGIC challenge, and more recently with three merged challenge datasets replayed in real-time— these are the largest multi-robot datasets described in the literature, with heterogeneous teams of 10, 14 and 23 robots exploring over 3.1 km, 6.1 km and 8.3 km of total odometry, respectively. Results include global occupancy gridmap fusion at over 20 Hz, with globally-referenced mapping accuracies of ± 0.27 m, ± 0.62 m and ± 0.35 m, without using Global Positioning System (GPS) sensors. The proposed submapping technique compresses sensor data approximately 50 fold, reducing communications bandwidth requirements to averages of 1.2 KB/s, 2.2 KB/s and 2.4 KB/s. Submap constraints with multimodal Gaussian distributions are generated in real-time and desirable convergence properties are demonstrated using COMBO. The total computation, storage and communications requirements are shown to scale linearly, enabling future deployments orders of magnitude larger.

By distributing the MR-SLAM back-end so that all robots are able to build their own copies of the global gridmap, the proposed architecture enables teams of robots to operate autonomously without continuous communications to a centralized server. This distributed approach is highly scalable, since each robot includes the computational resources it requires to process its own sensor data and maintain its own registration to the global pose graph. The proposed hybrid-decentralized and distributed MR-SLAM architecture provides robust localization and mapping capabilities for large-scale deployments of autonomous robots in real-world conditions. This approach enables many applications where teams of robots need to cooperate in GPS-denied environments, with imperfect communications and without *a priori* maps.

Contents

List of Figures

List of Tables

List of Abbreviations

COTS	Commercial Off-The-Shelf
DARPA	Defense Advanced Research Projects Agency
DCA	Decoupled Centralized Architecture
DDF	Decentralized Data Fusion
DDS	Data Distribution System
DOF	Degrees Of Freedom
EIF	Extended Information Filter
EKF	Extended Kalman Filter
FOV	Field Of View
GCS	Ground Control Station
GPS	Global Positioning System
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HMI	Human Machine Interface
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit
IPC	Inter-Process Communication
ISR	Intelligence, Surveillance and Reconnaissance
MAGIC	Multi-Autonomous Ground-robotics International Challenge
MAS	Multi-Agent System
MAV	Micro Aerial Vehicle
MCL	Monte Carlo Localization

MDA	Model-Driven Architecture
MEMS	Micro Electro Mechanical Systems
MOG	Mixture of Gaussians
MR-SLAM	Multi-Robot Simultaneous Localization and Mapping
MRS	Multi-Robot System
OOI	Objects Of Interest
Penn	University Of Pennsylvania
PR	Place Recognition
RASR	Reconnaissance and Autonomy for Small Robots
RBPF	Rao-Blackwellized Particle Filter
RF	Radio Frequency
RMSE	Root Mean Squared Error
SGD	Stochastic Gradient Descent
SLAM	Simultaneous Localization and Mapping
SOA	Service-Oriented Architecture
TM	Team Michigan
TOF	Time of Flight
TRL	Technology Readiness Level
UGV	Unmanned Ground Vehicle
USAR	Urban Search and Rescue
WAMbot	Western Australian MAGIC Robot

Acknowledgments

There are many people who have helped with my research over the last few years. I would like to express my sincerest gratitude and appreciation to my supervisors, Prof. Thomas Bräunl and Dr. Adrian Boeing. Thank you for your endless support, guidance and motivation during my candidature, and for your feedback on this manuscript. Thank you also to Keith Godfrey for your support and feedback.

Adrian, thank you for bringing the WAMbot team together and your tireless efforts that enabled us to compete in the 2010 Multi-Autonomous Ground-robotic International Challenge. Thanks go to the entire team, whose extensive efforts allowed us to compete on the international stage and achieve fourth place. Particular thanks to the core team members: Adrian Boeing, Michael Fazio, Adam Gandossi, Nicholas Garel, Aidan Morgan, Frank Ophelders and Kevin Vinsen. Thanks also to Anthony Attwood, Brian Frisch, Chang Su Lee, Mark Boulton, Martin Masek, Sam Lopes and Sushil Pangeni. Together all of your efforts produced a team of robots that helped to validate my research contributions. Acknowledgements also to our sponsors, particularly Thales and DSTO, who made it possible to field a team of robots at the challenge.

Sincere gratitude to Prof. Daniel Lee, Alex Kushleyev and the rest of the University of Pennsylvania team for sharing their challenge datasets and hosting my visit to the GRASP lab in 2011. Thank you also to Prof. Edward Olson and the rest of Team Michigan for sharing their challenge datasets.

Endless thanks to my family for all their love, support and words of encouragement from afar. Finally, my deepest and sincerest gratitude to my wife, Lauren. Your patience and unconditional support has made this possible.

1 Introduction

This thesis describes my research work in Simultaneous Localization and Mapping (SLAM) algorithms and system architectures that enable teams of robots to cooperate in real-world environments. This chapter motivates my research in multi-robot SLAM (MR-SLAM) and describes the significance of this area. Section 1.1 starts by summarizing key trends in robotics and introduces the SLAM problem. In Section 1.2 I introduce cooperative multi-robot systems (MRS), and the MR-SLAM solutions that they require. Section 1.3 provides an overview of current MRS and MR-SLAM capabilities and introduces the MAGIC challenge, which provided the initial motivations for my research in this area. Finally, Section 1.4 explores potential applications of these technologies and considers how they could enable real-world deployments of large teams of robots in the near future.

1.1. Robots

Humans have been fascinated by robots for millennia. The earliest descriptions of clockwork machines and automata that could serve and entertain people can be found in ancient Greek and Chinese texts [1]. Over the last few decades, modern microelectronics have enabled robots to permeate many aspects of our lives: from household appliances and toys that are both helpful and entertaining, to industrial machines that help drive economic growth and rovers that explore other planets.

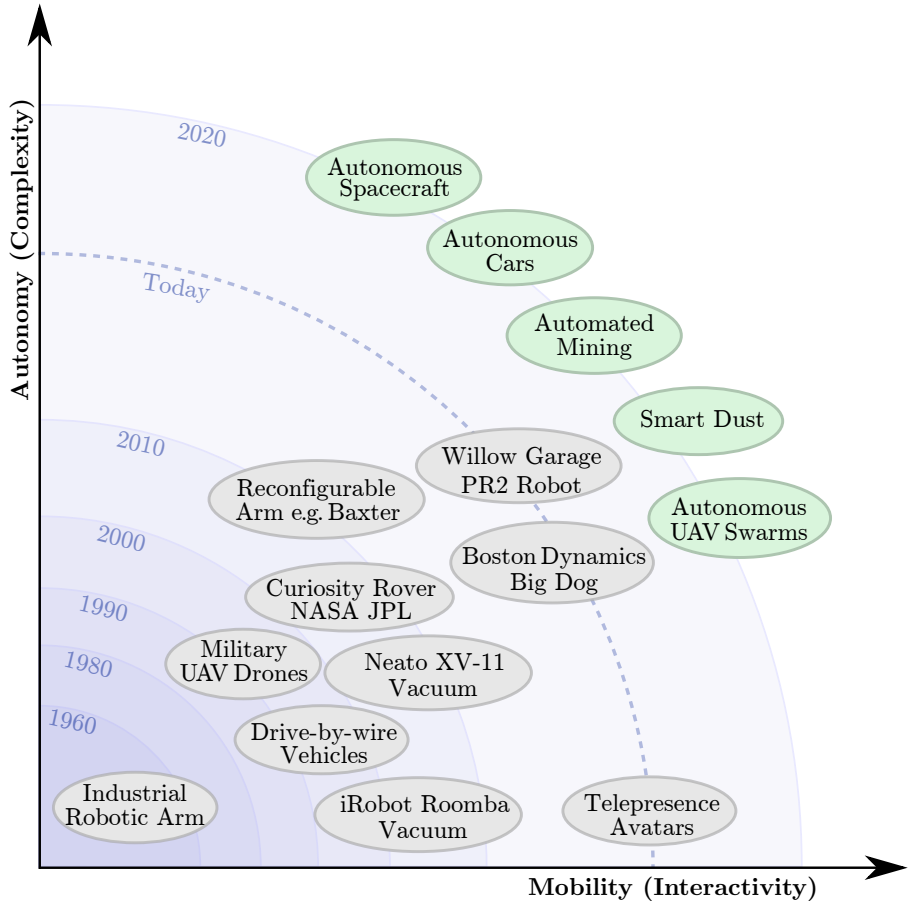


Figure 1.1. Trends in robotics: An anecdotal representation of the past and possible future directions in robotics. Over recent decades (blue), the frontiers of robotic technology have seen levels of autonomy and mobility increase, with corresponding increases in complexity and interactivity. While this trend is likely to continue, future robotic technologies (green) are likely to be increasingly connected, sharing information and cooperating to form complex multi-robot systems.

1.1.1. Trends in Robotics

The growth in robotic technologies has been accompanied with increased capabilities such as autonomy and mobility. The graph in Figure 1.1 combines these and plots several examples. While somewhat anecdotal, it suggests that autonomy and mobility, along with attributes such as complexity and interactivity, have increased steadily in recent decades. It is easy to imagine this trend continuing into the near future, where many robots from science fiction could become a reality. Technology "futurists" such as Brooks [2] and Kurzweil [3] have predicted that robotic systems, and their role in our everyday lives, will grow exponentially.

The two main forces driving these advancements are economics and safety. Often referred to as the "three Ds": *dirty*, *dangerous* and *dull*, many tasks performed by humans are either dangerous, such as bomb disposal, or highly repetitive, such as welding in an industrial factory. Robots are often well suited to these types of tasks, because they can be more

consistent and are relatively expendable. Robots have historically been designed to replace humans in these dirty, dangerous and dull tasks, which means they are often designed with human-like capabilities.

While this approach has proven to be effective, the exponential growth predicted by Kurzweil et al. is more likely to be realized when we start deploying robots that cooperate in teams. Just as teams of humans can be more productive than individuals, teams of robots have the potential to be more productive also. Figure 1.1 lists several examples of near-future robotic systems that could benefit significantly from cooperative behaviors.

Teams of cooperative robots, or *multi-robot systems* (MRS), have the potential to provide capabilities with no human equivalent. High bandwidth communications, for example, will allow robots to quickly and accurately share knowledge, which will introduce many novel efficiencies. Consider, for example, a team of robots that shares maps of their environment: this sharing will enable individual robots to plan and navigate efficiently to places they have never previously visited.

In recent years the term "cloud robotics" has been used to describe MRS that share knowledge using a *centralized* server, often over the Internet [4, 5]. While cloud-based robotics concepts are still in their infancy, future developments are likely to focus on increasingly *decentralized* and *distributed* approaches. My research focuses on the latter, decentralized MRSs, and more specifically on architectures that enable teams of distributed robots to share spatial knowledge.

1.1.2. Mobile Robots

It is useful to distinguish between robots that are immobile, such as a robotic arm in a factory, and robots that are mobile, such as a self-driving car. This research is concerned with the latter, *mobile robots*. We use the term to describe a self-propelled robot that can move kinematically between locations in its environment. We refer to the robot's combined position and orientation as its *pose*.

Mobile robots can refer to robots that move over ground, underwater, through air and in micro-gravity environments. While my research can be applied to any of these environments, the focus of this work is primarily concerned with mobile robots that remain in contact with the ground. The term *unmanned ground vehicles* (UGVs) is often used more specifically to describe ground-based mobile robots, however I use the two terms interchangeably.

Mobile robots move through large and potentially dynamic environments, making perception much more difficult than industrial robots with limited working environments and

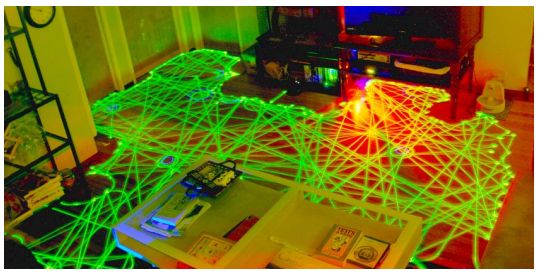
Figure 1.2. Two mobile robot vacuum cleaners: The iRobot Roomba 560 (left) and the Neato XV-11 (right). Image courtesy of Evan Ackerman, IEEE Spectrum.

rigid operating parameters. Mobile robots require additional sensors, better perception and higher degrees of autonomy to operate in frequently changing real-world environments.

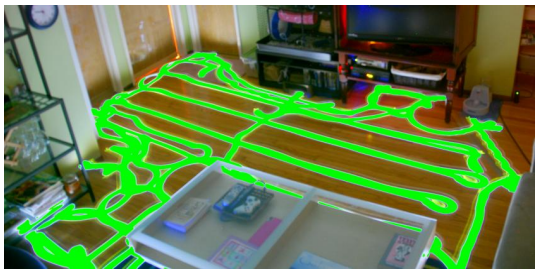
When explaining my research to others I have often used the iRobot Roomba as an example of a mobile robot (Figure 1.2, left). The Roomba is a robotic vacuum cleaner with two differentially driven wheels (Section 2.4.1). It alternates between wall-following and random driving modes, sensing the environment by bumping into objects. While watching a Roomba it is apparent that their randomized motions are inefficient, and that they do not model their environment or know where they are in it. The long-exposure photograph in Figure 1.3 (a) is an example of a random path driven by a Roomba. Contrast this with the way a human would perform the same task— we implicitly build an internal model of the environment, and keep track of our location and where we have already vacuumed.

1.1.3. Simultaneous Localization and Mapping

Navigating unknown environments requires a solution to the "Simultaneous Localization and Mapping" (SLAM) problem. The SLAM problem can be posed in the metaphorical



(a) The iRobot Roomba's random navigation will eventually cover the entire area.



(b) The Neato XV-11 navigates using SLAM, which allows it to vacuum more efficiently.

Figure 1.3. SLAM enables more efficient robotic vacuums: The paths taken by two commercial robots are shown in these long-exposure photographs. Images courtesy Evan Ackerman.

chicken-and-egg sense; for a robot to know its current location it needs an accurate map, however to incrementally build an accurate map it needs to know its location within the map. The two questions, "where am I?" (localization) and "what does the world look like?" (mapping) are intrinsically coupled by probabilistic uncertainties that must be answered simultaneously.

Handling these uncertainties requires a rigorous *probabilistic framework* and the ideal solution is often described as the *full* SLAM solution [6]. The SLAM problem is combinatorial, and in all but the simplest cases the full SLAM solution requires exponentially increasing computational resources (it is NP-complete) [7, 8]. Thus all practical SLAM algorithms, including my research contributions, exploit combinations of heuristics and approximations to run in polynomial time.

The SLAM problem has been the focus of considerable research since the late 1980s [9, 10, 11], and several broad approaches have been described for solving it. Thrun and Leonard provide a tutorial-style overview of these approaches in [12]. SLAM algorithms generally trade between *computational complexity*, *storage*, *accuracy*, *robustness* and *real-time execution*. Chapter 2 gives an in-depth review of the SLAM problem and the different classes of solutions.

A robust SLAM algorithm is essential for any mobile robot to navigate safely through an unstructured environment. The SLAM algorithm frequently defines a *global coordinate frame* for the robot to operate in; one that is generally used by all high-level functionality, such as navigation, path planning, exploration, object identification, object tracking and object manipulation. These dependencies make the SLAM algorithm a core part of any mobile robot architecture, and unrecoverable failures highly undesirable. Any robot disoriented by a SLAM failure is likely to fail to perform its task, or worse, may endanger humans, itself, or the environment.

While various SLAM algorithms have been demonstrated in laboratories, it is more difficult to produce robust solutions in *real-world* unstructured environments. Wheel slippage, for example, can produce noisy and biased odometry measurements, while Global Positioning System (GPS) sensors often produce very noisy and biased global localization. Robust real-world SLAM becomes even harder when these random and systematic sensing errors occur in environments that have redundant geometries or moving objects.

Consumer products using SLAM techniques have only recently begun appearing on the market. The Neato XV-11 vacuum is one example that was designed to compete with the Roomba described in Section 1.1.2; both products are shown in Figure 1.2. The XV-11 performs SLAM with an inexpensive lidar scanner [13] to dynamically build a map of its environment. Using this internal map the XV-11 is able to localize itself and navigate

while recording where it has already cleaned. The long-exposure photograph in Figure 1.3 demonstrates the purposeful navigation that SLAM enables on the XV-11.

1.2. Multi-Robot Systems

While cooperative teams of robots can often complete tasks more efficiently than uncooperative ones, this ability does not generally exist in consumer robots today. Multiple robot vacuum cleaners, for example, are unlikely to clean the entire floor of a large house: a team of Roombas will bump into each other and their paths will overlap inefficiently; and while a team of XV-11s may navigate efficiently, each robot will attempt to clean the entire floor once. Such a cleaning task could be manually divided by barriers, however any failures would result in only partial task completion. This household example hints at some of the core problems that must be solved by MRS. For a thorough review refer to [14, 15] and [16].

1.2.1. MRS Architectures

Current MRS research can be grouped loosely into two different architectures that are based on the method of coordination; the first approach uses a large number of simple and inexpensive robots operating in *uncoordinated swarms*. In these biologically inspired systems each robot responds only to its local environment with no formal *task allocation*; each robot performs simple tasks and any high-level abilities are often considered *emergent* behavior [17]. Using the robot vacuum example, an uncoordinated group of Roombas might eventually clean every room in a large house, however they would be inefficient and would require a large team to ensure coverage.

The second coordination approach, and the focus of this research work, involves *strongly coordinated* behaviors between relatively complex robots with higher fidelity sensors and actuators [18]. Task allocation is performed *explicitly* in a strongly-coordinated MRS, where each task is divided into subtasks that are dynamically allocated and re-allocated in response to changing conditions or failure. In the case of robotic vacuuming, the task may be optimally divided according to traffic or dirt distribution rather than floor area. Optimal task allocation is an active area of research [19, 20, 21].

Farinelli et al. [18] reviewed a large number of cooperative MRS and arranged them in a taxonomy according to the level of knowledge sharing, method of coordination and type of organization (Figure 1.4). Using Farinelli's terminology, this thesis is concerned with cooperative MRS where each robot is *aware* of its team members, and the team is *strongly coordinated* using structured communication.

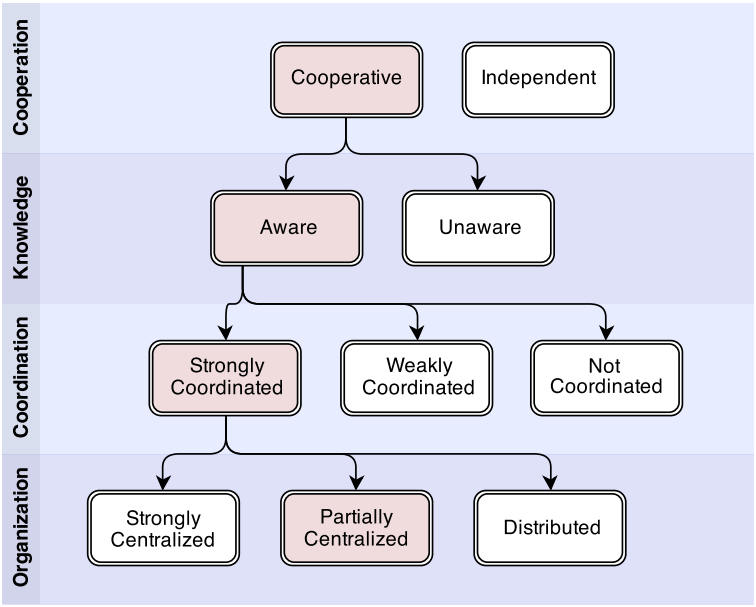


Figure 1.4. Multi-robot system taxonomy: Types of cooperation, levels of knowledge sharing, methods of coordination and types of centralization are shown. Boxes highlighted in red indicate the MRS considered in this research. Image from Farinelli et al. [18]

1.2.2. Multi-Robot Localization

Robust localization estimates are required for a team of robots to navigate an environment and perform cooperative tasks. While relatively precise localization is possible with systems such as real-time kinematic GPS (RTK-GPS) or marker-based tracking (e.g. Vicon), these and similar systems are expensive, require external infrastructure, and only work in limited environments (Section 2.4.3). For rapid MRS deployment in unstructured or mixed indoor/outdoor environments, the localization system needs to be either partially or fully contained within each robot.

If an existing, *a priori*, map is available, it is often possible for robots to localize themselves in the map using on-board sensors (Section 2.4.4). Such localization-only approaches are brittle, however, particularly when the environment can change. Moving the furniture in a house, for example, could make a robot vacuum's *a priori* map unusable. Furthermore, in some environments, such as the site of a natural disaster or the surface of another planet it may not be possible to obtain detailed maps before deployment.

1.2.3. Multi-Robot SLAM

The most flexible way to localize a MRS is with an architecture that enables individual robots to localize themselves at the same time as building a shared global map of the environment. This is the *multi-robot* SLAM (MR-SLAM) problem, a natural extension of the single-robot SLAM problem introduced in Section 1.1.3.

A robust solution to the MR-SLAM problem must be designed into the core of any coordinated MRS that is deployed into unstructured real-world environments, since navigation, task allocation and high-level cooperative behaviors all require mapping and localization information. The MR-SLAM problem is the focus of this research work, and is described in detail in Chapter 3.

Robust real-world MR-SLAM solutions are considerably more complex than the single-robot case due to their distributed nature. MR-SLAM solutions for coordinated MRS are more than a collection of single-robot SLAM algorithms: localization requires each robot to be registered in a single global coordinate frame that is *consistent*, while mapping requires large volumes of sensor data to be fused on-line. In real-world deployments, MR-SLAM solutions must also contend with shared wireless communications that are lossy and have varying latencies and bandwidth. The distributed nature of MR-SLAM introduces several unique problems that require careful architectural considerations, these are described in detail in Chapter 3.

Several problems encountered in single-robot SLAM become more frequent and computationally complex in MR-SLAM. When a group of robots explore an environment together, sensing the environment from slightly different vantage points, the *data association* problem (Section 2.1.1) becomes more difficult, while *loop closures* (Section 2.1.3) occur almost constantly between robots. These loop closures create large sequences of *constraint cycles* (Section 3.3.3), which can cause a combinatorial increase in computational complexity.

In Chapter 4, Chapter 5 and Chapter 6 I describe solutions to many of these MR-SLAM problems. These solutions are combined into a hybrid-decentralized distributed software architecture called *Mapbuilder*, which provides a robust MR-SLAM system for on-line deployments into large-scale urban environments. The Mapbuilder MR-SLAM system is demonstrated in Chapter 7.

1.3. State of the Art

Several MR-SLAM system prototypes have been demonstrated in coordinated MRS over the last two decades; however none have matured sufficiently to be packaged in commercial products or deployed by the military. In 2008 Thrun and Leonard wrote:

> “ *Multi-robot SLAM has benefited greatly from substantial recent interest; nevertheless the existing methods have not yet matured to a level where they can be used by non-experts in the field.* ” [12]

As of 2015, only a handful of consumer products have been released with on-board SLAM solutions. In each case, such as the robotic vacuums discussed previously, only single-robot

SLAM is performed. While many MRS have been deployed in warehouses [22] and at mine sites [23, 24], they typically localize themselves within *a priori* maps using GPS, beacons or visual bar codes. Extensive searches of the literature and the Internet did not find any examples of MR-SLAM deployed outside of research laboratories.

MRS designs are typically complex systems of systems, and they integrate a wide range of technologies. These in turn require engineers with a wide range of skills, from mechanical and electrical design, software engineering, control systems, state estimation and communications, to knowledge of high-level technologies such as perception, path planning, and computer vision. Research in this area is by nature multi-disciplinary, which, when combined with the cost of sensors and robotic hardware, makes MRS and MR-SLAM research expensive. To minimize costs, researchers typically make assumptions and simplifications that limit the scope of their systems, and as a result the state of the art is progressing on several frontiers as outlined in this section.

The current interest in MRS can be illustrated by the MRS-focused workshops held in conjunction with major robotics conferences over the last few years. For example:

- ICRA 2013 “Towards Fully Decentralized Multi-Robot Systems”
- ICRA 2014 “The Centrality of Decentralization in Multi-Robot Systems”
- IROS 2014 “The Future of Multiple-Robot Research”

1.3.1. Academic Research

Many MRS research projects have been described in the literature over the last two decades, e.g. [25, 26, 27, 28, 29, 30, 31]. These projects have demonstrated a range of low-level capabilities such as cooperative SLAM, *exploration, object identification, object tracking* and *object manipulation*. Researchers have demonstrated strategies for optimal *task allocation* in MRS [19] and integrating *heterogeneous* robot types [29, 32, 33]. For a wider review refer to [14, 34].

Research projects can be divided into *off-line* systems that collect sensor data first and process it later, and *on-line* systems that perform MR-SLAM and other tasks while deployed in real-time. In the last two decades, the majority of MRS publications have been validated off-line, thus avoiding computation and communications limitations. Off-line systems are often prototyped using Matlab, for example, which eases implementation and validation. In contrast, the work described in this thesis executes on-line and with limited computational resources.

MRS research can also be divided by the target environment. Indoor laboratory-based research generally avoids many of the irregularities, hostilities and sources of noise found in real-world environments. Indoor research in laboratories can be performed with simpler

research platforms with limited mobility and less expensive sensing: complete packages such as the TurtleBot are available for USD $1,400 [35].

The research described in this thesis is primarily concerned with MRS that can be deployed in outdoor urban environments; environments that are more harsh and unpredictable and that often require more capable and expensive sensors. The minimum cost of a UGV deployed at the MAGIC challenge in 2010, for example, was USD $11,500 (Section 1.3.4.2). Servicing UGVs and repairing hardware failures requires a level of human support that scales with the number of UGVs. Hardware and labor costs combine to ensure that academic research in large-scale outdoor MR-SLAM is expensive.

1.3.2. Localization-Only Systems

This section describes several recent projects that demonstrate MRS behaviors using localization-only approaches. While these projects currently lack MR-SLAM solutions, they demonstrate capabilities that could inspire novel and/or commercially valuable applications if MR-SLAM was integrated.

The "Swarmanoid" project by Dorigo et al. [33] is particularly notable as it combines many low-level capabilities with a team of heterogeneous robots. In 2011 they demonstrated a cooperative MRS that climbs a book case to retrieve a book. Other work shows a flying quadrotor micro aerial vehicle (MAV) overseeing small UGVs while they cooperate to climb a ramp. While impressive, their work uses radio and infra-red sensors to provide localization, while making no attempts to map the environment or perform SLAM.

In [36], Lindsey et al. demonstrated a team of quadrotors MAVs cooperatively building 3-D structures, while in [37], Kushleyev et al. demonstrated a MRS of 20 quadrotors executing trajectories in tight formations (Figure 1.5). In both cases the robots lacked on-board sensing and the MRS relied on external localization by a Vicon system and centralized control. Integrating a *robust* SLAM solution into these systems remains an open research problem. Since 2012 other researchers have demonstrated results with on-board visual SLAM [38], including fusing maps from multiple MAVs as an off-line step [39, 40].

Structured and controlled environments allow localization-only MRS approaches that are simpler and more robust. Several commercial systems have been described in recent years. A logistics MRS designed by Kiva Systems [22], moves storage units around a warehouse while localizing with bar codes printed on the floor (the company was purchased for USD $775 million in 2012). In the mining industry, several companies are using fleets of semi-autonomous trucks to haul ore. These trucks navigate using external localization sensors only, such as RTK-GPS or radio beacons [24].

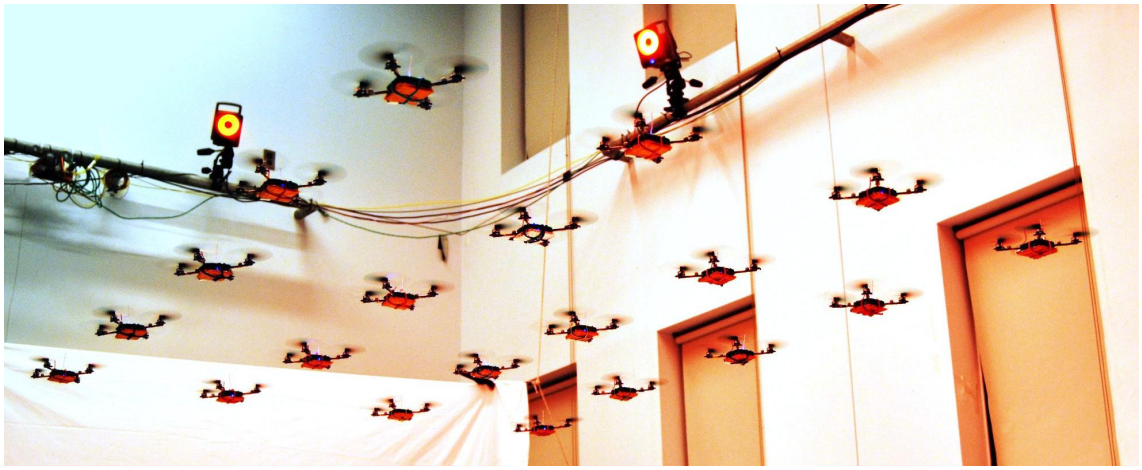


Figure 1.5. MRS of quadrotors MAVs: 20 nano quadrotor MAVs flying in a 5×4 grid formation. Two of the cameras used for external localization can be seen mounted to the wall. Image courtesy of Alex Kushleyev and KMel Robotics.

1.3.3. Multi-Robot Simulations

The hardware and maintenance costs described in Section 1.3.1 have encouraged the development of a multitude of software packages that simulate robots. While simulations have been used to validate many MRS and SLAM algorithms, the fidelity of their models vary widely and none capture the complex dynamics and sensor behaviors observed in the real world.

The most well-cited open-source tools that can simulate a MRS with hundreds of robots is Player and Stage [41, 42]. These tools model the environment in 2-D, however, a deficiency that is addressed by the Gazebo project [43], which simulates 3-D environments including physical interactions. Another notable open source 3-D simulator is USARsim [44], which is used for the RoboCup rescue virtual robot competition [45]. Other MRS simulations such as ARGoS [46] and the EyeBot Simulator [47] are described in the literature. The various simulators differ widely in terms of their fidelity and ability to reproduce accurate real-world physics.

1.3.4. On-line MR-SLAM Systems

In the last two decades, two significant research programs in coordinated MRS have been funded with grants from both government and industry: the Software for Distributed Robotics (SDR) program in 2003, and the Multi-Autonomous Ground-robotics International Challenge (MAGIC) in 2010. Both programs focused on intelligence, surveillance and reconnaissance (ISR) missions.

(a) Howard et al.'s MRS at Fort A.P. Hill, Virginia [29, 50, 51]. Image courtesy of Andrew Howard.

(b) Konolige et al.'s "Centibots" [28, 27, 49]. Image © SRI Int.

Figure 1.6. Two MRS from the 2003 SDR program: Both teams of robots autonomously explored and mapped a 45×25 m^2 indoor environment with minimal human intervention.

1.3.4.1. SDR Program

Funded by DARPA, the SDR program produced two notable MRS with MR-SLAM [48]: Fox et al.'s "Centibots" [28, 27, 49], and Howard et al.'s MRS [29, 50, 51]. Each team designed and assembled a large MRS, shown in Figure 1.6. Their robots autonomously explored and mapped a 45×25 meter indoor environment with minimal human interaction, and then in a second phase they performed an ISR task.

While both Fox et al. and Howard et al. demonstrated an impressive number of robots, 100 and 80 respectively, the on-line MR-SLAM and exploration were only performed by three and four robots, respectively. There was a clear delineation between robots that performed MR-SLAM and robots that performed ISR with localization only [20]. For comparison, in the work presented here, 23 robots perform MR-SLAM while exploring and mapping an area more than 20 times larger.

1.3.4.2. MAGIC Challenge

MAGIC was a USD $1.2 million competition held in November 2010. Five competitors demonstrated multi-robot UGV systems in real-world conditions, including some of the most convincing MRS research to-date. A brief overview of the challenge is given here, refer to Finn et al. for a more in-depth account [52]. The competitors and results are listed in Table 1.1, along with their UGV counts and relevant publications. Figure 1.7 shows a photo of each of the competitors' UGVs.

The challenge motivated the research work I describe in this thesis, while providing most of the datasets used to prepare the results in Chapter 7. I was one of a handful of core team members in the Western Australian MAGIC robot team (WAMbot), having

spent many months working closely with the MRS and UGVs. While I was heavily involved in both software and hardware development, and the final challenge event, my primary contribution was the Mapbuilder hybrid-decentralized and distributed MR-SLAM system that I describe throughout this thesis. More information on the challenge test environments and our team's MRS design is given in Appendix A.

The main objective of the MAGIC challenge was to advance robotic technologies by encouraging the development of MRS that could perform autonomous ISR missions. High-level challenge tasks included identifying and neutralizing both static and moving objects of interest (OOIs) with limited interaction from human operators. The challenge allocated 3.5 hours to explore three phases that became progressively more difficult. Over 200 pages of rules and amendments described a complex set of requirements for the MRS [52].

To accomplish the ISR mission, each MRS required a MR-SLAM system that could provide global localization for a team of five or more UGVs while they explored and mapped a 500×500 meter urban environment. The MR-SLAM system had to provide maps with sufficient detail to enable the UGVs to navigate from unstructured outdoor environments, through doorways and into buildings without GPS. Figure 1.8 shows two photos from the second phase of the challenge, where multiple buildings, large sparse spaces, sand-pits and ditches created a difficult environment for UGVs to navigate. The MR-SLAM system had to be robust, allowing UGVs to continue operating with intermittent communications, while providing both operators and judges with global maps and localization information.

Prior to the challenge, real-world MR-SLAM had not been demonstrated at such a "large scale" (Section 3.2 reviews other systems in the literature and Section 3.3 considers the definition of "large scale"). The results I present in Chapter 7 combine multiple challenge datasets from Team Michigan (TM), the University of Pennsylvania (Penn) and WAMbot to demonstrate real-time MR-SLAM on datasets larger than any previously described.

Place	Team	Abbrev.	UGVs	Publications
1st	Team Michigan (US)	TM	14	[53, 54, 55, 56, 57]
2nd	University of Pennsylvania (US)	Penn	9	[58, 59, 60]
3rd	Recon. & Autonomy For Robots (US)	RASR	8	[61, 62, 63]
4th	Team WAMbot/MAGICian (Aust.)	WAMbot	7	[64, 65, 66, 67, 68]
5th	Cappadocia/ASELSAN (Turkey)	Capp	5	[69]

Table 1.1. MAGIC challenge results: Five teams competed in the final challenge in November 2010. Our team, WAMbot/MAGICian, placed fourth.

1.3.5. Technology Maturity

Using the US Department of Defense and NASA definitions for Technology Readiness Level (TRL) [70], it is interesting to evaluate the maturity of the state of the art in coordinated MRS and MR-SLAM solutions. As noted at the start of this section, there are no systems that are regularly deployed in unstructured real-world environments (TRL 9). Researchers have proven various subsystems, however very few complete "systems of systems" have been demonstrated.

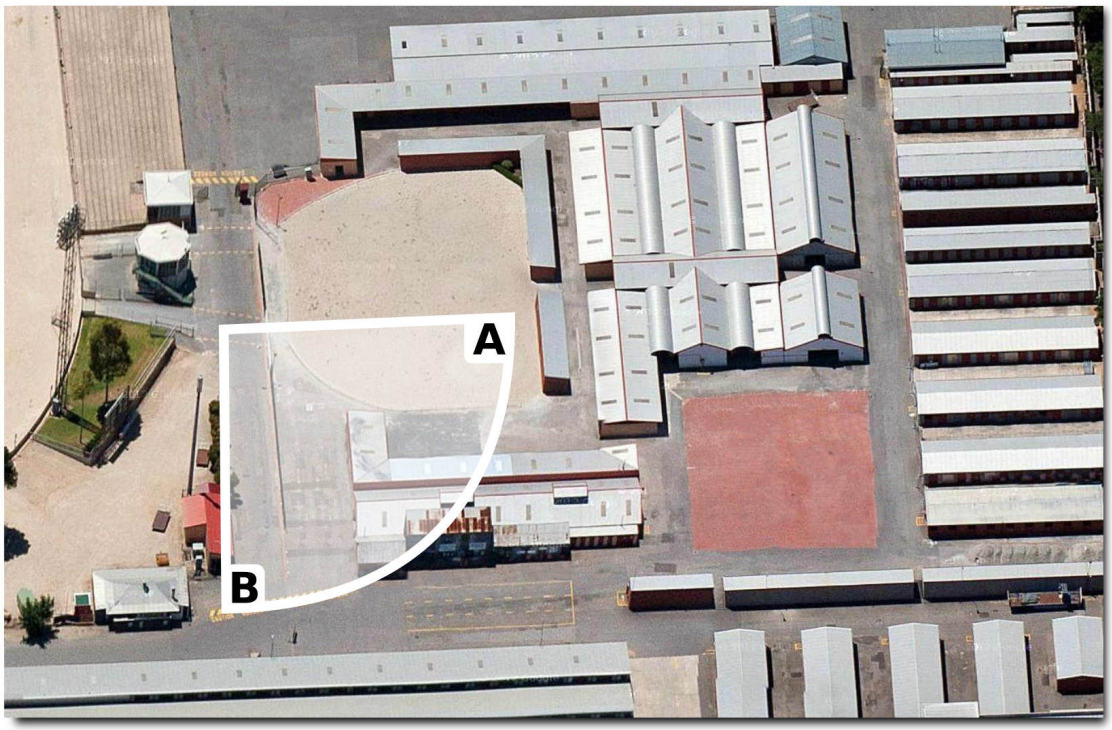
The cooperative MRSs described in this section vary between TRL 4 (components validated in a laboratory) and TRL 5 (components evaluated in relevant environments). Based on results described in publications, the 2003 SDR program could be considered TRL $5\frac{1}{2}$ or TRL 6 (integrated system in relevant environment). While the SDR program validated two complete MRSs, the test environments were indoors and not unlike a laboratory [51].

Using the same classification, the MRS presented at the MAGIC challenge could be considered between TRL 6 and TRL $6\frac{1}{2}$. While the phases of the challenge were representative of increasingly hostile urban environments, the environment was far from an operational one (required for TRL 7 or above). The MRS deployments at the MAGIC challenge raised many interesting questions and suggested that work is still needed before they can be rapidly deployed in real world (TRL 9).

1.4. Potential Multi-Robot Applications

So far, this chapter has introduced a range of capabilities that MRS and MR-SLAM can provide, i.e. the "technology push" for my research work. In this section, I describe several potential real-world applications enabled by these capabilities, i.e. the "market pull" for MR-SLAM systems like Mapbuilder.

Figure 1.7. MAGIC challenge finalists: From left to right, **1st**: Team Michigan (US), **2nd**: University of Pennsylvania (US), **3rd**: Reconnaissance & Autonomy for Small Robots (US), **4th**: Team WAMbot/ MAGICian (Australia), **5th**: Cappadocia/ASELSAN (Turkey).



(a) Oblique aerial photo of Phase 2. This area is approximately 23 % of the total challenge area. Photo courtesy Google Maps, 2011.

(b) Panorama at the start of Phase 2. The field-of-view (A-B) is indicated on the aerial photo above.

Figure 1.8. MAGIC challenge: Many results presented in this thesis are from this 170×150 meter section of Phase 2. The large sparse spaces, sand-pits and drains made UGV mobility, SLAM and navigation difficult.

Many potential applications are simple adaptations of existing single-robot ones; additional robots may complete a task more efficiently or provide valuable redundancy. This is the case for the robotic vacuum cleaner example from Section 1.2, where the benefit from deploying a coordinated team of robots is similar to hiring a team of people.

Potential applications may arise in the future, however, that have no human analog. With communication rates that are orders of magnitude faster than human language, the concept of "cloud robotics" or "hive minds" may create previously unparalleled possibilities [4, 5]. The discussions in this section are restricted to MRS that are designed to assist humans.

1.4.1. Search and Rescue

Natural disasters, such as the 2011 earthquake off Sendai in Japan, and the resulting destruction of buildings can create unstructured environments for which no *a priori* maps exist [71, 72]. A fast Urban Search and Rescue (USAR) response is necessary, however the time available to gather situational awareness, generate maps and search for survivors is often very limited. Damaged or collapsed buildings and rubble piles, may be unstable or have chemical or nuclear leaks that are hazardous to human rescuers.

Many types of USAR robots have been developed for these situations, however they are currently much less capable than humans. In real-world deployments, USAR robots are slowly teleoperated by one or more human operators. Programs such as the recent DARPA Robotics Challenge [73] suggest that increasing autonomy, mobility and manipulation are high priorities in this area. A review of recent USAR robot research is given in [74].

Large-scale disasters could potentially benefit from teams of USAR robots that can be rapidly deployed. Current approaches using teleoperation, however, would require proportionally sized teams of human operators. While studies such as [75] suggest that operators could teleoperate more than a single robot, they are quickly overloaded cognitively. To scale to larger teams of USAR robots, increased autonomy is required, which therefore requires robust MR-SLAM solutions.

In the near future, on-line MR-SLAM solutions will enable USAR robot teams to collaboratively survey disaster sites and provide valuable situational awareness to human first responders [76]. As sensors and robot platforms reduce in size, it is feasible that teams of small, highly-mobile robots could descend into a collapsed building, locating survivors while mapping the rubble to help plan rescue attempts. These systems would require robust MR-SLAM solutions.

1.4.2. Military and Law Enforcement

There are many potential applications for MRS in military and law enforcement. Mobile robots are ideal for tasks such as reconnaissance, surveillance and bomb disposal. These

robots are currently teleoperated individually, and have limited autonomy, much like the USAR robots described in the previous section. The scale and duration of missions are often limited by the endurance of both the robots and their human operators.

It is suggested that the majority of military operations in the future will occur in complex urban environments [77]. Heterogeneous teams of autonomous UGVs and MAVs could be well-suited to these types of ISR missions. Exploration and mapping tasks can be subdivided between multiple robots, while surveillance tasks can benefit from increased coverage and persistence. Multiple robots provide more "eyes" on a target area and redundancy allows robots to be swapped out. The MAGIC challenge described in Section 1.3.4.2 demonstrated these ISR capabilities and more.

Many scenarios in urban warfare can be dangerous to both humans and robots. This danger may be increased if *a priori* maps and GPS localization are relied on exclusively. Explosions can dramatically change the structure and appearance of an environment, while active radio frequency (RF) jamming or spoofing can make GPS localization unreliable or misleading [78, 79]. A robust MR-SLAM solution is ideal for providing localization and mapping for heterogeneous deployments of UGV and MAV teams.

UGV and MAV teams could rapidly explore and map hostile environments that are not in direct line of sight. These over-the-hill ISR missions could provide rich situational awareness that might help to save human lives. There are other applications beyond ISR missions, however the requirements for the underlying MR-SLAM solution remain the same.

1.4.3. Agriculture and Farming

MR-SLAM systems could enable increasing levels of autonomy in agriculture. Innovations in agricultural are being motivated by increasing labor costs and decreasing profit margins. The desire to improve efficiencies has encouraged *precision agriculture,* or micro-managing crops based intra-field variability.

Various technologies have already been applied to increase efficiency and leverage economies of scale. Semi-autonomous drive-by-wire systems have been navigating machinery around fields for the last two decades [80, 81]. These systems rely on external GPS localization, often augmented to centimeter-level accuracy with RTK-GPS. They are typically used in large fields that have been carefully prepared to remove any hazards. These localization-only systems lack the spatial awareness that could allow them to respond to changing situations autonomously. SLAM could provide updated maps of the environment that would enable autonomous tractors to steer around newly discovered obstacles.

In the near-future, MR-SLAM systems could share mapping and localization data between different types of machinery in real-time. This sharing could allow even more autonomous

behaviors to be realized, including close-proximity operations, such as the loading and unloading of materials and produce [82].

The increased availability and decreased cost of fixed-wing and quadrotor MAVs are creating new opportunities for remote sensing of soil and crop conditions [83]. Fleets of MAVs could perform detailed surveys [84, 85], while in the distant future MR-SLAM systems could allow collaborative localization and mapping between these MAVs and machinery on the ground.

Intensive agricultural industries, such as fruit and vegetable growing, often have highly dynamic environments. Humans, machinery and other moving objects, combined with frequently changing terrains, require robust SLAM solutions [86]. In these environments a MR-SLAM solution could enable teams of robots to cooperate while sharing knowledge of the changing environment.

1.4.4. Mining and Resource Extraction

Rising labor costs and the desire to increase safety have encouraged increased automation in the mining industry over the last two decades [87]. In above-ground open-pit mines, semi-autonomous trucks are already using drive-by-wire techniques with external localization, such as RTK-GPS, and lidar for obstacle detection. GPS signals become intermittent when descending into pits and other time-of-flight RF systems, such as Locata [88], are sometimes used. Mining typically involves moving large quantities of rock and ore around, which alters the environment over time. MR-SLAM solutions could allow teams of trucks to share their maps of while providing robust localization despite changing environments and intermittent GPS.

Accurate localization in underground mines is harder since GPS signals are completely denied and rough roads produce noisy wheel-based odometry. Some mine sites use external range-only localization with Ultra Wideband (UWB) beacons [89], however such infrastructure is difficult to install and maintain in harsh underground environments. Single robot SLAM solutions have been investigated in a range of underground applications [90, 23], including high-speed autonomous tramming [24]. A MR-SLAM solution could allow multiple autonomous vehicles to localize accurately relative to each other, while sharing maps. This capability could dramatically increase efficiency if it enabled vehicles to operate in closer proximities. Researchers are also developing mining-specific search and rescue robots [91].

1.4.5. Space Exploration and In-Situ Resource Utilization

NASA has explored the surface of Mars with four rovers over the last two decades. Sojourner landed in 1997, Spirit and Opportunity in 2004, and Curiosity has been exploring

since 2012 [92, 93]. Using cameras, the latter rovers have navigated autonomously for many kilometers [94, 95, 96]. While the scientific data returned from these missions is unprecedented, the launch and operational costs are high. Autonomous functions are limited to minimize risk, and no on-board SLAM is performed.

Multi-robot space exploration has been proposed many times over the last few decades. Leitner provides a review of these MRS in [97]. In [98], Brooks argues that for the same launch cost, a team of tens or hundreds of smaller robots could maximize surface coverage. The lower cost per robot, restricted life expectancy, and large communications delays all favor using more autonomy instead of teleoperation [99]. MR-SLAM could help to enable increased autonomy.

Decreasing launch costs and increasing autonomy in MRS may enable a range of space-based industries in the near future. In-situ resource utilization could become commonplace, where water ice and rare elements are mined from the surface of asteroids, comets, the Moon and perhaps even Mars. While these environments are hostile to humans, teams of robots could mine resources cooperatively. Without the Earth-bound GPS network, or real-time communications to Earth, these MRS will require robust MR-SLAM solutions.

1.5. Notation

Throughout this thesis I use the mathematical notation from Bishop's seminal text [100], and for state estimation the notation from Bar-Shalom et. al. [101]. A summary of the notation used is given in Table 1.2.

1.6. Publications

Contributions described in this thesis have been peer reviewed and published in the Journal of Field Robotics:

- A. Boeing, M. Boulton, T. Bräunl, B. Frisch, S. Lopes, A. Morgan, F. Ophelders, S. Pangeni, **R. Reid**, K. Vinsen, N. Garel, C. S. Lee, M. Masek, A. Attwood, M. Fazio, and A. Gandossi, "WAMbot: Team MAGICian's entry to the Multi Autonomous Ground-robotic International Challenge 2010," *Journal of Field Robotics*, 2012.

They have also been published in several peer-reviewed conference papers:

- A. Boeing, M. Boulton, T. Bräunl, B. Frisch, S. Lopes, A. Morgan, F. Ophelders, S. Pangeni, **R. Reid**, K. Vinsen, N. Garel, C. S. Lee, M. Masek, A. Attwood, M. Fazio, and A. Gandossi, "Team MAGICian," in *Land Warfare Conference*, 2010.

- **R. Reid** and T. Bräunl, "Large-scale multi-robot mapping in MAGIC 2010," in *IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, 2011.

- A. Boeing, T. Bräunl, **R. Reid**, A. Morgan, and K. Vinsen, "Cooperative Multi-Robot Navigation and Mapping of Unknown Terrain," in *2011 IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, 2011.

- **R. Reid**, A. Cann, C. Meiklejohn, L. Poli, and T. Bräunl, "Cooperative Multi-Robot Navigation, Exploration, Mapping and Object Detection with ROS," in *IEEE Intelligent Vehicles (IV)*, 2013.

While these publications are co-authored, any SLAM-related sections are easily separated as my individual contributions. My contributions to these publications are superseded by this thesis, which expands upon each considerably.

1.7. Thesis Structure

This thesis is comprised of eight chapters and two appendices:

Chapter 1. Introduction: This introductory chapter motivates my research work in MR-SLAM. It summarizes key trends, introduces the SLAM and multi-robot SLAM problems, reviews current capabilities and introduces several potential application areas.

Chapter 2. Review: Simultaneous Localization and Mapping: This chapter describes the single-robot SLAM problem and the main approaches for solving it with a focus on pose-graph SLAM. It also describes map parameterizations, sensors and their models.

Chapter 3. Review: Large-Scale Multi-Robot SLAM: This chapter introduces the main MR-SLAM problems and key system architectures. It reviews 15 years of MR-SLAM literature and summarizes large-scale systems that have been demonstrated.

Chapter 4. Hybrid-Decentralized and Distributed Multi-Robot SLAM: This chapter describes research contributions that form an on-line MR-SLAM system. It includes the system architecture and design, requirements, assumptions, conceptual and logical design.

Chapter 5. Efficient Occupancy Gridmap Fusion and Matching: This chapter describes two algorithmic contributions: one parallelizes the fusion of submaps into global gridmaps, the other parallelizes correlative submap matching to search for multimodal constraints.

Chapter 6. Robust Multimodal Pose Graph Optimization: This chapter describes continuous mode blending optimization (COMBO), a contribution that allows robust multimodal Gaussian constraints to be optimized using existing pose graph techniques.

Chapter 7. Results: This chapter presents results demonstrating my research, including: on-line hybrid-decentralized and distributed MR-SLAM, real-time large-scale MR-SLAM and multimodal pose graph optimizations. The chapter closes with an in-depth discussion.

Chapter 8. Conclusion: This chapter summarizes the key aspects of the MR-SLAM problem that my research addressed. It highlights my contributions and their significance, describes directions for future work, and considers the broader implications of my work.

Appendix A. MRS Architecture and UGV Design: This appendix describes three MRS and UGV front-end designs (experimental setups) that were used to demonstrate research contributions.

Appendix B. MAGIC Challenge Datasets: This appendix describes the various datasets recorded at the MAGIC challenge. These are the test environments used to demonstrate my research contributions.

Symbol	Meaning
$\mathbf{x} = (x_1, x_2, \ldots, x_N)^T$	The system state represented as a vector (bold type)
$\mathbf{x}_t \simeq \mathbf{x}$ and $\mathbf{x}_{t=0} \simeq \mathbf{x}_0$	Subscripts are omitted when inferred by context
$\mathbf{A}$, $\mathbf{\Sigma}$	Matrices (bold type)
$\mathrm{diag}(\mathbf{x})$	Matrix with vector $\mathbf{x}$ along the diagonal
$\mathbf{x}$	The exact or actual value of $(x_1, x_2, \ldots, x_N)^T$
$\hat{\mathbf{x}}_t$	An estimate for $\mathbf{x}$ at time t
$\bar{\mathbf{x}}_t$	A prediction for $\mathbf{x}$ based on $\hat{\mathbf{x}}_{t-1}$
$\mathbf{y} = (y_1, y_2, \ldots, y_M)^T$	A measurement of the system state
$\tilde{\mathbf{y}} = \mathbf{y} - \bar{\mathbf{y}}$	The residual in the measurement prediction $\bar{\mathbf{y}}$
$p(\mathbf{r} \mid \mathbf{z})$	Probability distribution of $\mathbf{r}$ given $\mathbf{z}$
$\mathcal{N}(\mathbf{u}\|\hat{\mathbf{u}}, \mathbf{\Sigma}_u)$	Gaussian distribution with mean $\hat{\mathbf{u}}$ and covariance $\mathbf{P}_u$
$\|\mathbf{e}\|^2_{\mathbf{\Sigma}}$	Mahalanobis distance $D^2 = \mathbf{e}^T\mathbf{P}^{-1}\mathbf{e}$
$\mathbb{R}^2$	2-D Euclidean space
$\mathbb{SE}(2)$	Special Euclidean group of 2-D transforms (Section 2.2)
$\mathfrak{se}(2)$	Lie algebra corresponding to $\mathbb{SE}(2)$ (Section 2.2)
$\oplus$ and $\ominus$	Pose composition operators (Section 2.2)

Table 1.2. Mathematical notation

2 Review: Simultaneous Localization and Mapping

This chapter describes the SLAM problem, and reviews the main approaches for solving it. Section 2.1 introduces the problem, while Section 2.2 presents the mathematical transformations necessary to manipulate robot poses. In Section 2.3 I discuss mapping, including various map parameterizations, sensors and sensor models, while in Section 2.4 I explore localization, including robot motion models, odometry and various global and relative localization sensors. Section 2.5 reviews the main probabilistic approaches for solving the SLAM problem, with a focus on graph-based SLAM and submapping techniques. Section 2.6 concludes by summarizing key aspects of the problem.

2.1. Problem Statement

Navigating an unknown environment requires a solution to the SLAM problem. The problem has two parts; the first addresses the question "where am I?" (localization), while the second addresses the question "what does the world look like?" (mapping). The two questions are intrinsically coupled by probabilistic uncertainties and they must be answered simultaneously. The problem can be expressed in the metaphorical chicken-and-egg sense; for a robot to know its location it needs an accurate map of its environment, however to build an accurate map a robot need to know its location in the environment.

At the heart of the SLAM problem is the fact that real-world sensors are noisy, thus measurements of either the robot's pose or the environment are subject to *uncertainty* and *bias*. We often characterize this measurement noise with Gaussian distributions, which allows the robot's pose and environment map to be jointly parameterized using multivariate normal distributions. Thus the chicken-and-egg coupling between localization and mapping is captured probabilistically with estimates of mean and covariance matrices.

This probabilistic representation is typically maintained using a Bayes filter, such as the Extended Kalman Filter (EKF) [101]. The EKF was introduced in the *state estimation* literature in the 1960s [102]; its application to the SLAM problem is described in detail in Section 2.5.3. Using state estimation approaches to describe the SLAM problem, however, trivializes many important aspects as simple *data association* problems. This chapter describes these important aspects, including data association, along with various solutions that have been described in the literature.

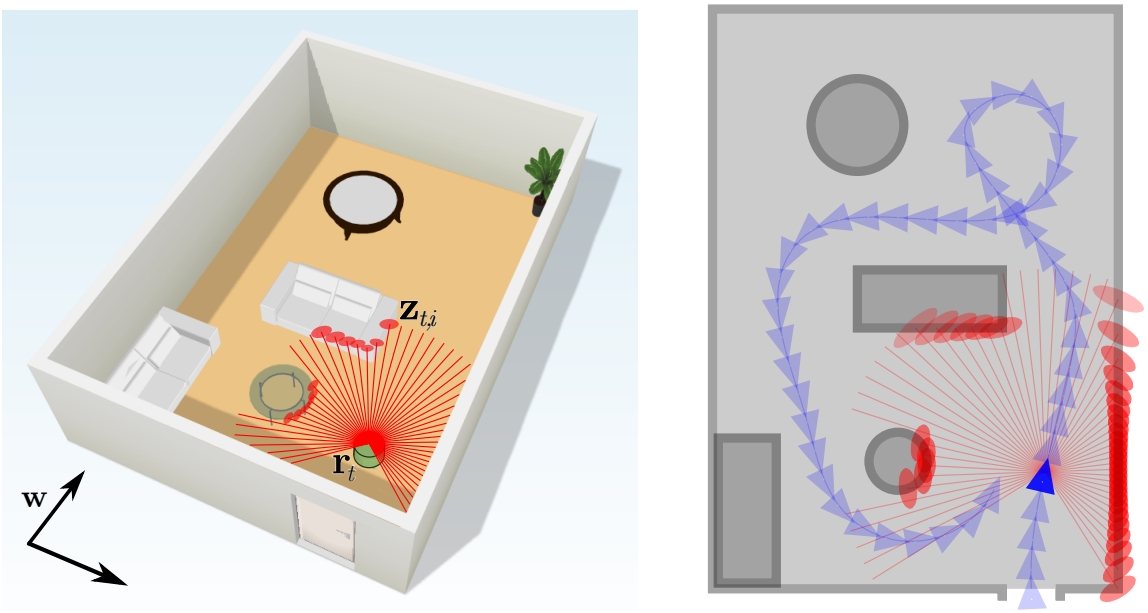
2.1.1. Data Association

Data association is a key aspect of the SLAM problem [103, 104]. In its simplest form it estimates pairings between sensor measurements and objects in the environment. While a typical SLAM sensor, such as a lidar scanner, can take thousands of measurements of the environment per second, they generally produce insufficient information to solve for these pairings directly. Figure 2.1 shows an example scan from a lidar scanner; while each scan may capture detailed range and bearing measurements, they only hint at the shape of the environment since no information is available that explicitly pairs each laser pulse to the object it was reflected from.

With an ever-present uncertainty in robot pose, we typically do not know the exact direction a sensor is pointing. This uncertainty, coupled with our inability to identify which object each measurement comes from, creates uncertainties and potential ambiguities in the measurement-to-object pairings, –or– the data association problem.

2.1.2. Full SLAM

The "full" SLAM problem aims to jointly estimate the entire history of a robot's poses and a map of the environment while considering the data associations pairing each measurement to objects in the map. Uncertainties in data associations, however, make the "full" SLAM problem combinatorial in nature and computationally intractable— consider that any update to either the robot's pose or the map requires every data association to be re-evaluated. Section 2.5.2 explores the "full" SLAM problem in more detail. It is NP-complete [7, 8] and all practical algorithms described in the literature exploit



(a) Perspective view **(b) 2-D top view**

Figure 2.1. Example problem: A robotic vacuum cleaner drives around a room containing two couches and two tables. In (a) measurements $\mathbf{z}_{t,i}$ from a lidar scanner are shown extending from the robot in red. Each lidar return measures the mean range to the objects overlapped by the corresponding red ellipse. In (b) the robot's exact path is shown in blue. The lidar measurements $\mathbf{z}_{4,i}$, from $t = 4$, are shown in red.

approximations and heuristics to run in polynomial time. Since it is too expensive to re-evaluate every data association decision, many algorithms make initial decisions, and then irreversibly "bake" them into the current estimate to achieve polynomial time complexity and maintain on-line execution.

In Chapter 4 I describe a submapping technique that "bakes" the most certain data associations into the map, while retaining the ability to re-evaluate the most uncertain of data associations on-line and in polynomial time. These highly-uncertain data associations are often called *loop closures*, and the ability to re-evaluate them becomes important when solving the multi-robot SLAM problem.

2.1.3. Loop Closures

A robot that follows a long path, while measuring its progress with on-board sensors, will accumulate pose uncertainty due to sensor noise. If the robot keeps visiting new places, the pose uncertainty will continue to grow without bound. When the robot re-visits a place it has previously been, a loop closure events occurs [103, 104]. Loop closures are a type of data association that are crucial to SLAM, since they allow accumulated errors to be corrected and robot pose uncertainties to be reduced.

Candidate loop closures are typically *detected* and *verified* over repeated observations, and then *integrated* into the current SLAM estimate. Large areas of a SLAM map may be altered by loop closure events, and some SLAM algorithms handle these changes more effectively than others (Section 2.5).

In multi-robot SLAM, loop closures occur more frequently since robots that are driving in the same environment with overlapping sensors are likely to identify their pose in the other robots' maps. In some MRS architectures (Section 3.2.2) this can create hundreds of loop closures per minute, while potentially causing widespread structural changes to the map. This research work shows how high rates of loop closures and large map changes can be accommodated efficiently and in real-time.

2.1.4. Example Problem

To illustrate the SLAM problem in the context of mobile robots I introduce an example problem that is used for the remainder of this chapter. The problem, shown in Figure 2.1, is inspired by our Neato XV-25 robotic vacuum cleaner.

A small mobile robot drives around a furnished room. The floor is relatively flat and by using a planar approximation the robot's pose is represented by the 2-D translation, $[x_r, y_r]^T$, and angular heading, ϕ_r. In 2-D Euclidean space the full pose is given by $\mathbf{r} = [x_r, y_r, \phi_r]^T$. As the robot drives through the environment, sensors on the wheels measure its *odometry*, $\mathbf{u}_t^{t-1}$, which describe the incremental changes in the robot's pose between successive time-steps, t. The robot uses a scanning lidar sensor to navigate around several pieces of furniture in the room. At each time-step the lidar records M_t range and bearing measurements, $\mathbf{z}_{t,i}^t = [r, \theta]^T$, to objects that are within the lidar's range.

Many practical details are ignored in this example by assuming that the sensors are calibrated and that their acquisition is instantaneous and synchronized. The robot produces a discretized representation of its journey by recording the set of odometry measurements $\mathbf{u}$ and lidar scans $\mathbf{z}$, at T regular time intervals. The mathematical operators required to manipulate these poses and measurements are introduced next.

2.2. Poses and Transformations

While the mathematics required to manipulate robot poses (coordinate frames) and odometry (rigid-body transformations) can be simplified into a set of formula, it is more flexible and robust to define them using a Lie group, $\mathbb{SE}(n)$, and its associated Lie algebra, $\mathfrak{se}(n)$. Lie groups are frequently used in robotics for 2-D or 3-D problems in $\mathbb{R}^n$ Euclidean space, where $n = 2$ or $n = 3$, respectively. Refer to Ivancevic, [105], Murray et al., [106], or Fulton and Harris, [107], for in-depth treatments of these mathematical representations.

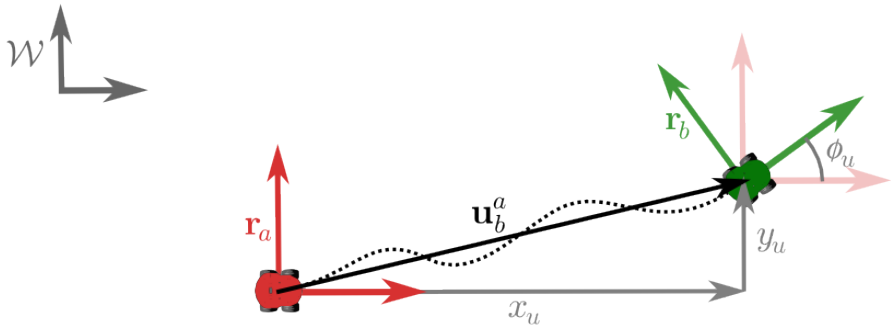


Figure 2.2. 2-D rigid-body transformations: A robot drives from pose $\mathbf{r}_a$ to $\mathbf{r}_b$. During this drive it measures the incremental pose change, or odometry, $\mathbf{u}_b^a = [x_u, y_u, \phi_u]^T$. Both poses describe local coordinate frames in the world frame $\mathcal{W}$, while $\mathbf{u}_b^a$ describes a rigid-body transformation.

My research work is primarily concerned with mobile robots moving over large terrains that are approximately flat (locally-planar). This motivates us to approximate robot poses in a 2-D Euclidean space using the Lie group where $n = 2$, or the special Euclidean group $\mathbb{SE}(2)$, and its associated algebra, $\mathfrak{se}(2)$. In the following sections I introduce two transform operators, $\oplus$ and $\ominus$, that are used throughout my thesis to manipulate robot pose and odometry. These operators were first defined by Smith et al. in [108].

2.2.1. Rigid-Body Transformations

2.2.1.1. Rigid-Body Transformations in $\mathbb{R}^2$

Referring to Figure 2.2, a robot starts at pose $\mathbf{r}_a$ and drives to pose $\mathbf{r}_b$, producing an incremental odometry measurement $\mathbf{u}_b^a = [x_u, y_u, \phi_u]^T$. An observer standing at $\mathbf{r}_a$ would see a change in the robot's pose of x_u meters forward, y_u meters to the left and a rotation of ϕ_u degrees anti-clockwise (using a right-handed coordinate system). This incremental motion, $\mathbf{u}_b^a$, is a rigid-body transformation represented as an element of $\mathfrak{se}(2)$.

Every measurement in Euclidean space is made with respect to a *base coordinate frame*. The odometry transform above, $\mathbf{u}_b^a$, is made with respect to the pose $\mathbf{r}_a$. Throughout this thesis superscripts, like 'a' in $\mathbf{u}_b^a$, are used to indicate the base frame. In general, transform $\mathbf{u}_b^a$ is read as *"the transform to frame* $\mathbf{r}_b$*, as measured from the base frame* $\mathbf{r}_a$*."*

While I have made a clear distinction between robot poses and odometry, they are in fact both rigid-body transforms in $\mathbb{R}^2$. By defining a fixed world coordinate frame, $\mathcal{W}$, the pose at time t can be written as a relative transform, $\mathbf{r}_t^{\mathcal{W}}$, similar to odometry transforms. Here $\mathbf{r}_t^{\mathcal{W}}$ is the transform to pose $\mathbf{r}_t$, as measured from the world frame $\mathcal{W}$. For clarity the superscript is dropped since robot poses are only ever defined with respect to the global frame.

The 3-vector representation of a transform, $\mathbf{u}_b^a = [x_u, y_u, \phi_u]^T$, is *minimal* since its components directly define the 3 degrees of freedom of any Euclidean transform in $\mathbb{R}^2$. This compact representation as an element of $\mathfrak{se}(2)$ is not easy to manipulate, however, which motivates the use of the Lie group $\mathbb{SE}(2)$.

2.2.1.2. Rigid-Body Transformations in $\mathbb{SE}(2)$ and $\mathfrak{se}(2)$

Any rigid-body transformation in $\mathbb{R}^2$, such as the odometry transform $\mathbf{u}_b^a = [x_u, y_u, \phi_u]^T \in \mathfrak{se}(2)$ can be represented by a unique 3×3 matrix, $\mathbf{T}_b^a \in \mathbb{SE}(2)$ [106]. We use the *exponential map* to convert $\mathbf{u}_b^a \longmapsto \mathbf{T}_b^a$, which is expressed as $\mathbf{T}_b^a = \exp(\mathbf{u}_b^a)$. The transform can be broken into a translational component, $\mathbf{t}_b^a = [x_u, y_u]^T$, and a 2×2 rotation matrix $\mathbf{R}_b^a$, which belongs to the special orthogonal group, $\mathbb{SO}(2)$:

$$\mathbf{R}_b^a = \begin{bmatrix} \cos(\phi_u) & -\sin(\phi_u) \\ \sin(\phi_u) & \cos(\phi_u) \end{bmatrix} \quad \in \mathbb{SO}(2) \tag{2.1}$$

The transform $\mathbf{u}_b^a$ maps into its $\mathbb{SE}(2)$ representation $\mathbf{T}_b^a$:

$$\mathbf{T}_b^a = \exp(\mathbf{u}_b^a) = \left[\begin{array}{c|c} \mathbf{R}_b^a & \mathbf{t}_b^a \\ \hline 0 & 1 \end{array}\right] = \begin{bmatrix} \cos(\phi_u) & -\sin(\phi_u) & x_u \\ \sin(\phi_u) & \cos(\phi_u) & y_u \\ 0 & 0 & 1 \end{bmatrix} \quad \in \mathbb{SE}(2) \tag{2.2}$$

Similarly the robot pose, $\mathbf{r}_a = [x_a, y_a, \phi_a]^T$, maps into its $\mathbb{SE}(2)$ representation $\mathbf{T}_a^{\mathcal{W}}$:

$$\mathbf{T}_a^{\mathcal{W}} = \exp\left(\mathbf{r}_a^{\mathcal{W}}\right) = \left[\begin{array}{c|c} \mathbf{R}_a^{\mathcal{W}} & \mathbf{t}_a^{\mathcal{W}} \\ \hline 0 & 1 \end{array}\right] = \begin{bmatrix} \cos(\phi_a) & -\sin(\phi_a) & x_a \\ \sin(\phi_a) & \cos(\phi_a) & y_a \\ 0 & 0 & 1 \end{bmatrix} \quad \in \mathbb{SE}(2) \tag{2.3}$$

For any well-formed matrix transform $\mathbf{T}_b^a \in \mathbb{SE}(2)$ the inverse mapping, or the *logarithmic map,* is trivially defined: $\mathbf{T}_b^a \longmapsto \mathbf{u}_b^a$, where $\mathbf{u}_b^a = \log(\mathbf{T}_b^a) \in \mathfrak{se}(2)$.

2.2.2. Transform Compositions

2.2.2.1. Transform Compositions in $\mathbb{SE}(2)$ and $\mathfrak{se}(2)$

Following the notation of Smith et al. in [108], I define the *transformation composition* operator '⊕', which is used extensively throughout this thesis. Referring to Figure 2.2, a robot moving from a known pose, $\mathbf{r}_a$, to a new pose, $\mathbf{r}_b$, produces an odometry measurement, $\mathbf{u}_b^a$. We can calculate the new pose, $\mathbf{r}_b$, using transform composition:

$$\mathbf{r}_b = \mathbf{r_a} \oplus \mathbf{u}_b^a \qquad \in \mathfrak{se}(2) \tag{2.4}$$

While this appears like a 3-vector addition, composition is *not* a linear operation, an important fact that affects every aspect of SLAM described throughout this thesis. Using the $\mathbb{SE}(2)$ matrix form, transforms can be composed easily using matrix multiplication[1]:

$$\mathbf{T}_b^{\mathcal{W}} = \mathbf{T}_{\not{\emptyset}}^{\mathcal{W}} \mathbf{T}_b^{\not{\emptyset}} \qquad \in \mathbb{SE}(2) \tag{2.5}$$

Using this expression, we can compose the transforms in minimal $\mathfrak{se}(2)$ form by chaining the exponential and logarithmic maps. Substituting the $\mathbb{SE}(2)$ matrix forms from Equation 2.2 and Equation 2.3, the new pose can be expressed in its minimal form [109, 110]:

$$\mathbf{r}_b = \mathbf{r_a} \oplus \mathbf{u}_b^a = \log\left[\exp(\mathbf{r_a}) \cdot \exp(\mathbf{u}_b^a)\right] = \begin{bmatrix} x_a + \cos(\phi_a)\, x_u - \sin(\phi_a)\, y_u \\ y_a + \sin(\phi_a)\, x_u + \cos(\phi_a)\, y_u \\ \phi_a + \phi_u \end{bmatrix} \in \mathfrak{se}(2) \tag{2.6}$$

The composition operator extends to large kinematic chains of transforms. In Figure 2.3 the final pose, $\mathbf{r}_3$, can be composed by chaining first pose and subsequent odometry measurements: $\mathbf{r}_3^{\mathcal{W}} = \mathbf{r}_{\not{\emptyset}}^{\mathcal{W}} \oplus \mathbf{u}_{\not{1}}^{\not{\emptyset}} \oplus \mathbf{u}_{\not{2}}^{\not{1}} \oplus \mathbf{u}_3^{\not{2}}$. Transform compositions are *non-commutative*, justifying a rigorous use of subscripts.

2.2.2.2. Transform Inversion in $\mathbb{SE}(2)$ and $\mathfrak{se}(2)$

The *transform inversion* operator '$\ominus$', provides the transform inverse $\mathbf{u}_a^b = \ominus \mathbf{u}_b^a$. If the forward transform is given by $\mathbf{T}_b^a = \exp(\mathbf{u}_b^a) \in \mathbb{SE}(2)$, then the inverse transform is conveniently given by the matrix inverse:

$$\exp(\ominus \mathbf{u}_b^a) = [\mathbf{T}_b^a]^{-1} = \left[\begin{array}{c|c} \mathbf{R}_b^a & \mathbf{t}_b^a \\ \hline 0 & 1 \end{array}\right]^{-1} = \left[\begin{array}{c|c} [\mathbf{R}_b^a]^T & -[\mathbf{R}_b^a]^T \mathbf{t}_b^a \\ \hline 0 & 1 \end{array}\right] \in \mathbb{SE}(2) \tag{2.7}$$

Similar to Equation 2.6, by chaining the exponential and logarithmic maps, the inverse transform $\ominus \mathbf{u}_b^a$, can be represented in its minimal form:

$$\ominus \mathbf{u}_b^a = \log\left([\exp(\mathbf{u}_b^a)]^{-1}\right) = \begin{bmatrix} -\cos(\phi_u)\, x_u - \sin(\phi_u)\, y_u \\ \sin(\phi_u)\, x_u - \cos(\phi_u)\, y_u \\ -\phi_u \end{bmatrix} \in \mathfrak{se}(2) \tag{2.8}$$

2.2.2.3. Transform Subtraction in $\mathbb{SE}(2)$ and $\mathfrak{se}(2)$

Referring to Figure 2.2, if a robot drives a known distance, $\mathbf{u}_b^a$, arriving at the known pose, $\mathbf{r}_b$, its initial pose, $\mathbf{r}_a$, can be calculated. From Equation 2.5: the relationship $\mathbf{r}_b = \mathbf{r_a} \oplus \mathbf{u}_b^a$,

[1] For rigor in large compositions, the designators for the intermediate frames can be struck-out in matching pairs, e.g. robot pose frame $\not{\emptyset}$ in $\mathbf{T}_b^{\mathcal{W}} = \mathbf{T}_{\not{\emptyset}}^{\mathcal{W}} \mathbf{T}_b^{\not{\emptyset}}$.

represented in $\mathbb{SE}(2)$ is $\mathbf{T}_b^{\mathcal{W}} = \mathbf{T}_a^{\mathcal{W}}\mathbf{T}_b^a$. Multiplying both sides by the inverse transform, $[\mathbf{T}_b^a]^{-1}$, and rearranging gives $\mathbf{T}_a^{\mathcal{W}}$:

$$\mathbf{T}_a^{\mathcal{W}} = \mathbf{T}_b^{\mathcal{W}}\,[\mathbf{T}_b^a]^{-1} \qquad \in \mathbb{SE}(2) \tag{2.9}$$

Taking the log of both sides, similar to Equation 2.6, and substituting Equation 2.7, maps the unknown initial pose, $\mathbf{T}_a^{\mathcal{W}} = \exp(\mathbf{r}_a)$, back to the minimal form:

$$\begin{aligned}
\mathbf{r}_a &= \mathbf{r_b} \oplus (\ominus \mathbf{u}_b^a) \simeq \mathbf{r_b} \ominus \mathbf{u}_b^a \\
&= \log\left[\exp(\mathbf{r_b}) \cdot [\exp(\mathbf{u}_b^a)]^{-1}\right] \\
&= \begin{bmatrix} x_b - \cos(\phi_b - \phi_u)\, x_u + \sin(\phi_b - \phi_u)\, y_u \\ y_b - \sin(\phi_b - \phi_u)\, x_u - \cos(\phi_b - \phi_u)\, y_u \\ \phi_b - \phi_u \end{bmatrix} \qquad \in \mathfrak{se}(2)
\end{aligned} \tag{2.10}$$

We can express *transform subtraction* with the notation $\mathbf{r}_a = \mathbf{r}_b \ominus \mathbf{u}_b^a$, without loss of generality. As with other operations in $\mathfrak{se}(2)$, this is *not* a linear operation.

2.2.3. Homogeneous Coordinate Transforms

To build maps, measurements made by a moving robot need to be transformed into common coordinate frames so they can be fused. It is considerably easier, and less error-prone, to transform measurements using $\mathbb{SE}(2)$ representations and matrix operations. Measurements are first converted into their *homogeneous form* [111], by concatenating each vector with a '1', and then transformed by post-multiplying with a $\mathbb{SE}(2)$ transform.

A robot at pose $\mathbf{r}_a$ takes measurements $\mathbf{z}_{a,i}^a = [x_i, y_i, 1]^T$, represented in homogeneous coordinates. After driving a small distance, measured by odometry to be $\mathbf{T}_b^a = \exp(\mathbf{u}_b^a) \in \mathbb{SE}(2)$, the robot takes additional measurements, $\mathbf{z}_{b,i}^b$. The two sets of measurements can be compared by transforming the second set into the first coordinate frame, $\mathbf{r}_a$, with:

$$\mathbf{z}_{b,m}^a = \mathbf{T}_b^a \mathbf{z}_{b,i}^b \tag{2.11}$$

When using $\mathbb{SE}(2)$ transforms the homogeneous coordinates will remain normalized and the trailing '1' is truncated when no longer required.

It follows that measurements can also be transformed through chains of rigid-body transformations. Using the example in Figure 2.3, a robot starts at the known pose $\mathbf{r}_0$ and makes 3 short drives, measured to be $\mathbf{u}_1^0$, $\mathbf{u}_2^1$ and $\mathbf{u}_3^2$. After each drive it records measurements to the nearby trees, $\mathbf{z}_{1,i}^1$, $\mathbf{z}_{2,i}^2$ and $\mathbf{z}_{3,i}^3$, respectively. Using transform composition the final pose, $\mathbf{r}_3$, can be formed by chaining the first pose and subsequent odometry

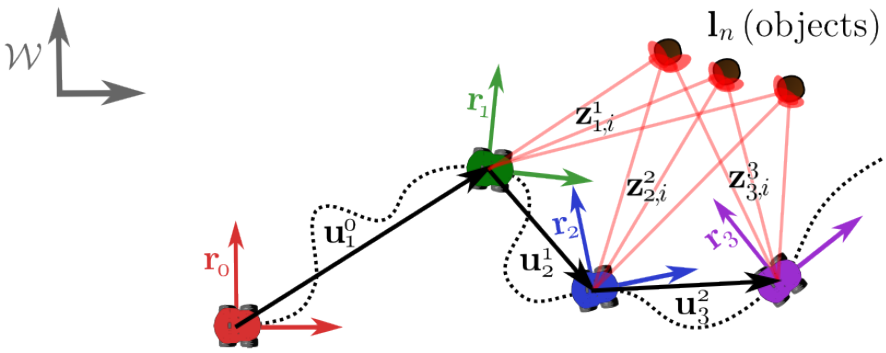


Figure 2.3. Measurement transformations using $\mathbb{SE}(2)$: A robot drives from pose $\mathbf{r}_0$ to $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{r}_3$ (local coordinate frames), measuring odometry transform $\mathbf{u}^0_1$, $\mathbf{u}^1_2$ and $\mathbf{u}^2_3$. From each pose, $\mathbf{r}_t$, it makes measurements, $\mathbf{z}^t_{t,i}$, to the objects in the environment.

measurements: $\mathbf{r}^{\mathcal{W}}_3 = \mathbf{r}^{\mathcal{W}}_0 \oplus \mathbf{u}^0_1 \oplus \mathbf{u}^1_2 \oplus \mathbf{u}^2_3$. In a similar manner, substituting the $\mathbb{SE}(2)$ matrix forms from Equation 2.2 and Equation 2.3, the measurements taken from the 3 poses can be transformed into the global frame, $\mathcal{W}$, via:

$$\begin{aligned}
\mathbf{z}^{\mathcal{W}}_{1,i} &= \mathbf{T}^{\mathcal{W}}_0 \mathbf{T}^0_1 \mathbf{z}^1_{1,i} \\
\mathbf{z}^{\mathcal{W}}_{2,i} &= \mathbf{T}^{\mathcal{W}}_0 \mathbf{T}^0_1 \mathbf{T}^1_2 \mathbf{z}^2_{2,i} \\
\mathbf{z}^{\mathcal{W}}_{3,i} &= \mathbf{T}^{\mathcal{W}}_0 \mathbf{T}^0_1 \mathbf{T}^1_2 \mathbf{T}^2_3 \mathbf{z}^3_{3,i}
\end{aligned}$$

More generally, if a robot starts from an initial pose, such as $\mathbf{r}_0 = [0,0,0]^T$, and explores its environment while recording odometry $\mathbf{u}^{t-1}_t$ and measurements $\mathbf{z}^t_{t,i}$, it can transform all measurements from time τ into a global frame $\mathcal{W}$, via:

$$\mathbf{z}^{\mathcal{W}}_{\tau,m} = \mathbf{T}^{\mathcal{W}}_0 \left(\prod_{t=1}^{\tau} \mathbf{T}^{t-1}_t \right) \mathbf{z}^{\tau}_{\tau,i} \tag{2.12}$$

Using Equation 2.12 it is possible to build a global map by plotting all measurements $\mathbf{z}^{\mathcal{W}}_{\tau,Ii}$ for $\tau = 1...T$. However, without SLAM the noisy odometry will inevitably cause the map to diverge as it grows. Figure 2.8 demonstrates this on page 46.

2.3. Environment Mapping and Parameterization

To navigate robustly through unknown and unstructured environments, robots need to be able to perceive and model their environment. This motivates us to build SLAM systems that can build accurate maps using only on-board sensors. Before showing how SLAM can incorporate measurements from noisy sensors and odometry, it is important to characterize the noise, its effects and potential sensor failure modes. This section describes the *mapping* part of SLAM; the various sensors and parameterizations that are used to measure and

represent the environment. In probabilistic form the entire map $\mathbf{m}$, given by *all* of the robot poses, $\mathbf{r}$, and *all* measurements, $\mathbf{z}$, is expressed as:

$$p(\mathbf{m} \mid \mathbf{r}, \mathbf{z}) \tag{2.13}$$

2.3.1. Environment Mapping Sensors

Sensors that detect the environment from mobile robots typically provide 1, 2 or 3 degrees of freedom (DOF) measurements that can be either *range-only*, *bearing-only* or combinations of both. Sensors that directly observe and estimate the environment are typically more useful in mobile robotics since they do not require robot motion. Common sensors types include:

Sonar sensors: such as ultrasonic transducers were frequently used by SLAM researchers in the 1980s and 1990s. Either servo-mounted or fixed arrays of transducers were often fitted to mobile robots and were considered inexpensive [11, 110]. Using echo-location techniques, much like bats or dolphins, ultrasonic transducers produce range and bearing measurements. However, the primary -3 dB lobe in their sensitivity pattern is often as wide as 30 degrees, making bearing measurements noisy. Range estimates are calculated from time-of-flight (TOF) measurements that are registered when the received signal surpasses a fixed threshold. This signal gating produces foreshortening and lengthening effects as the angles of incidence to surfaces are varied. Spurious measurements also occur as a result of specular effects, multipath returns, confusion from side-lobes and cross-talk errors [112].

Lidar sensors: "light radar" sensors[2] have been used increasingly in SLAM research since the late 1990s [114, 115], displacing sonar as commercial off-the-shelf (COTS) prices have decreased. Like sonar, most lidar sensors acquire range measurements by measuring the time-of-flight, however they use light rather than sound resulting in faster and more accurate measurements. When packaged with a rotating mirror, a typical *scanning* lidar (sometimes called a scanning laser range finder) can make thousands or millions of planar range-bearing measurements per second. Beam divergence effects, combined with varying reflectivity of target objects, introduce many potential sources of measurement noise and bias. Lidar measurement models are of particular interest in this research and are discussed in the following section. Many researchers have servo-mounted scanning lidar sensors in either nodding [116, 117] or sweeping [118] configurations to produce full 3-D scans at 1 Hz or slower. The 3-D field of view comes at the cost of increased complexity (servo temporal synchronization) and decreased coverage in critical directions.

Multi-plane lidar: sensors integrate multiple lidar into a single scanning unit such that full 3-D scans can be acquired at high rates. The Velodyne HDL-64E combines 64 lidar

[2]The term "lidar" was introduced by astronomer James Ring in [113].

units and typically scans at 10 Hz, with a 26.8° vertical field of view (FOV). Multi-plane lidar were widely used in the 2007 DARPA Urban Challenge [119], and more recently in Google's Autonomous car [120]. In 2015, the least expensive 32-plane lidar that would be practical for mobile robotics costs USD $30,000. Perhaps due to this cost, I could not find any multi-robot SLAM research in the literature that uses multi-plane lidar.

Time-of-flight cameras: include a range of techniques that capture range measurements on a 2-D sensor array. Horizontal and vertical FOVs and resolutions vary widely. Flash lidar sensors employ an array of avalanche photo-diodes and counters, typically in a 128×128 array. Due to cost they are mostly used in space applications [121]. Photonic Mixer Devices (PMDs), such as the Swiss Ranger, use modulated infrared light to measure ranges over a 2-D pixel array, and have been used in a number of robotics experiments [122]. While less expensive, PMDs typically only work indoors due to their limited output power. Outdoor ambient light is generally too intense for TOF cameras to achieve sufficient signal-to-noise ratios with both useful ranges and FOVs.

Camera sensors: can also be used passively for mapping, where pixels from a *calibrated camera* and lens are considered as 2-D bearing-only measurements. By identifying and tracking objects between sets of camera images where the *baseline* (relative $\mathbb{SE}(3)$ transformation) is known, ranges can be estimated. Well-established stereo vision techniques use two cameras with a fixed baseline; image feature correspondences are identified between images using epipolar geometry, [111], and ranges are calculated where image disparities have been measured. Passive approaches suffer from many failure modes, from depth holes in image areas with no visual features, to self-similar scenes that create perceptual aliasing ambiguities [123]. Single cameras can also be used passively, by tracking 2-D image features across camera motion. However the baseline between images must be estimated from odometry which gives rise to the well-studied monocular SLAM problem [124, 125]. There is increasing research in high-level perception, where cues such as the relative size of an object can be used to estimate image scale, and hence depth [126].

Structured lighting: techniques overcome the common issues with passive vision by projecting coded patterns of light into the environment. Scenes are imaged with a calibrated camera and the coded light patterns allows correspondences (a local data association problem) to be solved more easily. Ranges are calculated using triangulation using techniques established in the 1980s [127]. Modern RGB-D cameras, such as Microsoft's Kinect, are inexpensive COTS products that use structured lighting techniques to output full color images with depth measurements, often referred to as RGB-D. In these sensors, an infrared (IR) laser and diffraction grating projects a speckled pattern into the scene. The pattern is subsequently imaged by a camera and IR filter and depth measurements triangulated [128]. Impressive mapping results have been demonstrated indoors, e.g. Kinectfusion [129],

however, similar to many TOF cameras their limited output power prevents them from being used in sunlight and outdoors.

Radar sensors: have seen limited, but increasing, use in SLAM research. Recent studies with millimeter-wave and ultra-wide band radar have shown promise, [130, 131, 132], however there are numerous sources of noise that must be handled robustly. Radar sensors measure range using either TOF or frequency modulation of RF signals, and typically make 360° range-bearing measurements in a single plane. Their signals can often penetrate through dust, smoke and rain since they operate with longer electromagnetic wavelengths than lidar. Radar sensors have a wider beam divergence than lidar, however, making data association harder, and like sonar they are highly sensitive to a target's surface properties and incidence angle. Specular reflections, from metallic objects in particular, can cause many spurious returns [130].

2.3.2. Lidar Measurement Noise

Lidar sensors are currently one of the most reliable options for robotic SLAM in both indoor and outdoor environments. The results presented in this thesis are based on several different COTS scanning lidars mounted on various mobile robots (see Appendix A for details). Scanning lidar sensors often exhibit similar sources and types of noise that can be loosely modeled as Gaussian [133].

The Hokuyo UTM-30LX is a typical COTS scanning lidar that uses an infrared laser. It acquires 1080 range measurements with 0.25° angular resolution at 40 Hz, with $\sigma = 30$ mm range error [133]. Similar to wave-guides in sonar sensors, lenses can only help to minimize, but not prevent, beam divergence and various other sources of noise. The UTM-30LX has a beam arc-width (along the scan direction) of 0.8°. At the maximum range of 30 meters this creates an elliptical laser spot size of 400 mm.

As each laser pulse diverges, it traces a volume that is approximately conical. If any part of this conical volume intersects an object, some of the reflected light may return through the lidar's lens. Once the lidar's receiver front-end has collected enough light (i.e. gating to a threshold) it registers the return and calculates the distance. This signal gating introduces range foreshortening and lengthening effects. Each of the noise sources described here occur in various datasets and affect the results presented in this thesis:

Angle of incidence: if a lidar pulse hits a surface perpendicularly, the entire laser wave front is reflected at the same time. As the angle of incidence increases, however, part of the wave front is reflected sooner and the received signal will trigger the threshold either sooner or later depending on the front-end detector design. As the angle of incidence approaches 90 degrees, a common configuration for horizontally mounted lidar, the lidar pulses travel

Figure 2.4. Lidar-based mapping in 3-D: In this work I mounted a servo-actuated "nodding" lidar on a quadrotor. With careful spatial and temporal calibration, lidar measurements are registered to images from a camera. The resulting 3-D lidar point clouds are colored like those generated by an RGB-D sensor, however the system can operate in direct sunlight. **Left:** a 3-D point cloud self portrait (perspective projection). Range noise can be seen in the lidar measurements that graze my arm and head. **Center:** lidar intensity measurements used for calibration (polar projection). **Right:** image from the wide FOV camera used to colorize the point cloud.

almost parallel to the ground. These "ground-grazing" returns are extremely sensitive to the robot's pitch, variations in the surface and other noise. In this configuration, even a non-moving robot can produce lidar measurements with meters of range noise. These ground-grazing returns can present a challenge for SLAM with a horizontally-mounted lidar.

Boundary effects: are closely related to grazing returns, spurious returns can occur at the boundary of objects, where the elliptical wave front may intersect part of one or more objects as it travels. The resulting range measurement is often averaged and a spurious measurement is created "hanging" in empty space between objects. Figure 2.4 demonstrates this effect.

Object surface properties: cause the amount of reflected laser light to varies greatly. A highly specular object, such as a mirror or still water, will reflect most of the laser light and will often return measurements to more distant objects (at the incorrect bearing). The more diffusely an object reflects light, the better it can be measured over a wider range of angles and distances. Objects that absorb infrared light (typically appearing black to humans) can often fail to reflect enough light, limiting measurement ranges. Further, in some lidar sensors white objects can appear slightly closer than black objects, since the receiver threshold is triggered slightly sooner [133].

Ambient light: Most commercial lidar sensors use infrared notch filters to increase the signal to noise ratio. While this allows them to function outdoors, strong ambient light, such as direct sunlight, will decrease their range. When the reflected light from a lidar pulse is not strong enough to trigger a measurement, a typical sensor model assumes there is free-space up to some fraction of the sensor's maximum range. This free-space assumption should vary depending on the ambient light levels, however without additional sensors, it is not possible to determine when missing lidar measurement are due to actual free space, or excessive ambient light.

Temporal synchronization: when mounted on a moving robot, a lidar's origin will move as the rotating sensor completes each scan. Moving at 1 ms^{-1}, for example, a 40 Hz lidar will produce up to 25 mm of skew if left uncompensated. Compensation can be performed using a continuous motion model, such as [134], however this requires rigorous synchronization and timestamping of sensor data. Figure 2.4 shows a colored 3-D point cloud generated from a lidar mounted on a fast moving servo-motor, with a global shutter camera and microsecond-level synchronization.

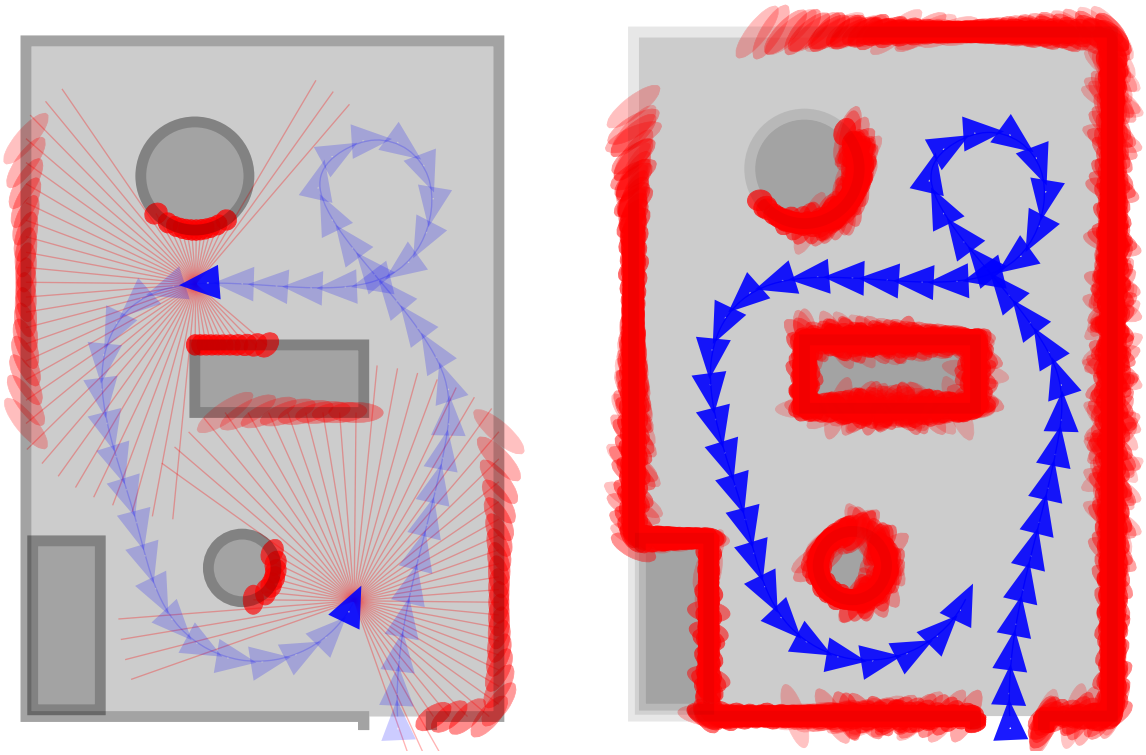
2.3.3. Lidar Sensor Model

Like all sensors, lidar range measurements are noisy. For each lidar measurement, only range, bearing and, in some models, intensity values are returned. Without knowledge of the surface properties of objects in the environment, or exact knowledge of a lidar sensor's movement through the environment, we lack the necessary information to estimate how much noise may be included in a return, or which of the error modes described in the last section may have occurred. In [135], Thrun et al. describe various models for sensor noise.

To demonstrate the challenges encountered in lidar-based mapping, I first describe a high-fidelity lidar model. This model is important, since it helps to explain the errors introduced when it is simplified later. The high-fidelity lidar model assumes a constant angular beam divergence that includes bearing uncertainty, and Gaussian noise in the range measurements. Combined, the range-bearing uncertainties for each lidar return are approximated with a 3-D uncertainty volume that has an ellipsoid shape. If we consider a horizontally mounted lidar, these uncertainty volumes can be projected into a horizontal plane, such that each return is modeled as a 2-D uncertainty ellipse.

Using this model, Figure 2.5 (a) shows two lidar scans with the 3-σ uncertainty ellipses for each measurement plotted. The width of the uncertainty ellipses increase linearly with range due to beam divergence. Large parts of the lidar measurements' 3-σ ellipses overlap free-space so that object boundaries are not well defined. In Figure 2.5 (b) many lidar scans are superimposed using Equation 2.12, while assuming no odometry noise. The structure of the environment can be discerned, however, the robot's noisy view is apparent. While it is possible to use this diverging sensor model to build maps by fusing uncertainty ellipses [136], it is computationally expensive.

In practical mapping systems we often treat the lidar scanner as an *ideal sensor*. The ideal sensor model defines two geometric features for each valid lidar return: 1) a narrow and non-diverging ray of free-space that spans from the sensor's origin to the object it hit, and 2) a 3-D point object at the range indicated by the return, which may belong to the surface of a larger object. While this model is computationally less expensive, it introduces systematic errors that the resulting maps do not describe probabilistically. Ruhnke et al. demonstrate the differences between the ideal model and a high-fidelity model in [136].



(a) Lidar scans from $t = 30$ and $t = 48$. Laser pulses diverge, increasing the bearing uncertainty to more distant objects.

(b) Superimposing all lidar measurements demonstrates the noisy view the robot sees of its environment.

Figure 2.5. Mapping with a diverging-beam model: For the robot and lidar sensor in the example problem, this 2-D top view shows exact robot poses (no odometry noise) as blue triangles, while lidar measurements and 3-σ uncertainty ellipses are projected in red.

Measurements within a single scan from a scanning lidar are typically modeled as independent, given the robot's position, i.e:

$$p\left(\mathbf{z} \mid \mathbf{r}, \mathbf{m}\right) = \prod_{i=1}^{M} p\left(\mathbf{z}_i \mid \mathbf{r}, \mathbf{m}\right) \tag{2.14}$$

This model is optimistic, since jitter in the scanner's angular encoder could produce correlated errors, however in practice these errors are very small [135].

2.3.4. Map Parameterization and Storage

The primary motivations for SLAM and map building are to enable robots to navigate through the environment and execute tasks, while optionally providing situational awareness to a human operator. The choice of map parameterization is driven by these motivations, along with other factors such as the types of sensors available. In this section I use lidar scanners to introduce various map parameterizations, however note that these parameterizations can be used with many of the sensor types described in the previous

sections. Both 2-D and 3-D parameterizations are considered here, with a brief comparison given in Section 2.3.5.

To perform on-line SLAM with sensors that acquire millions of measurements per minute, map data must be parameterized and stored in a *spatial database* that allows efficient:

1. **Storage:** transforming and merging new sensor measurements into the map.
2. **Retrieval:** extracting spatial data from the map, e.g. for robot navigation.
3. **Matching:** comparing spatial data to perform data association.
4. **Updates:** global/structural map modifications after loop closure events.

The efficiency of these operations are considered for the various parameterization and storage techniques in the following sections.

Data association is one of the most critical parts of any SLAM implementation, and the efficiency when matching and aligning either pairs of lidar scans, lidar scans with maps, or pairs of maps, varies widely between parameterizations and storage techniques. The cost when applying global/structural map corrections during loop closures can vary widely, since some techniques require the entire database to be rebuilt.

2.3.4.1. Points With Scan Matching

The simplest map parameterization is collect all of the lidar points and store them in a spatially-efficient data structure. One such structure is the *quadtree*, which provides $\mathcal{O}(\log n)$ storage and retrieval time complexity [137]. While quadtrees are efficient for storage and retrieval, they can be slow when used for matching and during structural updates.

Scan matching describes a range of algorithms that match and align batches of lidar points. A frequently used scan matching algorithm is Iterative Closest Point (ICP) [138, 115]. ICP forms a non-convex optimization problem that frequently becomes stuck in local minima, thus many variants have been described in the literature. Iterative Closest Line (ICL) [139] matches points to high-level features, often avoiding creating local minima, however this regularization introduces errors (ICL is not well-suited to curved surfaces) while adding computational complexity. Both ICP and ICL involve highly-repetitive searches for point correspondences that are often redundant. In multi-robot SLAM, overlapping lidar points can quickly become dense, and the $\mathcal{O}(\log n)$ retrieval time will eventually prevent on-line operation. Lidar point maps can be relatively large to store, which makes them expensive to transmit over a wireless network, and inefficient if redundant points are not removed.

2.3.4.2. Landmark-based Mapping

Many environments can be simplified into sets of high-level *landmarks* that are sometimes referred to as *features*. Landmark-based maps typically reduce the volume of sensor data, while making data associations less computationally expensive. The popular Victoria Park dataset, used widely in landmark-based SLAM research [140, 141, 142, 143], contains data from a horizontal lidar scanner that is driven around hundreds of trees. The first processing stage identifies groups of lidar points that form curves (tree trunks). These tree observations are parameterized by the trunk center, $\mathbf{z}_{t,i} = [x, y]^T$, and radius. Observations are either matched to existing landmarks, or inserted into the map as new landmarks. This data association problem is well defined, and efficient techniques have been developed to handle matching in the presence of ambiguities [144].

High-level landmark parameterizations are chosen to match the shape of objects in the environment, and while compelling results have been demonstrated on the Victoria Park data-set, it is not an everyday environment. Urban environments frequently have flat surfaces that are better parameterized as line segments. As such, line features have been studied extensively, however they introduce systematic errors when describing environments with curved surfaces [145]. While the large dimensionality reduction makes landmark-based maps very efficient for storage, retrieval, matching and structural updates, they make the first stage in the SLAM pipeline entirely responsible for accurately detecting, segmenting and matching landmarks. Furthermore, this data reduction typically prevents segmentation and data association decisions from being reconsidered at a later time.

2.3.4.3. Occupancy Gridmaps

Gridmaps are used extensively throughout this research work to describe 2-D environments with arbitrary geometry. They were first described in the 1980s by Moravec and Elfes, in the context of sonar-based mapping [146, 147]. As the name suggests, the environment is divided into a grid, with equal-sized cells. Gridmap storage and computation costs increase with the inverse square of the cell size, and the size must be carefully selected to suit both the robot and environment geometry. The robot in the example problem (Section 2.1.4) drives through a 90 cm doorway, while it is 50 cm in diameter. A 10 cm cell size leaves enough margin to enables navigation through the doors, making this room about 80 × 100 cells. Gridmaps for this problem are shown in Figure 2.6 and Figure 2.7.

The map, $\mathbf{m}$, is partitioned into $w \times h$ cells, m_i, such that:

$$\mathbf{m} = \sum_{i}^{w \times h} \mathrm{m}_i \tag{2.15}$$

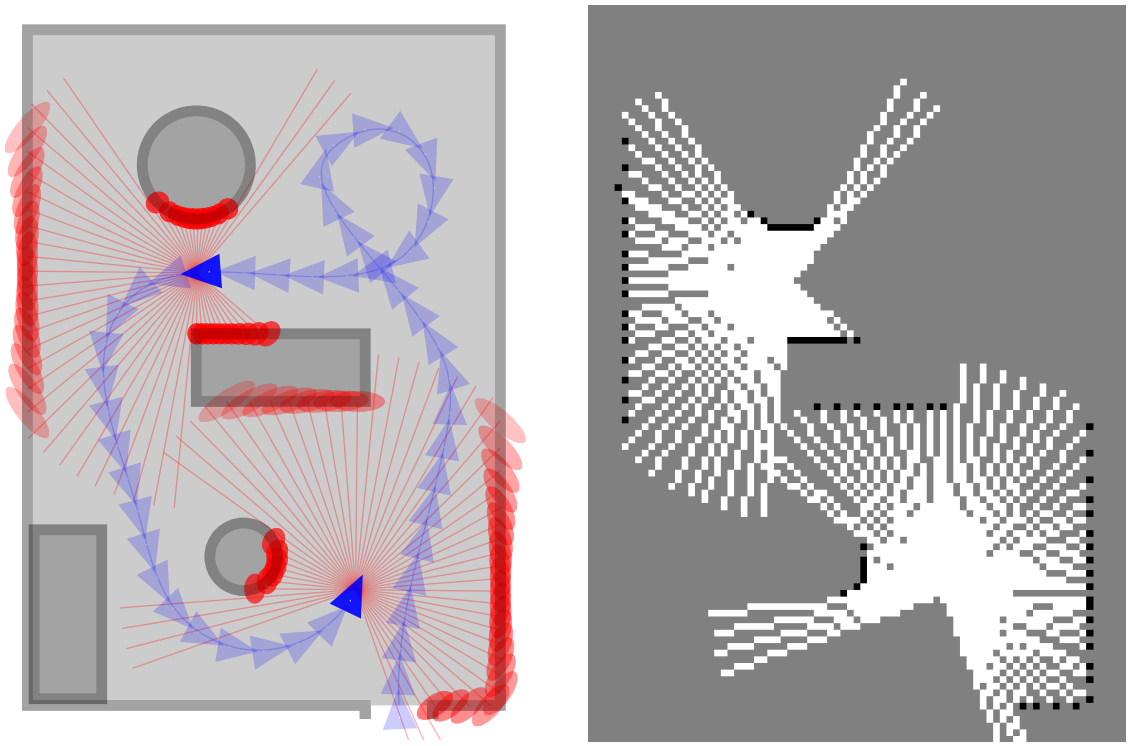


Figure 2.6. Occupancy gridmap: Two lidar scans from the example problem, at $t = 30$ and $t = 48$, projected from exact robot poses (no odometry noise). **Left:** 2-D top view. **Right:** Occupancy gridmaps with two lidar scans fused. For each grid cell m_i: **Gray:** Unknown $p(\mathrm{m}_i) = 0.5$. **Black:** Occupied $p(\mathrm{m}_i) = 1$. **White:** Free $p(\mathrm{m}_i) = 0$.

Each grid cell, m_i, encodes the current belief, or probability, that an object exists at that particular location in the environment. Following convention, [135], all cells in a map are initialized to an unknown state where $p(\mathrm{m}_i) = 0.5$. As sensor data is added to the map, occupied cells tend towards $p(\mathrm{m}_i) = 1$, while cells that are free-space tend towards $p(\mathrm{m}_i) = 0$.

The ideal gridmap would maintain the full posterior distribution over all cells, given the uncertain robot path and noisy measurements, as represented in the probability distribution in Equation 2.13. Even with a modest 80×100 cell gridmap, however, the posterior explodes in complexity. In practice we approximate the posterior distribution by treating the robots path, $\mathbf{r}$, as exact and each cell, m_i, as an independent variable:

$$p(\mathrm{m}_i \mid \mathbf{r}, \mathbf{z}) \tag{2.16}$$

Thus, the state of each cell is given only by the measurements that cross its path. To ease computation we typically iterate over measurements and *ray trace* the path each one took, updating grid cells according to the sensor model. Normally, only range-bearing sensors that have well-defined sensor models are used with gridmaps. While sonar sensors have wide beam divergence and require complex models [147], the ideal model suffices for

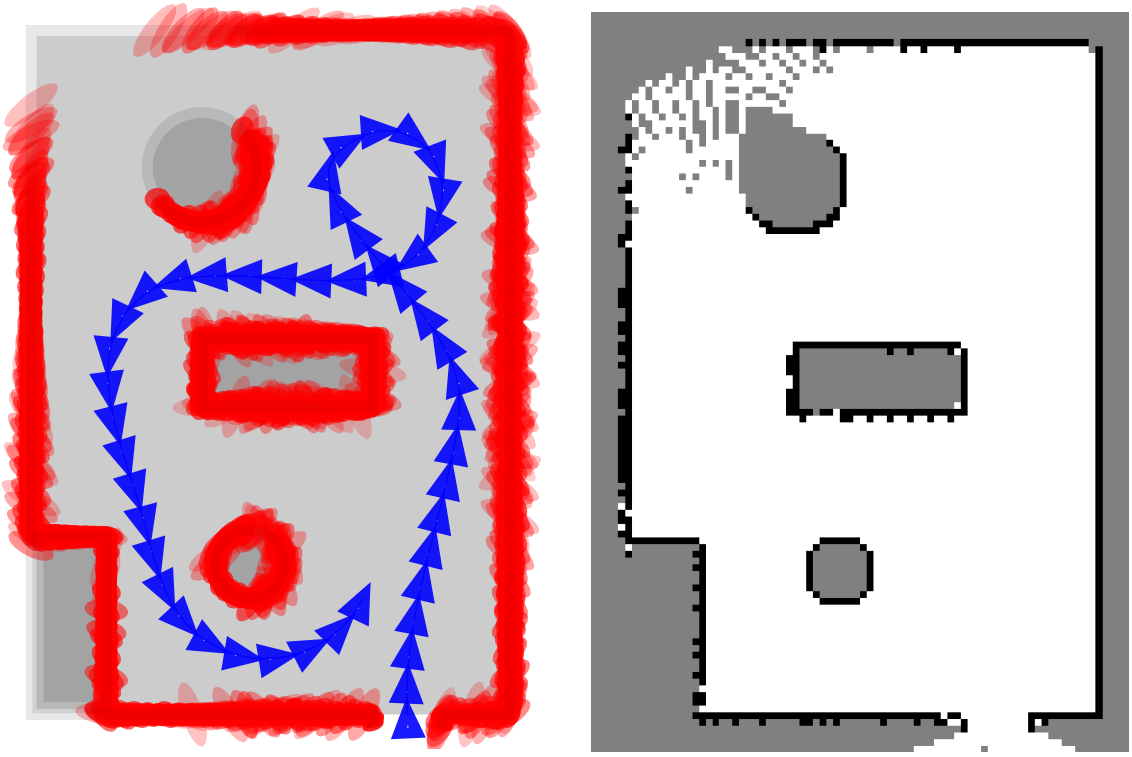


Figure 2.7. Occupancy gridmap: All lidar scans from the example problem, projected from exact poses. The door is visible at the bottom right, while an unmapped area remains at the top. **Left:** 2-D top view. **Right:** Occupancy gridmaps. For each grid cell m_i: **Gray:** Unknown $p(\mathrm{m}_i) = 0.5$. **Black:** Occupied $p(\mathrm{m}_i) = 1$. **White:** Free $p(\mathrm{m}_i) = 0$.

most lidar scanners (Section 2.3.3). Ray tracing lidar measurements involves updating the cells from the sensor's origin along the ray to each range-bearing measurement. Both of these 2-D coordinates are likely to be irrational (floating point) numbers and they are first rounded to the nearest grid cell. Treating the gridmap as an image, the ray is drawn using a line rasterization algorithm, such as Bresenham's [148].

By treating neighboring cells as independent, each cell can be considered a binary Bayes filter [135]. To simplify cell updates, and avoid numerical issues when $p(\mathrm{m}_i)$ is 0 or 1, a log odds ratio ($\log \frac{p(\mathrm{m}_i)}{1-p(\mathrm{m}_i)}$) is used to accumulate the likelihood that each cell is occupied. Cells are updated using addition and subtraction, which further encourages the treatment of gridmaps as images.

Figure 2.6 shows an empty gridmap that has been updated with two lidar scans from the example problem. The gray unknown cells have been updated to indicate the known free-space in white, and occupied cells in black. Large areas close to these scans remain unknown. In Figure 2.7 these areas are filled when all of the lidar scans are added. In the lower right an open door is clearly visible, and an area with bad coverage remains in the top left.

It is important to note that gridmaps are an approximation, since spatial data is rounded to the nearest cell. For the ideal lidar sensor model, half a cell-width of noise will be add to each return. In practice this rounding has little noticeable effect, since the noise in a typical lidar sensor and the rounding on a 10 cm gridmap are of similar magnitude.

The example in Figure 2.7 compresses 2,700 lidar measurements into a gridmap with 8,800 cells (23 % reduction in memory use). However, as additional lidar scans are added to the gridmap, memory usage does not increase unless the map boundary expands. Measurements are added to the gridmap in $\mathcal{O}(n)$ linear time, and cell states retrieved in $\mathcal{O}(n)$ time also. At the cost of adding a small amount of noise, gridmaps are very efficient at storage, retrieval and matching.

In Equation 2.16 the uncertainty in the robot pose, $\mathbf{r}$, was ignored in the posterior distribution $p(\mathrm{m}_i \mid \mathbf{r}, \mathbf{z})$. In practice this uncertainty increases as a robot explores, suggesting that gridmaps should appear increasingly "blurry" along the robot's path. I could not find any examples in the literature where this blurring is performed on gridmaps, perhaps, historically, because it would have been computationally expensive (this is revisited as potential future work in Chapter 8). Instead, gridmaps typically show a crisp representation of the environment that becomes increasing misaligned along the robot's path (see Figure 2.8 on page 46). Loop closures can correct these misalignments, however at relatively high computational cost.

Gridmaps can become computationally expensive during loop closures and other batch updates, since large structural changes can require the gridmap to be rebuilt from scratch. While ray tracing a single measurement is inexpensive, the cost of large structural updates can become considerable when maps grow large. This cost is compounded when multi-robot SLAM uses gridmaps, since loops are closed much more frequently (Section 2.1.3).

2.3.5. 2-D vs. 3-D Maps for Wheeled Robots

My research work focuses on 2-D map parameterizations, however it is interesting to briefly consider the cost and utility of 3-D maps. Several parameterizations exist that store map data in compact 3-D structures. Similar to the quadtrees described in Section 2.3.4.1, the *octree* data structure [149] can be very efficient when storing and retrieving 3-D data. Octrees, however, are also relatively slow when used for matching and structural updates, and would quickly become a bottleneck in a 3-D multi-robot SLAM implementation.

Trajectory planning in 3-D volumes is much more computationally demanding than in 2-D maps. In the case of mobile robots that move in constant contact with the ground, this additional computation is difficult to justify. The pose is typically restricted to a 2-D manifold in 3-D space, and although pitching and rolling is possible (i.e. full $\mathbb{SE}(3)$), the pose in most environments is parameterized adequately by $\mathbb{SE}(2)$.

The volume of space a trajectory planner must consider for robot collisions is restricted to positive and negative height obstacles, e.g. walls and cliffs, respectively. It is advantageous to project these obstacles into 2-D navigation cost maps and perform all trajectory planning in the dimensionally reduced space. The gridmap fusion technique described in Section 5.3 can augment 2-D gridmaps with navigation cost maps, which enables considerable efficiencies.

2.4. Motion Models and Localization

Estimating a robot's pose within a fixed reference frame forms the *localization* part of the SLAM problem. The complexity of the localization solution depends on the desired accuracy and external factors in the robot's environment; a GPS and compass may be good enough for some robots, however a robust system should not be stymied by inaccurate, intermittent or jammed sensor data.

This section introduces robot motion models, and three approaches for the robotic localization. The first two approaches use sensors designed for pose estimation: on-board odometry that measures relative pose changes, and external sensors that provide globally-referenced pose estimates. The third approach uses mapping sensors, such as lidar, along with maps to estimate pose. In this section I continue to use a planar-world assumption with mobile robots moving in $\mathbb{R}^2$ with $\mathbb{SE}(2)$ transforms.

2.4.1. Mobile Robot Motion Models

If it was possible to know the exact pose of a robot at an instant in time; subsequent motion and noisy measurements would always introduce uncertainty into future poses. It is important to have a *motion model* that predicts how a robot's pose will change over time, and, importantly, how the pose uncertainty will grow. Typical motion models use either wheel rotation measurements, commanded motion or assumptions about vehicle dynamics. They describe the distribution:

$$p\left(\mathbf{r}_t \mid \mathbf{r}_{t-1}, \mathbf{u}_t\right) \tag{2.17}$$

For wheeled robots, the motion model is dominated by the mechanical design of the drive mechanism. Refer to Bräunl [47], or Dudek and Jenkin [150] for a review of different drive types. Motion models for each are given in [151].

2.4.1.1. Differential Drive Motion Model

The mobile robots used in my research work all have *differential* drives [47, 150]. Differentially driven robots are actuated by two drive motors and are *non-holonomic,* that is, they

are under-actuated compared to their three degree-of-freedom configuration space. Differential robots, such as the robotic vacuum cleaners described in Section 1.1.2, can follow arbitrary curves and turn in place, making them well-suited to cluttered environments.

Following the example in Section 2.1.4, we assume the robot stays in contact with the ground (a 2-D manifold in 3-D space) and parameterize its $\mathbb{SE}(2)$ pose with $\mathbf{r} = [x_r, y_r, \phi_r]^T \in \mathfrak{se}(2)$. The motion model estimates incremental changes to this pose, or:

$$\hat{\mathbf{u}}_t^{t-1} = \ominus \mathbf{r}_{t-1} \oplus \mathbf{r}_t = \left[\hat{x}_u, \hat{y}_u, \hat{\phi}_u\right]^T \tag{2.18}$$

By convention, the hat on $\hat{\mathbf{u}}$ indicates an estimate. Motion models using wheel rotation measurements and dead reckoning are briefly described here:

Wheel measurements: most commercial robots measure the rotation of each wheel (parameterized θ_L and θ_R). Rotational velocities can be estimated with $\omega_L = \frac{\Delta\theta_L}{\Delta t}$ and $\omega_R = \frac{\Delta\theta_R}{\Delta t}$, where the accuracy depends on the encoder resolution. Estimates are often generated by embedded processors, with filtering at rates up to 100 Hz or more. Across a small time period Δt, and small distance, it is reasonable to assume that ω_L and ω_R remain constant. A differential drive robot's ego-motion is given by:

$$\hat{\mathbf{u}}_{t+1}^t = \begin{bmatrix} \rho \sin \hat{\phi}_u \\ \rho - \rho \cos \hat{\phi}_u \\ \hat{\phi}_u \end{bmatrix} \quad \in \mathfrak{se}(2) \tag{2.19}$$

Where the change to the robot's heading, $\hat{\phi}_u$, for drive wheels having equal radii, R, and wheel separation L, is:

$$\hat{\phi}_u = \frac{R}{L}(\omega_R - \omega_L)\,\Delta t \tag{2.20}$$

While the robot drives around a smooth arc with radius:

$$\rho = \frac{L(\omega_R + \omega_L)}{2(\omega_R - \omega_L)} \quad \forall \quad \omega_R \neq \omega_L \tag{2.21}$$

In the trivial case where the robot is driving straight, the incremental ego-motion estimate is $\hat{\mathbf{u}}_{t+1}^t = [R\omega\Delta t, 0, 0]^T$ and $\omega = \omega_R = \omega_L$. A differentially driven robot is assumed to not slide sideways, thus as Δt is made smaller the component $\hat{y}_u$ tends towards zero.

Dead reckoning: predicts the future state of the robot with a dynamic model. A typical model assumes constant linear and angular velocities, thus for equal time intervals:

$$\hat{\mathbf{u}}_{t+1}^t = \hat{\mathbf{u}}_t^{t-1} \tag{2.22}$$

In this model, accelerations are generally treated as zero-mean Gaussian noise [124]. This assumption is well-formed for differential drive robots, since the velocity components $\dot{\mathbf{r}} = \left[\dot{x_r}, \dot{y_r}, \dot{\phi_r}\right]^T \in \mathfrak{se}(2)$ correspond directly to constant angular wheel velocities. The zero-mean noise assumption is challenged during sustained constant accelerations, such as when control torques are applied by the motors. Note that motion models like these are frequently inaccurate.

2.4.2. Ego-Motion Estimation with Odometry Sensors

Odometry sensors are incorporated in mobile robots to help estimate a robot's relative motion. They typically augment the motion model to estimate changes to a robot's pose, i.e. the ego-motion $\mathbf{u}_{t+1}^t = \ominus \mathbf{r}_t \oplus \mathbf{r}_{t+1}$. Relative motion sensors are distinct from global (absolute) sensors, such as GPS, which are discussed in Section 2.4.3. Every sensor produces measurements that are corrupted by noise, this section describes a few common odometry sensors and their noise models.

2.4.2.1. Odometry Sensors

Wheel encoders: measure relative wheel rotations, $\Delta\theta_L$ and $\Delta\theta_R$. Encoders are typically fixed to either the motor or gearbox output, with design choices often trading angular resolution with maximum speed. Rotational velocities are estimated with $\omega = \frac{\Delta\theta}{\Delta t}$, however resolution and sampling effects often produce noisy measurements that require digital filtering. Using Equation 2.19 wheel encoders can only estimate planar motion in $\mathbb{SE}(2)$. Wheel-based odometry assumes there is no slippage between wheels and the terrain. In practice, mobile robots frequently overcome static and rolling friction, and when one or more rolling wheels slip, or "skid", large odometry errors can occur. Terrain slope, friction and other properties can vary, even between individual wheels. Differential drive robots with four wheels often experience large odometry errors when turning, since wheel-slip is required to actually turn [150].

Rate gyroscopes: measure rotational velocities, or rates. They are a key component of Inertial Measurement Units (IMUs), where three orthogonal rate "gyros" measure rotation in 3-D, $\mathbf{\Omega} = [\Omega_{roll}, \Omega_{pitch}, \Omega_{yaw}]^T$. Integrating the rate measurements produces rotation estimates in $\mathbb{SO}(3)$ [152], where typical fiber optic gyros drift about 0.1° per minute [153], and less-expensive micro-electro-mechanical system (MEMS) gyros drift about 5° per minute. MEMS rate gyros are sensitive to temperature and drift over time, requiring bias parameters to be estimated on-line.

Accelerometers: measure linear accelerations. They are also key components in IMUs, typically arranged in orthogonal sets measuring $\mathbf{a} = [a_x, a_y, a_z]^T$. Typical MEMS accelerometers are both noisy and have biases that drift over time. Translational motion

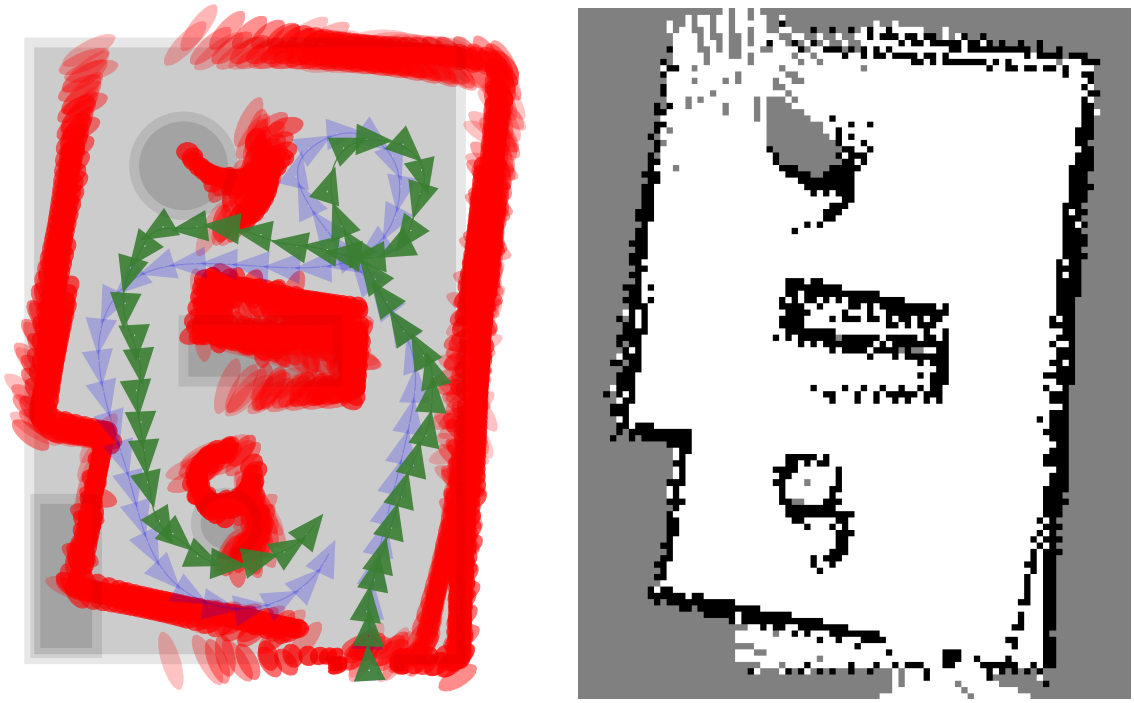


Figure 2.8. Occupancy gridmap: All lidar scans from the example problem projected from odometry estimates accumulated with noise. **Left:** 2-D top view showing lidar measurements superimposed. Blue: the robot's true path. Green: the robot's noisy path based on odometry. **Right:** In the corresponding gridmap the structure of the environment can be discerned, however large misalignments are visible.

can be estimated through double integration, however meaningful estimates are difficult with accelerometers alone. Accelerometers also provide globally-referenced pitch and roll information, described in Section 2.4.3.

Visual odometry: (VO) solutions estimate motion in $\mathbb{SE}(3)$ using one or more cameras. Visual optical-flow, and/or 2-D feature tracking produces estimates for ego-motion. VO forgets the environment as it passes out of view, differentiating it from visual SLAM, e.g. [124]. By restricting motion estimation to $\mathbb{SE}(2)$, efficient VO solutions exist for single cameras [125]. VO is computationally expensive, while degenerate scenes and camera motions can cause it to fail or give spurious translation estimates [154]. It is also affected by environmental issues such as dust, insufficient light, or sunlight blinding the camera. I have included VO in this list because it works like a passive odometry sensor and is complementary to the sensing modalities above.

2.4.2.2. Odometry Noise Models

The actual path driven by a robot will never match what was commanded due to errors such as wheel slippage. Motor controllers typically use the same odometry sensors for position feedback, and thus the estimate $\hat{\mathbf{u}} = \left[\hat{x}_u, \hat{y}_u, \hat{\phi}_u\right]^T$ is the ego-motion the robot *thinks* it underwent in the real world.

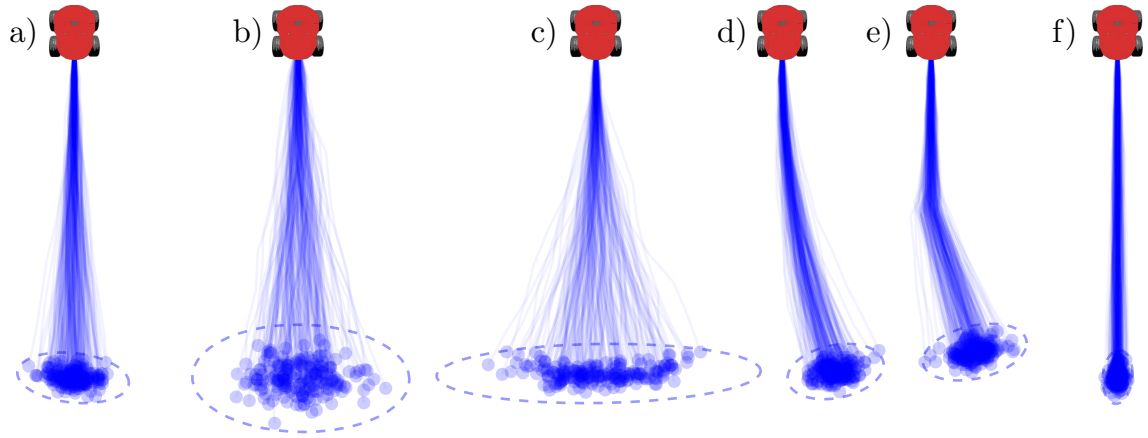


Figure 2.9. Accumulated odometry noise: Robot paths for six simulated drives corrupted with various types and amounts of odometry noise. For each drive, a) through f), 200 paths have been simulated using Monte Carlo sampling, with 3-σ covariances shown as dashed ellipses. In each case, the odometry estimate $\hat{\mathbf{u}} = [\hat{x}_u, \hat{y}_u, \hat{\phi}_u]^T = [4, 0, 0]^T$ describes the motion we *think* the robot underwent, here four meters directly forwards. Drive a) simulates typical paths that are well-modeled by Gaussian noise $\mathbf{\Sigma}_u = \text{diag}(\sigma_x^2, \sigma_y^2, \sigma_\phi^2)$, b) simulates twice the Gaussian noise, as if the surface type was slipperier. Drive c) simulates $3 \times \sigma_\phi$ higher heading noise, d) simulates a bias in the heading, caused by a flat left tire. Drive e) simulates a large yaw mid-drive, as if the left wheel had slipped on some leaves, while, f) simulates robust yaw control using wheel encoders fused with gyroscope data.

The error between this odometry estimate and the actual ego-motion, $\tilde{\mathbf{u}} = \hat{\mathbf{u}} - \mathbf{u}$, is often modeled as a zero-mean Gaussian process, i.e. $\tilde{\mathbf{u}} \simeq \mathcal{N}(0, \Sigma_{\mathbf{u}})$. The odometry estimate $\hat{\mathbf{u}} = \mathbf{u} + \tilde{\mathbf{u}}$ can be written in the standard form:

$$\hat{\mathbf{u}} = \mathcal{N}(\mathbf{u}|\hat{\mathbf{u}}, \mathbf{\Sigma}_u) \tag{2.23}$$

$$= \frac{1}{(2\pi)^{3/2}|\mathbf{\Sigma}_u|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{u} - \hat{\mathbf{u}})^T \mathbf{\Sigma}_u^{-1}(\mathbf{u} - \hat{\mathbf{u}})\right) \tag{2.24}$$

This is typically estimated from raw sensor measurements using Equation 2.19. The multivariate normal distribution is generally assumed to be uncorrelated, with covariance matrix $\mathbf{\Sigma}_u = \text{diag}\left(\sigma_x^2, \sigma_y^2, \sigma_\phi^2\right)$. Other models exist to characterize odometry noise, such as triangular distributions or sample-based distributions [135]. The normal distribution is used in the majority of SLAM algorithms because it is relatively simple to manipulate.

While the odometry noise covariance may be small, a large number of measurements are often integrated over a robot's entire path, where considerable errors can accumulate. For a robot starting at pose $\mathbf{r}_0$, the pose estimate $\hat{\mathbf{r}}_\tau$ at time τ is:

$$\hat{\mathbf{r}}_\tau = \mathbf{r}_0 \oplus \sum_{t=0}^{\tau-1} \hat{\mathbf{u}}_{t+1}^t \tag{2.25}$$

Here the summation Σ applies the composition operator, $\oplus$, repeatedly. Accumulating pose error is demonstrated using the example problem in Figure 2.8. After driving in a loop around the room, the robot's gridmaps show considerable misalignment.

Figure 2.9 uses Monte Carlo sampling to further demonstrate odometry errors. Here a simulated robot is commanded to drive four meters. Six drives, a) through f), are corrupted with various types and amounts of odometry noise. Each drive is sampled 200 times, and the range of accumulated errors are very apparent in the spread of the actual poses after driving. In each case the odometry estimate indicates the robot has driven exactly four meters forward. In a) the paths are sampled from the normal distribution, $\mathcal{N}(\mathbf{u}|\hat{\mathbf{u}}, \mathbf{\Sigma}_u)$, while b) experiences twice this noise ($4 \times \mathbf{\Sigma}_u$). In both cases the distribution of final poses fits a normal distribution well, with the points lying inside their 3-σ covariance ellipse. This suggests the Gaussian noise model is adequate in some circumstances.

2.4.2.3. Odometry Linearization Errors

When odometry measurements are repeatedly composed, as in Equation 2.25, their covariances are composed also. The accumulated estimate, $\hat{\mathbf{u}}$, in the normal distribution $\mathcal{N}(\mathbf{u}|\hat{\mathbf{u}}, \mathbf{\Sigma}_u)$, can be calculated in closed form using the composition operator '$\oplus$'. The covariances, however, require a linearization step when composed:

$$\mathbf{\Sigma}_{\mathbf{u}_1 \oplus \mathbf{u}_2} = J_{\mathbf{u}_1}^T \mathbf{\Sigma}_{\mathbf{u}_1} J_{\mathbf{u}_1} + J_{\mathbf{u}_2}^T \mathbf{\Sigma}_{\mathbf{u}_2} J_{\mathbf{u}_2} \tag{2.26}$$

Where the Jacobians $J_{\mathbf{u}_1} = \frac{\delta(\mathbf{u}_1 \oplus \mathbf{u_2})}{\delta \mathbf{u_1}}$ and $J_{\mathbf{u}_2} = \frac{\delta(\mathbf{u_1} \oplus \mathbf{u}_2)}{\delta \mathbf{u}_2}$, are defined in [110].

As covariances are composed using Equation 2.26, each linearization step introduces a small error that is accumulated along the robot's path. The linearization shrinks the covariances slightly, artificially reducing the estimated uncertainty. The effects of this are well understood [155], and have been shown to prevent classic SLAM solutions from scaling to large areas [156, 157]. As the heading noise, σ_ϕ, decreases, the linearization errors decrease also; this motivates the search for more robust odometry (Section 2.4.2.5).

Figure 2.10 extends each of the sampled paths in Figure 2.9 by simulating a further four meter drive. The exaggerated heading noise in c) now reveals a distinct "banana" shaped distribution in the final poses. A similar non-Gaussian distribution is visible for b) also. In all cases, as the paths grow, the normal distribution will always become over-optimistic at representing pose uncertainty.

For these simulated drives, the Gaussian distribution, $\mathcal{N}(\mathbf{u}|\hat{\mathbf{u}}, \mathbf{\Sigma}_u)$, was sampled to generate the random distribution of paths. To show the linearization error, the expected robot

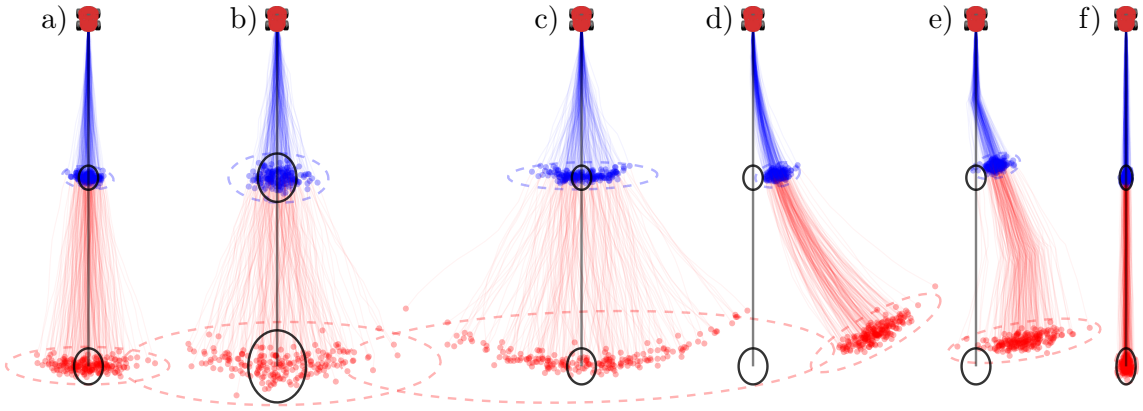


Figure 2.10. Accumulated linearization errors: Linearization errors will accumulate for any robot that composes noisy odometry estimates in $\mathbb{SE}(2)$. The six simulated drives from Figure 2.9 have been extended by four meters. **Red ellipses:** the 3-σ covariance ellipses can no longer describe the emerging "banana" shaped distributions of actual robot poses. **Black ellipses:** after 20 odometry compositions the accumulated pose uncertainties are greatly underestimated due to linearization errors.

path and its uncertainty can be calculated by composing odometry, $\hat{\mathbf{u}}$, using Equation 2.25, and the covariances, $\mathbf{\Sigma_u}$, using Equation 2.26. The expected path and 3-σ pose uncertainties are plotted in Figure 2.10 in black. Based on odometry, the uncertainty is dramatically underestimated in a), b) and c).

Linearization errors can be minimized by reducing heading uncertainty [110], with either a global pose sensor (Section 2.4.3) or by re-localizing in an existing map (Section 2.4.4). While SLAM algorithms (Section 2.5) can constrain heading uncertainty (and thus linearization errors), heading uncertainty in SLAM will *always* accumulate as a robot drives further from its origin, and thus linearization errors will always be introduced. In section Section 4.3.6 I show how submapping approaches can effectively isolate local linearization errors from the global SLAM solution.

2.4.2.4. Odometry Failure Modes

Common odometry issues can be caused by failures in the robot hardware, incorrect calibration and irregularities in the environment. Drives d) and e) in Figure 2.9 and Figure 2.10, show examples where the zero-mean Gaussian noise model fails to match reality. In each case the odometry suggests the robot has driven straight forwards. Drive d) shows a positive bias in $\hat{\phi}_u$ that could be caused by a flat tire or damaged encoder. Simple calibration issues, such as an incorrectly measured distance between wheel centers, or a misalignment can add large biases also.

Uneven and irregular surfaces, or interference from debris such as pebbles or leaves, can cause differential slippage between wheels. In Figure 2.10 e), for example, the robot yaws suddenly mid-drive, as if the wheels had slipped on some leaves. Large odometry errors

like these are not well-modeled by Gaussian noise, and are difficult to model in general. In Section 4.3.6 I describe a method for handling sudden spikes in pose uncertainty.

2.4.2.5. Robust Odometry

Robust odometry solutions fuse data from multiple sensors to reduce errors and correctly handle failure modes. For example, an unexpected change in heading from a single wheel slipping or an under-inflated tire could be detected by the yaw axis in a gyroscope. With an appropriate failure detector and sensor fusion, for example [158, 159], the first five drives in Figure 2.10 could reject the heading drift and appear similar to drive f).

Odometry sensors with complementary strengths and weaknesses can be paired to improve robustness. With a single camera observing a distant scene, for example, it is difficult to distinguish between rotations and translations, whereas an IMU can directly observe rotations and disambiguate. For example, in [160] Li and Mourikis describe a robust algorithm that uses camera images and IMU data to produce ego-motion estimates, however while their algorithm's accuracy is impressive, odometry errors still accumulate over time.

2.4.3. Global Pose Estimation with External Localization

Externally referenced sensors produce pose estimates with respect to the environment. A GPS receiver is a common example, producing globally-referenced latitude, longitude and altitude estimates. Often the environment is augmented with infrastructure, such as GPS satellites, RF beacons or visual markers that have been pre-localized within a chosen global frame. Alternatively, measurements can be made to existing landmarks and phenomena that have previously been measured and mapped, such as the Earth's magnetic field.

External localization techniques produce pose estimates with respect to the chosen global frame. In some cases they only observe partial pose information, such as heading from a magnetometer, or pitch and roll from an accelerometer. Measurements from multiple sensors are generally fused to produce complete pose estimates, including uncertainties. These techniques are often complex, and similar to the odometry sensors described in Section 2.4.2, they produce noisy measurements and have a variety of failure modes. A few common external localization sensors are described in this section, along with their noise and failure modes.

2.4.3.1. Externally Referenced Localization Sensors

GPS: receivers have become both inexpensive and commonplace for terrestrial navigation. They estimate globally-referenced latitude, longitude and altitudes (3 DOF pose) with

varying accuracies depending on the arrangement of the satellites. Their numerical solvers produce covariance estimates, often expressed as a dilution of precision (DOP) [161], that indicate a Gaussian error model. GPS errors are frequently non-Gaussian, however, and multiple sources of error exist. Variations in the ionosphere cause biases of 5 to 15 meters that change slowly over the course of each day. Satellite orbit (ephemeris) errors and clock drifts induce additional biases of 4 or more meters [162]. The pose is estimated from RF range measurements to 4 or more satellites. In urban environments, intermittent primary signals, and secondary reflections (multipath errors, see Appendix B) can cause large transient errors in the pose estimate. These biases and large step errors make robotic navigation with GPS problematic, especially with large teams of robots operating in urban environments and over many hours. The RF signals from GPS satellite are very low power, and in a military context they are susceptible to both hostile jamming and spoofing [78, 79].

Differential GPS (DGPS): solutions use signals broadcast from fixed base stations to correct for the ionosphere, ephemeris and clock drift errors. DGPS requires a separate communications channel to correct for errors in real-time, introducing an additional point of failure. Correction signals can reduce errors to below one meter, however they are only valid in the vicinity of the base station [162].

Real-time Kinematic GPS (RTK-GPS): solutions also rely on base station infrastructure and a second communications channel. Rather than sending pose correction signals, they send carrier-phase, or per satellite pseudo-range data that RTK-GPS receivers integrate to produce pose estimates accurate to 1-2 centimeters [162]. There are some interesting possibilities described in the literature where carrier-phase data is incorporated into the SLAM algorithm to estimate relative motion with similar accuracies, but without the RTK-GPS base station [163, 164].

Magnetometers: measure external magnetic fields. Three magnetometers are often used in an orthogonal configuration to estimate orientation with respect to the Earth's geomagnetic field vector, a configuration that is often integrated into MEMS IMUs. The geomagnetic field is relatively weak, however, and magnetometers are highly susceptible to stray magnetic fields, particularly in urban environments. Ferrous materials can become magnetized over time and hidden metal objects, such as reinforcing bars in concrete, often cause magnetometer measurements to be too unreliable for robotics applications.

Accelerometers: can estimate a globally-referenced "down" vector from the local gravitational field when used in orthogonal configurations on ground-based robots. Stationary robots can use this "down" vector to estimate pitch and roll and hence the local slope of the terrain. These accelerometer measurements can only constrain two of three DOF of orientation, which means they cannot estimate a robot's heading on flat terrain. Non-stationary robots can filter multiple sensors to estimate pitch and roll while driving, however inexpensive MEMS sensors are likely to produce noisy estimates.

External tracking: systems estimate pose using external infrastructure that has been previously installed and localized in a global frame. These off-board systems use a wide variety of sensing techniques and are typically complex. The tracking sensors are mounted externally distinguishing them from the map-based localization techniques discussed in Section 2.4.4. Common examples include Vicon suits (shown in Figure 1.5), where visual tracking to reflective IR markers produces 6 DOF pose estimates at 200 Hz with sub-centimeter accuracy. Less expensive visual tracking solutions use visual fiducial markers, such as [165, 166]. RF-based systems use radio beacons detected by many fixed receivers to produce range-only measurements. Given surveyed receiver locations, 3 DOF pose estimates can be produced. It is important to note that these external tracking systems required infrastructure to be installed and the site surveyed before use, a limitation if rapid or large-scale deployment is desired.

2.4.4. Map-Based Robot Localization

Section 2.3 describes various sensors that robots can use to measure their environment, and several map parameterizations used to represent it. These can be combined with motion models and odometry to provide map-based localization. Map-based localization describes techniques that use maps and on-board sensors to estimate the posterior pose distribution:

$$p\left(\mathbf{r}_t \mid \mathbf{r}_{t-1}, \mathbf{u}_t, \mathbf{z_t}, \mathbf{m}\right) \tag{2.27}$$

Where $\mathbf{r}_t$ is the current pose, given the previous pose, $\mathbf{r}_{t-1}$, odometry, $\mathbf{u}_t$, sensor measurements, $\mathbf{z}_t$, and map, $\mathbf{m}$. The maps may be given *a priori*, or built on-line as part of a SLAM session. Furthermore, the problem can be separated into *incremental* and *global* variations, depending on the prior knowledge of the robot's pose.

2.4.4.1. Incremental Map-Based Localization

The incremental, or local problem involves tracking and estimating a robot's pose with respect to its map. In an iterative *predict-update* cycle, the robot's pose is first estimated from its previous pose using its motion model. Observations are then predicted from this pose using the map and the sensor's measurement model. By comparing the predicted and actual observations, the estimator updates the robot's pose. This is a subset of the SLAM problem, where both pose and map are estimated together. Various Bayesian approaches to the SLAM problem are described in the next section. They can be reduced to localization-only estimators by fixing the map parameters.

Figure 2.11. Visual place recognition: Visual odometry (VO) and visual place recognition (PR) from a flying quadrotor MAV. After completing a 200 meter loop, PR found 10 potential matches to previously seen images. Results from my unpublished research. Video courtesy Cybertech.

2.4.4.2. Global Map-Based Localization

The global map-based localization problem attempts estimate a robot's pose in an existing map, without any prior knowledge of its pose. It is also described in the literature as *relocalization* or the "kidnapped robot" problem [167, 168, 135]. For a mobile robot in $\mathbb{SE}(2)$, this is potentially a large configuration space. Environments often have minimal defining geometry (e.g. a straight corridor with no doors), or self-similar geometry that create *perceptual aliasing* ambiguities (e.g. multiple identical offices that open onto a corridor) [123]. Global relocalization algorithms typically track multiple hypotheses, often over large distances, until sensor data can confirm a single pose hypothesis (or null hypothesis).

Monte Carlo Localization (MCL): is a Bayesian formulation where the pose probability distribution is represented by a set of weighted hypotheses (samples) [168, 169]. In an iterative predict-update cycle each sample is treated similar to the local problem above. As the robot moves, the samples are updated and sensor observations are predicted from the map. By comparing the predicted and actual observations, the samples' weights are updated according to the likelihood that the pose fits the observations. The samples representing the posterior distribution are re-sampled, and if a single hypothesis remains the global pose has been found.

Visual place recognition (PR): algorithms generate pose estimates from camera images. PR decomposes camera images into sets of 2-D visual features, or *bags-of-words*, that are inserted into a large database. A query image, possibly taken by a lost robot, is decomposed into a bag-of-words and compared against the database, from which candidates matches are retrieved [170, 171, 172, 173]. The visual words can be biased according to

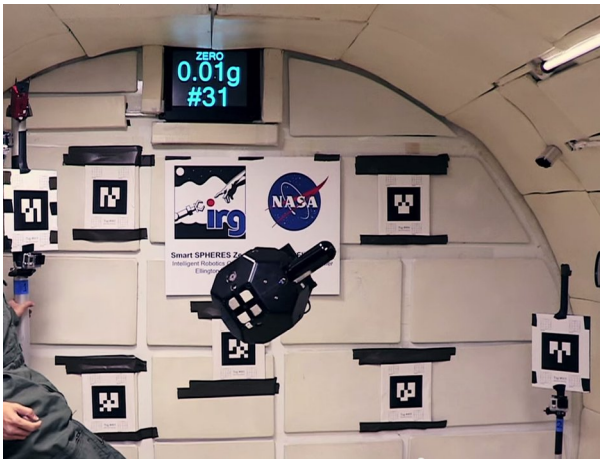


Figure 2.12. Map-based localization using visual fiducials: One of NASA's SPHERES robots, [175], floating in a zero-g aircraft. ALVAR visual tags allow accurate on-board map-based localization [174]. Image courtesy NASA Ames and Google ATAP.

their occurrence to encourage positive matches, however many matches are often returned. These multiple hypothesis are passed through geometry checks and motion models where they are culled as the robot continues to drive. PR only works in environments that have sufficient visual richness, and from locations and vantage points that are close to those that have previously been observed [172]. Figure 2.11 shows an example of PR running on a quadrotor after a 200 meters loop has been flown.

Fiducial markers: at the core of global relocalization is a data association problem. By augmenting a robot's environment, the data association problem can be made much easier (some examples are given in Section 2.4.3). Visual fiducial markers are described here since they are passive, provide perfect data association and have become popular over the last decade [165, 166, 174]. These 2-D bar codes, also called augmented reality tags (ARtags), are uniquely identifiable and include checksums for self-validation. A set of multiple visual fiducials, fixed around a robot's environment, create a landmark-based map that can be used for highly-accurate map-based localization. Figure 2.12 shows an environment augmented with 7 ALVAR tags [174]. Using an image from a calibrated camera the robot's full 6 DOF pose can be estimated accurately within the limited volume of its workspace.

2.4.4.3. Detecting and Verifying Loop Closures

When exploring a new environment with SLAM, a robot will typically alternate between growing its map by visiting new areas, and closing loops by revisiting previously seen areas. Loop closure detection is another form of the data association problem, introduced in Section 2.1. In SLAM sessions where sustained exploration grows the map without any loop closures, odometry noise can accumulate into very large pose uncertainties.

When attempting to close loops, these uncertainties and map search areas are similar to the global relocalization problem from the previous section. Degenerate and ambiguous geometry frequently requires loop closure hypotheses to be tracked until they can be verified with additional sensor data. Visual PR approaches, described in the previous section, have shown considerable promise in generating candidate loop closures given uncertain or nonexistent pose priors [176].

2.5. SLAM Algorithms

The SLAM problem was introduced in Section 2.1, along with the data association and loop closure problems, two key aspects that distinguish SLAM from a simple state estimation problem. Section 2.2 provided the mathematical framework for working with poses and measurements. Section 2.3 described sensors and parameterizations used to build maps of the environment, i.e. $p(\mathbf{m} \mid \mathbf{r}, \mathbf{z})$, while Section 2.4 described sensor models for robot motion and localization, i.e. $p(\mathbf{r}_t \mid \mathbf{r}_{t-1}, \mathbf{u}_t)$. These concepts were combined in Section 2.4.4 to enable map-based localization, i.e. $p(\mathbf{r}_t \mid \mathbf{r}_{t-1}, \mathbf{u}_t, \mathbf{z_t}, \mathbf{m})$. In probabilistic SLAM the map becomes a part of the estimation problem; a variable that is jointly estimated and refined along with the robot's pose, i.e:

$$p(\mathbf{r}, \mathbf{m} \mid \mathbf{r}_0, \mathbf{u}, \mathbf{z},) \tag{2.28}$$

Where the joint distribution of robot poses, $\mathbf{r}$, and the entire map, $\mathbf{m}$, are given from the history of odometry, $\mathbf{u}$, sensor measurements, $\mathbf{z}$, and starting pose $\mathbf{r}_0$.

SLAM maps are built from millions of sensor readings, matched against each other in a data association step that depends entirely on the current pose estimations. Evaluating and re-evaluating these data associations, while simultaneously estimating and updating the entire history of robot poses, describes the "full" SLAM problem. These data associations are combinatorial, and the problem is NP-complete [7, 8]. It is interesting to note that even with *given* data associations, a naive full SLAM algorithm scales at $\mathcal{O}((T+M)^3)$, for a trajectory length T and M map landmarks [177].

This section introduces the main approaches described in the literature for solving the SLAM problem. Each approach exploits various approximations and heuristics to run in polynomial time; they trade between *computational complexity*, *storage*, *accuracy*, *robustness* and *real-time execution*. While introducing the various approaches, I also note how well they scale to large environments and if they have been extended to the MR-SLAM problem.

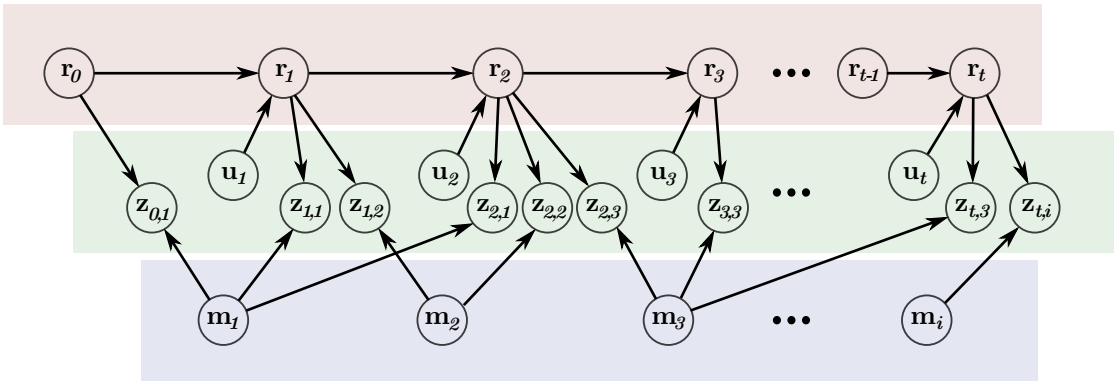


Figure 2.13. Graphical model for landmark-based SLAM: The red and blue boxes contain variables estimated in the SLAM state. The green box indicates measurements. Arrows indicate dependencies, or causal relationships. Robot pose estimates, $\mathbf{r}_t$, depend on the previous pose, $\mathbf{r}_{t-1}$, and odometry, $\mathbf{u}_t$. Map landmark estimates, $\mathbf{m}_i$, combined with the robot pose, cause the sensor measurement distributions $\mathbf{z}_{t,i}$.

2.5.1. SLAM Assumptions

While the full SLAM problem is unsolvable for non-trivial environments, approaches described in the literature frequently produce useful results in real-world environments. Two common assumptions reduce the complexity of the problem:

1. **Static environment:** maps do not vary over time; moving objects are segmented and treated as outliers that do not appear in the map.

2. **Sensor data is integrated sequentially:** robots move predictably, such that motion models can predict where a robot is likely to be, allowing the search for data associations, to begin close to the global optimum.

2.5.2. Full SLAM Graphical Model

The full SLAM Bayesian network, or *graphical model,* is introduced here as it is used to illustrate the structure of the simplifications employed in various SLAM algorithms. A thorough overview of graphical models in given by Bishop in [100]. Figure 2.13 provides the graph for the full SLAM probability distribution in Equation 2.28. This graph represents a *landmark-based* SLAM problem. It captures the robot pose, $\mathbf{r}_t$, and map landmarks, $\mathbf{m}_i$, which together form the SLAM state estimate (appearing in the red and blue boxes). Observations, including odometry measurements, $\mathbf{u}_t$, and sensor measurements, $\mathbf{z}_{t,i}$, appear in the green box. The structure of the graph indicates the dependencies between the various probability distributions. The arrows are most interpreted as causation, or "having an influence". For example, in Figure 2.13 the estimate of $\mathbf{r}_t$ is directly influenced by $\mathbf{r}_{t-1}$ and $\mathbf{u}_t$.

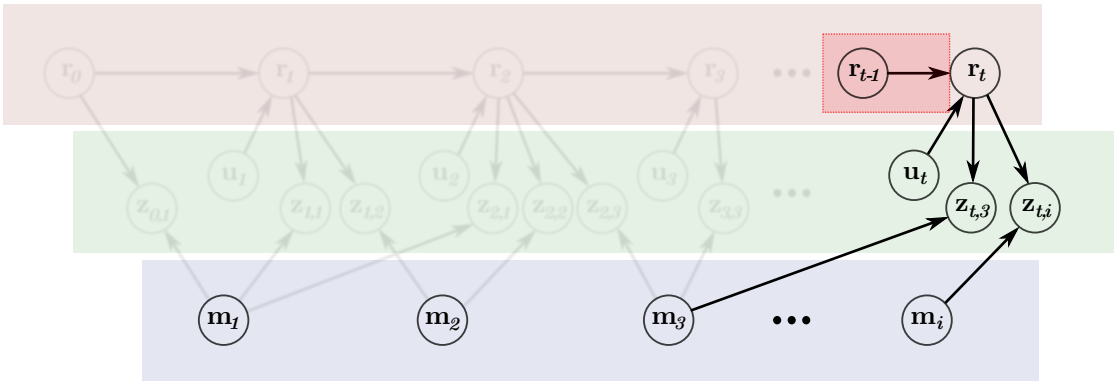


Figure 2.14. Graphical model with recursive Bayes filtering: At each time step t a predict-update cycle marginalizes out the previous pose $\mathbf{r}_{t-1}$ (red), and updates the state estimate based on current measurements, $\mathbf{u}_t$ and $\mathbf{z}_{t,i}$. Historical poses, measurements and data associations (gray) are "baked" into the state estimate.

The structure of the lower part of the graph in Figure 2.13 depends entirely on the path the robot drives, its sensor range and the shape of the environment. It is worth noting that in landmark-based SLAM the measurement dependencies, $p(\mathbf{z}_{t,i} \mid \mathbf{m}_i, \mathbf{r}_t)$, are the output of the data association step (see Section 2.3.4.2). This is a simple example— in a typical SLAM problem there are many orders of magnitude more landmarks and measurements.

2.5.3. Bayesian Filter-Based SLAM

2.5.3.1. Recursive Bayesian Filter SLAM

The first solutions to the SLAM problem were described in the 1980s by Smith and Cheeseman [9], based on established state estimation techniques [102]. The SLAM problem was treated as an on-line estimation problem using *recursive Bayesian filters*. In this approach the Bayes filter performs an iterative *prediction→update* cycle that integrates new sensor data, while marginalizing out the previous robot pose estimates, $\mathbf{r}_{t-1}$. A graphical model that describes SLAM with Bayes filters is shown in Figure 2.14, where old measurements and marginalized poses are grayed-out. The approach aims to maintain on-line performance by minimizing the growth of the state variable. It models the SLAM problem as a first-order discrete-time Markov chain, where the future is assumed independent of the past, given the current state. A Bayes filter for SLAM with an odometry-based motion model is given below, while Chen provides a more general tutorial for Bayes filters in [178].

The state at time t is parameterized by concatenating the robot pose estimate, $\mathbf{r}_t$, and the M map landmark estimates, $\mathbf{m}_i$, into a stochastic state vector, $\mathbf{x}_t$:

$$\mathbf{x}_t^{\mathcal{W}} = [\mathbf{r}_t, \mathbf{m}_1, \mathbf{m}_2, \cdots \mathbf{m}_M]^T \tag{2.29}$$

Coordinates are typically maintained in the global frame, $\mathcal{W}$, however the superscript is omitted for brevity. The map landmarks, $\mathbf{m}_i$, are assumed static, and so are not given a time subscript, however their positions are recursively estimated by the filter. Each iteration of the filter performs two steps: a prediction, followed by an update that produces a *posterior* state estimate:

Prediction: the current state is predicted using the prior belief $p(\mathbf{x}_{t-1} \mid \mathbf{z}_{0:t-1}, \mathbf{u}_{0:t-1})$, the latest odometry measurements and the robot's motion model (Section 2.4.1):

$$\underbrace{p(\mathbf{x}_t \mid \mathbf{z}_{0:t-1}, \mathbf{u}_{0:t})}_{\text{Prediction}} = \int \underbrace{p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t)}_{\text{Motion model}} \underbrace{p(\mathbf{x}_{t-1} \mid \mathbf{z}_{0:t-1}, \mathbf{u}_{0:t-1})}_{\text{Prior}} d\mathbf{x}_{t-1} \tag{2.30}$$

Here, using the *law of total probability*, the prior belief is marginalized out to form the prediction, $p(\mathbf{x}_t \mid \mathbf{z}_{0:t-1}, \mathbf{u}_{0:t})$. The prediction incorporates only odometry measurements and any assumptions made in the motion model (for example constant velocities).

Update: the prediction is updated using the sensor model and Bayes rule:

$$\underbrace{p(\mathbf{x}_t \mid \mathbf{z}_{0:t}, \mathbf{u}_{0:t})}_{\text{Posterior}} = \eta \underbrace{p(\mathbf{z}_t \mid \mathbf{x}_t)}_{\text{Sensor model}} \underbrace{p(\mathbf{x}_t \mid \mathbf{z}_{0:t-1}, \mathbf{u}_{0:t})}_{\text{Prediction}} \tag{2.31}$$

Here, $p(\mathbf{z}_t \mid \mathbf{x}_t)$ is the sensor model, or observation model. It describes the likelihood of an observation given the current state of the system. The normalization term, η is given by $\eta^{-1} = p(\mathbf{z}_t \mid \mathbf{z}_{0:t-1}, \mathbf{u}_{0:t})$. This term does not depend on x and in practice this term is not calculated; rather the posterior distribution is normalized to one [135].

In a Bayes filter the posterior state estimate at time t becomes the prior for the next iteration at $t+1$ in Equation 2.30. The filter is a framework for recursive state estimation. Several realizations are given in the following sections.

2.5.3.2. Extended Kalman Filter SLAM

The Kalman Filter (KF) is a well-known realization of a Bayes filter, first described in 1960 [102, 101]. The Kalman filter typically uses a multivariate Gaussian distribution to represent the system state, $\mathbf{x}_t$, with the assumption that the motion and sensor models are linear. In SLAM, however, these models are typically nonlinear and some form of linearization is required [179, 6].

The *Extended* Kalman Filter (EKF) linearizes the nonlinear models using a Taylor Series expansion around the current state estimate. The multivariate Gaussian is expressed in terms of the state variable's expected value $\hat{\mathbf{x}}_t = \mathbb{E}(\mathbf{x}_t)$ and its covariance $\mathbf{\Sigma}_t$:

$$\mathbf{x}_t = \mathcal{N}\left(\mathbf{x}_t | \hat{\mathbf{x}}_t, \mathbf{\Sigma}_t\right) \tag{2.32}$$

$$= \frac{1}{(2\pi)^{k/2} \left|\mathbf{\Sigma}_t\right|^{1/2}} \exp\left(-\frac{1}{2}\left(\mathbf{x}_t - \hat{\mathbf{x}}_t\right)^T \mathbf{\Sigma}_t^{-1} \left(\mathbf{x}_t - \hat{\mathbf{x}}_t\right)\right) \tag{2.33}$$

Here k is the dimension of the state variable, $\mathbf{x}_t$. The covariance, $\mathbf{\Sigma}_t$, is a square $k \times k$ block matrix that is positive-semidefinite:

$$\mathbf{\Sigma}_t = \begin{bmatrix} \mathbf{\Sigma}_r & \mathbf{\Sigma}_{r:m_1} & \cdots & \mathbf{\Sigma}_{r:m_M} \\ \mathbf{\Sigma}_{r:m_1}^T & \mathbf{\Sigma}_{m_1} & \cdots & \mathbf{\Sigma}_{m_1:m_M} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{\Sigma}_{r:m_j}^T & \mathbf{\Sigma}_{m_1:m_M}^T & \cdots & \mathbf{\Sigma}_{m_M} \end{bmatrix} \tag{2.34}$$

As with the general Bayes filter, the EKF iterates a predict-update cycle. A brief overview of the filter is given here using mathematical notation that is used throughout the rest of this thesis. Thrun et al. present EKF SLAM in depth in [135], chapter 10, while Paz et al. give an informative view of the matrix block structure in [180].

Prediction: the state prediction, $\hat{\mathbf{x}}_{t|t-1}$, is estimated using the motion model, and the posterior estimate from the last time step:

$$\hat{\mathbf{x}}_{t|t-1} = f\left(\hat{\mathbf{x}}_{t-1}, \mathbf{u}_t\right) \tag{2.35}$$

$$\mathbf{\Sigma}_{t|t-1} = \mathbf{F}_{t-1} \mathbf{\Sigma}_{t-1} \mathbf{F}_{t-1}^T + \mathbf{Q}_{t-1} \tag{2.36}$$

In a typical EKF SLAM implementation with static map landmarks, the motion model $f\left(\hat{\mathbf{x}}_{t-1}, \mathbf{u}_t\right)$ updates robot pose using odometry given by Equation 2.19. Using pose composition the motion model gives: $\hat{\mathbf{x}}_{t|t-1} = \hat{\mathbf{x}}_{t-1} \oplus \mathbf{u}_t^{t-1}$. The covariance estimate $\mathbf{\Sigma}_{t|t-1}$ is enlarged by the motion model's Jacobians $\mathbf{F}_{t-1} = \frac{\delta f}{\delta \hat{\mathbf{x}}_{t-1}}$, given in the appendix of [110], and the process noise $\mathbf{Q}_{t-1}$. The process noise results from the motion model, which is assumed to be additive, and is typically the same as the odometry noise.

Update: the predicted state is updated using the measurement residual, $\tilde{\mathbf{y}}_t$, the difference between the sensor measurements, $\mathbf{z}_t$, and the measurements predicted by the sensor model $h(\hat{\mathbf{x}}_{t|t-1})$:

$$\tilde{\mathbf{y}}_t = \mathbf{z}_t - h\left(\hat{\mathbf{x}}_{t|t-1}\right) \tag{2.37}$$

$$\mathbf{S}_t = \mathbf{H}_t \mathbf{\Sigma}_{t|t-1} \mathbf{H}_t^T + \mathbf{R}_t \tag{2.38}$$

$$\mathbf{K}_t = \mathbf{\Sigma}_{t|t-1} \mathbf{H}_t^T \mathbf{S}_t^{-1} \tag{2.39}$$

The residual covariance, $\mathbf{S}_t$, indicates how certain we are that the sensor model is going to match the actual measurements. Here the sensor model's Jacobians $\mathbf{H}_t = \frac{\delta h}{\delta \hat{\mathbf{x}}_{t|t-1}}$ apply another linearization, and the sensor model noise is incorporated through the covariance $\mathbf{R}_t$. The Kalman gain, $\mathbf{K}_t$, and the measurement residual are used to calculate the posterior estimate:

$$\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t \tilde{\mathbf{y}}_t \tag{2.40}$$

$$\boldsymbol{\Sigma}_t = (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t)\, \boldsymbol{\Sigma}_{t|t-1} \tag{2.41}$$

The posterior $\hat{\mathbf{x}}_t$ contains updated estimates for the state vector in Equation 2.29, which includes both the robot's pose and M landmarks.

The errors introduced by linearizing and accumulating the odometry are described in detail in Section 2.4.2.3. At each time-step the EKF marginalizes out the previous estimate, $\hat{\mathbf{x}}_{t-1}$, which effectively "bakes" the most recent linearization errors and data associations into the state. While repeated observations of static landmarks can help constrain pose errors that accumulate due to linearization and odometry errors, EKF SLAM will always underestimate uncertainties and eventually become inconsistent [155, 156]. Alternative formulations for EKF SLAM have been proposed over the last decade. Castellanos et al. proposed a robot-centered coordinate system to delay the inconsistency [155, 181]. This parameterization has gained wider adoption [182, 125].

The unimodal representation of the posterior means that data associations must be decided, marginalized and "baked-in" at every time-step, with no opportunity to track alternate hypotheses. Recognizing that a single data association error can cause EKF SLAM to become immediately inconsistent, significant research has gone into efficiently detecting and validating correspondences. Neira and Tardós' joint compatibility branch and bound (JCBB) [144] and Bailey's combined constrained data association (CCDA) [142] are two approaches that search the combinatorial solution space efficiently. The residual covariance in Equation 2.38 provides useful hints where to actively search for landmark matches, and their relative spatial arrangement [183, 184].

EKF SLAM scales with $\mathcal{O}(M^2)$, where M is the number of landmarks in the map [142, 104]. While sparse Jacobians and limited sensor range allows for some optimizations, the covariance $\boldsymbol{\Sigma}_t$ becomes increasingly dense, limiting on-line execution to several hundred landmarks [180]. Specialized implementations using FPGAs have been demonstrated with 1800 features [185], however this solution scales with silicon size, which is ultimately still limited.

To scale to larger map sizes, EKF SLAM maps can be broken into many smaller "submaps" (Section 2.5.5). Several EKF-based submapping techniques have been described, including

decoupled stochastic mapping [186], postponement [187] and compressed EKFs [141], which while suboptimal, maintain on-line performance by restricting map updates to the landmarks around the robot. The map joining algorithm by Castellanos et al. is more consistent [181], however like the other techniques, the global state and covariance updates are still $\mathcal{O}(M^2)$, ultimately limiting global map size. Piniés et al. describe CI-Graph, a landmark-based submapping approach that uses conditional independence to efficiently subdivide an EKF [188]. Paz et al.'s divide-and-conquer technique uses a hierarchical composition that amortizes the $\mathcal{O}(M^2)$ cost, resulting in linear time EKF SLAM in some situations [180]. Section 2.5.5 describes how EKF submaps have also been used as the basis for graph-based SLAM techniques.

While EKF techniques can be used for multi-robot SLAM, submapping is generally required to maintain on-line operation. The additional computational cost depends on the robot's sensor overlap and landmark density. For highly-overlapping configurations, none of the previously described techniques scale more efficiently than $\mathcal{O}(M^2)$. Williams et al. apply their constrained local submap filter to a multi-robot simulation [189, 190]. Madhavan et al. describe a two robot EKF localization approach, however the mapping is not performed simultaneously [26]. Zhou and Roumeliotis describe a landmark-based EKF SLAM solution where both robots' maps are merged after each rendezvous [191]. A few years after the nonlinearity and inconsistency problems were well documented [155, 156], Huang et al. demonstrated the same issues in multi-robot EKFs and proposed a solution that applied more conservative uncertainty reduction [192].

Ultimately the $\mathcal{O}(M^2)$ scaling problem restricts large-scale multi-robot EKF deployments. Regardless of the type of landmarks used, EKFs are relatively limited in the size of the environment and accuracy of the geometry they can map. In light of this, I am surprised that researchers have continued to publish EKF-based SLAM solutions, a view shared by others [193].

2.5.3.3. Unscented Kalman Filter SLAM

Julier and Uhlmann's Unscented Kalman Filter (UKF) is briefly described here since it very elegantly addresses the linearization errors in the EKF [194]. First described in 2000, the UKF propagates a small set of samples, or sigma points, through the nonlinear motion and measurement models, to recover approximate estimates with covariances. The UKF avoids the need to calculate Jacobians and is no more complex than the EKF [195]. It has been shown to produce much smaller linearization errors in many difficult problems, including monocular SLAM [196]. UKFs, however, suffer from the same $\mathcal{O}(M^2)$ scaling issues as the EKF, and the approximations eventually cause the filter to become inconsistent [156].

2.5.3.4. Covariance Intersection SLAM

Julier and Uhlmann introduced Covariance Intersection (CI) into the Bayesian fusion literature in the late 1990s [197]. CI is a conservative approach to data fusion that combines two estimates when the correlations between them are unknown. Two Gaussians, with distributions $\mathcal{N}(\mu_a, \mathbf{\Sigma}_a)$ and $\mathcal{N}(\mu_b, \mathbf{\Sigma}_b)$, are fused into a single Gaussian, $\mathcal{N}(\mu_c, \mathbf{\Sigma}_c)$, using the CI rule:

$$\begin{aligned} \mathbf{\Sigma}_c^{-1} &= \beta \mathbf{\Sigma}_a^{-1} + (1-\beta)\, \mathbf{\Sigma}_b^{-1} \\ \mu_c &= \mathbf{\Sigma}_c \left(\beta \mathbf{\Sigma}_a^{-1} \mu_a + (1-\beta)\, \mathbf{\Sigma}_b^{-1} \mu_b \right) \end{aligned} \tag{2.42}$$

The coefficient β is bound by the interval $[0, 1]$ and is typically optimized to meet some desired uncertainty criteria. In [198] Julier and Uhlmann demonstrate CI-SLAM using a modified form of the Kalman filter, where CI is used to minimize computational by removing covariances.

2.5.3.5. Extended Information Filters

The Extended Information Filter (EIF) is the mathematical inverse of the EKF. It was first applied to the SLAM problem by Thrun et al. in 2002 [199], who had the insight to replace the covariance matrices with their *information form*. Sometimes called the *canonical form*, the multivariate Gaussian state vector, $\hat{\mathbf{x}}_t$, and its covariance, $\mathbf{\Sigma}_t$, are replaced by the information vector, $\hat{\mathbf{y}}_t = \mathbf{\Sigma}_t^{-1} \hat{\mathbf{x}}_t$, and information matrix, $\mathbf{\Omega}_t = \mathbf{\Sigma}_t^{-1}$, respectively.

In the information form, new measurements become simple block additions that can be performed in linear time [200]. This avoids the $\mathcal{O}(M^2)$ cost that the EKF incurs when performing measurement updates, however the information matrix grows in size. The standard EIF algorithm marginalizes previous poses, which gradually makes the information matrix dense. Thrun et al. describe the Sparse EIF (SEIF), which includes a constant-time sparsification step that approximates the information matrix to maintain sparseness over long-term operation [199]. While impressive results have been demonstrated with the SEIF, e.g. [201], the limitations of landmark-based maps remain.

Eustice et al. described the Exactly Sparse Delayed-State Filter (ESDF) [200], which maximizes sparseness in the information matrix by not marginalizing out old poses. They describe a "view-based" approach that aligns groups of measurements acquired from a each pose (Section 2.3.4.1). The map is not explicitly parameterized in the ESDF, avoiding the limitations of landmark-based maps.

ESDFs have been shown to scale to large map sizes. Mahon et al. demonstrated an ESDF-based visual SLAM system mapping a 150×150 meter area on-line. Their final map had

2,200 poses covering a 2.2 km trajectory [202]. Multi-robot SEIF SLAM was demonstrated by Thrun and Liu in [203], however they only showed the Victoria Park dataset [140] cut into 8 pieces.

While EIF updates can be performed in $\mathcal{O}(M)$ linear time, it becomes expensive to recover the covariances that are used in data association. In the SEIF and ESDF, it has been shown how the information matrix's sparseness can been exploited to avoid the $\mathcal{O}(M^3)$ cost of matrix inversion [201, 200].

Like the EKF, however, the various forms of EIFs linearize the motion and sensor models around current estimates. These approximations are "baked" into the information matrix, which can eventually lead to inconsistencies [200]. It is interesting to note that the ESDF's sparse information matrix is closely related to the Hessian matrix in the pose-graph SLAM formulations used throughout this research (Section 2.5.4). The pose graph formulations differ in their continuous re-linearization of measurements.

2.5.3.6. Tree-Based Filter SLAM

Closely related to the SEIF, Paskin's Thin Junction Tree Filter (TJTF) [204] uses variable contraction to maintain sparsity in SLAM constraints. First described in 2002, the filter builds an approximation of the SLAM state with a tree structure that is periodically "thinned" by contraction. This process, like the sparsification of SEIFs is supported theoretically by the weakening of links between distant landmarks [205]. Like SEIFs, TJTFs operate in $\mathcal{O}(M)$ linear time, however they can approximate global maps without matrix inversion.

Several years later, Frese described the closely related "Treemap" algorithm [206]. Treemap uses a hierarchical local-global map representation similar to EKF submapping approaches, however global updates are performed in $\mathcal{O}(K^3 \log M)$, where K is the number of local landmarks. It has been demonstrated on some impressive simulated datasets, including a sparse 350×350 meter area in which it took 21 ms to close a loop with 48,700 poses and over a million landmarks [207].

To the best of my knowledge there are no real-world multi-robot demonstrations of tree-based filters in the literature. In the case of 2-D lidar-based SLAM this may be due to the limited representations that landmark-based maps offer in real-world environments.

2.5.3.7. Particle Filter SLAM

Particle filters overcome many of the limitations of the filter-based SLAM solutions described so far. Nonlinear motion models and multimodal distributions can be represented

by a set of samples that allows multiple hypotheses to be tracked [208, 209]. Recognizing that map landmarks are conditionally independent given the robot's pose, Rao-Blackwellized Particle Filters (RBPFs) factorize the SLAM problem by separating the robot's complete trajectory, $\mathbf{r}_{0:t}$, from the map $\mathbf{m}$:

$$\underbrace{p\left(\mathbf{r}_{0:t}, \mathbf{m} \mid \mathbf{z}_{0:t}, \mathbf{u}_{0:t}\right)}_{\text{Posterior}} = \underbrace{p\left(\mathbf{m} \mid \mathbf{r}_{0:t}, \mathbf{z}_{0:t}\right)}_{\text{Map}} \underbrace{p\left(\mathbf{r}_{0:t} \mid \mathbf{z}_{0:t}, \mathbf{u}_{0:t}\right)}_{\text{Trajectory}} \tag{2.43}$$

Equation 2.43 allows the map $p\left(\mathbf{m} \mid \mathbf{r}_{0:t}, \mathbf{z}_{0:t}\right)$ to be efficiently computed from a complete trajectory estimate. The trajectory, $p\left(\mathbf{r}_{0:t} \mid \mathbf{z}_{0:t}, \mathbf{u}_{0:t}\right)$, is then estimated with a particle filter, where each particle represents a complete trajectory (and hence complete map). The particles are subsequently *weighted* according to how well the measurements agree with the map, and *resampled* to approximate a continuous distribution. RBPFs differ from the previously described filter-based solutions, rather than marginalizing old poses in the filter update, the entire trajectory is filtered.

The richness of a RBPF's multimodal distribution is controlled by the number of particles, often between 10 and 100. The computational cost is linear in the number of particles, thus most research effort has focused on minimizing particles while maintaining accurate posterior distributions.

In 2002 Montemerlo et al. demonstrated FastSLAM, a landmark-based RBPF SLAM implementation where the independent landmarks, $p\left(\mathbf{m}_i \mid \mathbf{r}_{0:t}, \mathbf{z}_{0:t}\right)$, are trivially estimated with a set of 2-D EKFs [210, 211, 212]. They demonstrated SLAM updates in $\mathcal{O}\left(N \log M\right)$ time, where N is the number of particles. FastSLAM has been demonstrated on several large datasets, including the Victoria Park dataset where it was 25 times faster than an EKF.

Unfortunately, the number of particles required to maintain accurate posterior distributions increases with the map size, the complexity of the environment and number of robots. In recent work Grisetti et al. demonstrated large-scale RBPF SLAM using lidar scan matching instead of landmarks [213]. They showed impressive results over a 1.7 km trajectory through a 250×250 meter area in real-time. For several datasets they search for the minimum number of particles required to produce topologically correct maps, and confirm that larger and more complex environments require more particles. Each time the particles are weighted and resampled information is lost and randomness is introduced. Even with thousands of particles, a growing RBPF map will accumulate errors and eventually become inconsistent [157].

Attempts to use RBPFs for multi-robot SLAM have met limited success due to the large number of particles required to avoid inconsistencies. Howard demonstrated multi-robot SLAM with RBPFs in a 45×25 meter area, first with the robots' initial poses known, and

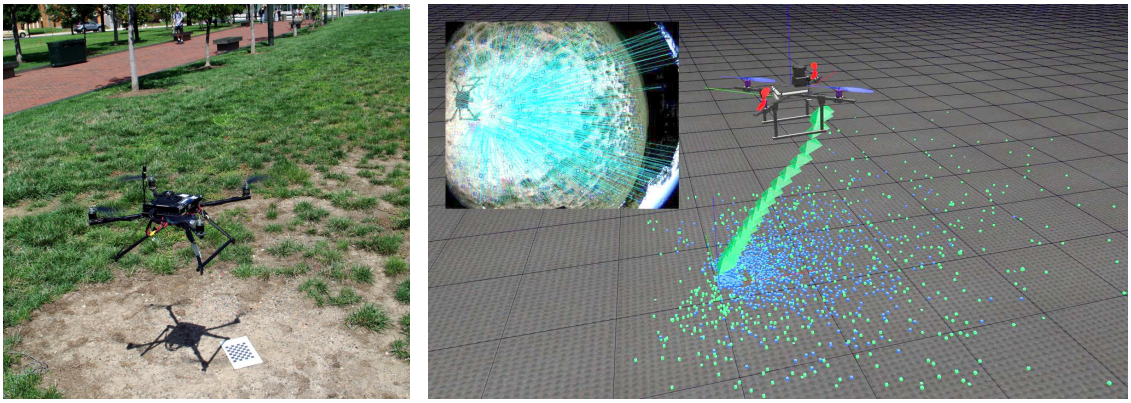


Figure 2.15. Visual SLAM with a sliding window filter: Visual SLAM on a quadrotor. **Left:** photo at take-off. **Right:** 3-D visualization with camera poses are shown as green pyramids, marginalized landmarks are blue dots, while currently estimated landmarks are shown in green. **Right Inset:** wide FOV camera view showing epipolar lines [111]. These images are from other research not included in this thesis.

then unknown [50]. In the latter case, each robot performed single robot SLAM until a rendezvous, at which point the particles and maps were transformed and merged. Any inconsistencies in the individual robots' maps results in an inconsistent global map after merging. Fox et al. used RBPFs in a MR-SLAM implementation, however only to estimate the initial relative poses [214, 27].

The minimum number of particles required for MR-SLAM is likely to be proportional to the number of robots. This gives a minimum computation cost of $\mathcal{O}(RN \log M)$, where R is the number of robots. While the additional computation could be distributed over these R robots, it remains difficult to merge globally consistent maps. Carlone et al. describe such a distributed RBPF approach, however present limited results with two robots in a small 20×10 meter environment [215].

2.5.3.8. Sliding Window Filter SLAM

Most filter-based SLAM algorithms, such as the EKF, minimize complexity by marginalizing out previous poses. This is done at the cost of accuracy when compared to the full SLAM solution of T poses. Recognizing that most of the quadratic computational complexity comes from the M map landmarks, sliding window filter (SWF) approaches aim to increase accuracy by adding the K most recent poses to the filter also. If $M \gg K$, the additional cost is minimal. Sibley et al. demonstrate a SWF with stereo visual odometry in [216]. With $K = 1$ their approach performs the same as an EKF, while for $K = T$ the SWF approaches the full SLAM solution. This work is important as it provides a continuum between filtering-based approaches and the graph-based smoothing approaches described in the next section.

Newman et al. show impressive SLAM results using the SWF on the 2.2 km New College mobile robot dataset [217]. Bibby and Reid describe reversible data association within SWF with $K = 6$, demonstrating the ability to revisit data associations over the limited time window [218]. Mei et al. describe relative SLAM in [219, 220], a stereo vision system that that avoids creating a global coordinate frame and optimizing large loop closures. A "double window" constant time visual SLAM algorithm is described by Strasdat et al. in [221], where the inner window is a SWF.

2.5.4. Graph-Based SLAM

Graph-based SLAM techniques use *maximum likelihood estimation* (MLE) to find the best configuration of a robot's pose history that aligns sensor observations. In graph-based techniques, robot poses form variable nodes in a graph, joined by probabilistic *factors* or *constraints* that determines the spatial relationship between them. Often referred to as *factor graphs,* these graph-based representations have been shown to scale well to both large-scale SLAM and multi-robot SLAM. Graph-based SLAM techniques are used extensively in this research work. This section gives a historical account of graph-based SLAM techniques, reviews current work and then describes, in detail, how to optimize graphs with MLE.

One of the most compelling reasons to use graph-base approaches is that they constantly re-linearize around their current estimate, unlike filters such as the EKF and EIF that "bake" linearization errors into their state. Performance is generally closer to the ideal full SLAM solution (Section 2.5.2), however graph-based approaches still include approximations to operate in polynomial-time.

One such approximation is to omit the map itself from estimation, and only optimize the history of robot poses. These so-called *pose graphs* leverage an approximation frequently used in SLAM: that measurements are conditionally independent given the robot's pose. For relatively accurate sensors, such as lidar scanners, this approximation holds well, so that the majority of uncertainty can be attributed to the robot's pose. The notion that observations from the same pose produce a particular "view" of the environment led to pose graph methods being introduced as "view-based" mapping [202, 222].

While the constraint factors in graph-based SLAM implicitly encode posterior distributions, obtaining the full joint posterior distribution for the entire graph is very expensive. Instead, graph optimization algorithms produce a single maximum likelihood estimate [135], and marginal covariances are only recovered if and when required [223].

2.5.4.1. Previous Work

There are several historical approaches to graph-based SLAM. The key computation typically requires linearizing a nonlinear least-squares problem, and solving it with a first or second order method. Consistent pose estimation (CPE) was first introduced in 1997 by Lu and Milios [224]. Their method directly inverted square matrices with a $\mathcal{O}(T^3)$ cost. Gutmann and Konolige demonstrated on-line incremental CPE with loop closure in small environments [225]. Graph-based SLAM was first demonstrated with multiple robots by Howard et al. in 1999, [226, 227]. Section 3.2.3 provides a review of previous work specific to MR-SLAM.

In 2006, Thrun and Montemerlo coined the term Graph SLAM for their EM-based large-scale urban mapping work in [228]. Their landmark-based approach was demonstrated off-line with a 600×800 meter dataset. At around the same time the SLAM community began considering graph-based methods as viable *on-line* alternatives to EKF and other filtering approaches.

There are two main methods for solving the least-squares optimization problem [229]; first order gradient-based *iterative methods*, and second-order *direct methods* are briefly described here:

Iterative methods: many gradient-based approaches have been proposed, including Howard et al. who used Gauss-Seidel relaxation [230]. Konolige demonstrated Preconjugate Gradient Descent (PCG) in large loopy environments with $\mathcal{O}(T \log T)$ time complexity [231]. Folkesson and Christensen described a trajectory-only iterative gradient descent optimization in [232]. Olson et al. demonstrated a modified SGD algorithm that converges quickly from poor initial estimates [233, 109].

Direct methods: insights into the structure of factor graphs, combined with better solvers, have led to dramatic increases in the efficiency of direct methods. Researchers noted structural similarities with the bundle adjustment (BA) problem from photogrammetry and computer vision [234]. In 2006 Dellaert and Kaess described Square Root SAM [7], which compared various factorization that exploited sparsity in the batch least-squares optimization problem. Their incremental Smoothing and Mapping (iSAM) work [235, 236], used sparse QR matrix factorizations to demonstrate on-line SLAM with costs varying from $\mathcal{O}(T)$ to $\mathcal{O}(T^2)$.

Further efficiencies were demonstrated by Konolige et al., who introduced Sparse Pose Adjustment (SPA) [237], which uses fast Cholesky factorization. In 2011 Kümmerle et al. released a generic graph-based optimization framework called g^{2}o, which is more efficient than both SPA and iSAM on many types of SLAM problems [238].

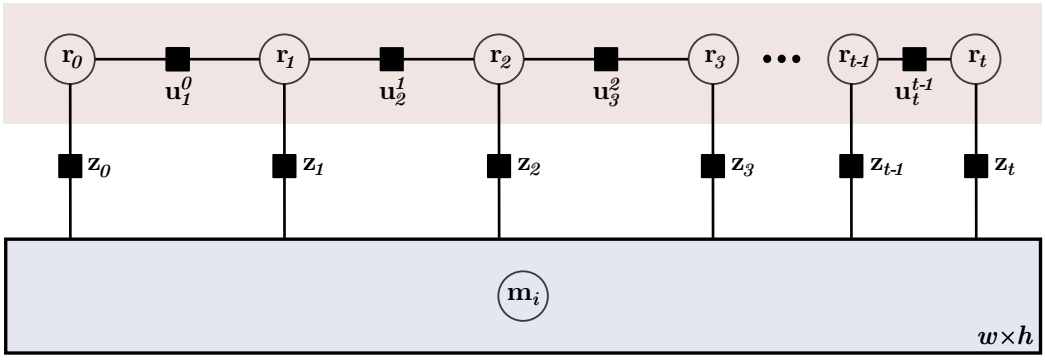


Figure 2.16. Factor graph for occupancy gridmap SLAM: Full SLAM factor graph with plate notation [100], the map is represented by an occupancy gridmap with cells, $\mathbf{m}_i$. Variable nodes are large circles, while factor nodes are squares. Robot poses, $\mathbf{r}_t$, are variable nodes that are separated by either odometry factors, $\mathbf{u}_t^{t-1}$, or measurement factors, $\mathbf{z}_t$.

Kaess et al. recently described iSAM2 [239], which introduces a Bayes Tree structure that is more efficient than Cholesky factorization on most single robot problems. It is not known how this approach performs in large MR-SLAM problems, especially when multiple distributed robots are producing frequent structural changes across the entire pose graph. Hybrid iterative-direct methods have been demonstrated [229], along with submapping approaches, which are discussed in Section 2.5.5. Research in graph-based SLAM has been greatly assisted by the release of the open source frameworks described here[3]. The main factorizations and optimization techniques are described in the rest of this section.

2.5.4.2. Factors Graphs

Factor graphs are bipartite graphical models [240, 100] that are well suited for describing graph-based SLAM problems. Figure 2.16 shows a full SLAM factor graph with an occupancy gridmap of cells $\mathbf{m}_i$. Factor graphs have two node types: *variable nodes* and *factor nodes*. Graph edges joining variable nodes are always separated by a single factor node. In graph-based SLAM, robot pose estimates, $\hat{\mathbf{r}}_t$, are separated by factors (or constraints) generated from either odometry or sensor measurements.

Factors can have arbitrary probability distributions, however in this section we only consider unimodal Gaussians (Chapter 6 describes a solution for multimodal Gaussians). Bayesian graphical models, such as the one in Figure 2.13, are easily converted into factor graphs. Dellaert provides a tutorial in [241].

[3] Available on-line: http://www.openslam.org

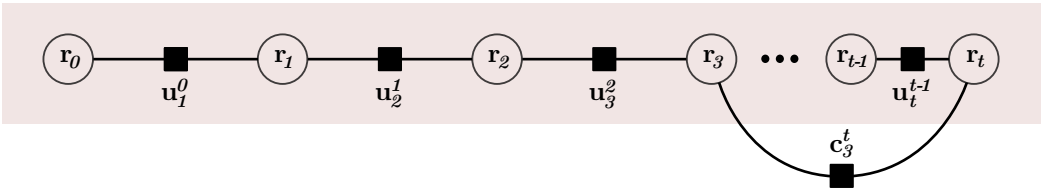


Figure 2.17. Factor graph with a loop closure: A pose-graph SLAM estimation problem where the robot poses, $\mathbf{r}_t$, are the variable nodes, separated by either odometry factors, $\mathbf{u}_t$, or loop closure factors, $\mathbf{c}_k^j$. A typical pose graph has many more loop closures.

2.5.4.3. Pose Graphs

Pose graphs are a type of factor graph where only the robot's poses are estimated. The map, $\mathbf{m}_i$, and raw sensor measurements, $\mathbf{z}_{t,i}$, from Figure 2.16 are marginalized out of the estimation, resulting in the minimal graph shown in Figure 2.17. In a pose graph, robot poses, $\mathbf{r}_t$, are linked either by odometry constraints, $\mathbf{u}_t^{t-1}$ (Section 2.4.2), or loop closure constraints, $\mathbf{c}_k^j$ (Section 2.4.4). Pose graphs are compact representation, which makes them well suited for large environments with multiple robots and frequent loop closures.

As an example, Figure 2.18 shows a single loop closure in the example problem described in Section 2.1.4. Lidar scans acquired at the *exact* poses $\mathbf{r}_3$ and $\mathbf{r}_{49}$ are aligned with a scan-matching algorithm based on ICP (Section 2.3.4.1). This relative $\mathbb{SE}(2)$ measurement becomes the Gaussian constraint $\mathbf{c}_3^{49} = \mathcal{N}(\mathbf{c}_3^{49}, \mathbf{\Sigma}_{49,3})$ when combined with covariance estimate $\mathbf{\Sigma}_{49,3}$. A similar loop closing factor, $\mathbf{c}_3^t$, is shown in Figure 2.17. Techniques for generating robust loop closure constraints are discussed in Section 5.4.

2.5.4.4. Spatial Constraint Factors

Spatial constraint factors encode distributions over the variable nodes they are connected to. In pose graphs, these factors represent $\mathbb{SE}(2)$ spatial constraints. Odometry measurements, $\mathbf{u}_t^{t-1} = [x_u, y_u, \phi_u]^T$, are added as factors joining poses $\mathbf{r}_{t-1}$ and $\mathbf{r}_t$ with the distribution $\mathcal{N}(\mathbf{u}_t^{t-1}, \mathbf{\Sigma}_u)$. Similarly, loop closing constraints, $\mathbf{c}_k^j = [x_c, y_c, \phi_c]^T$ form factors joining poses $\mathbf{r}_j$ and $\mathbf{r}_k$ with the distribution $\mathcal{N}(\mathbf{c}_k^j, \mathbf{\Sigma}_{j,k})$. Spatial constraint factors define a "virtual" *sensor model*:

$$z\left(\mathbf{r}_j, \mathbf{r}_k\right) = \ominus \mathbf{r}_j^{\mathcal{W}} \oplus \mathbf{r}_k^{\mathcal{W}} \tag{2.44}$$

The sensor model in Equation 2.44 predicts the constraint measurement based on the current pose estimates $\mathbf{r}_j^{\mathcal{W}}$ and $\mathbf{r}_k^{\mathcal{W}}$.

Figure 2.19 shows an enlarged view of the example problem at the moment the single loop closure is detected. Before optimization, the green robot poses $\mathbf{r}_3$ and $\mathbf{r}_{49}$ are the

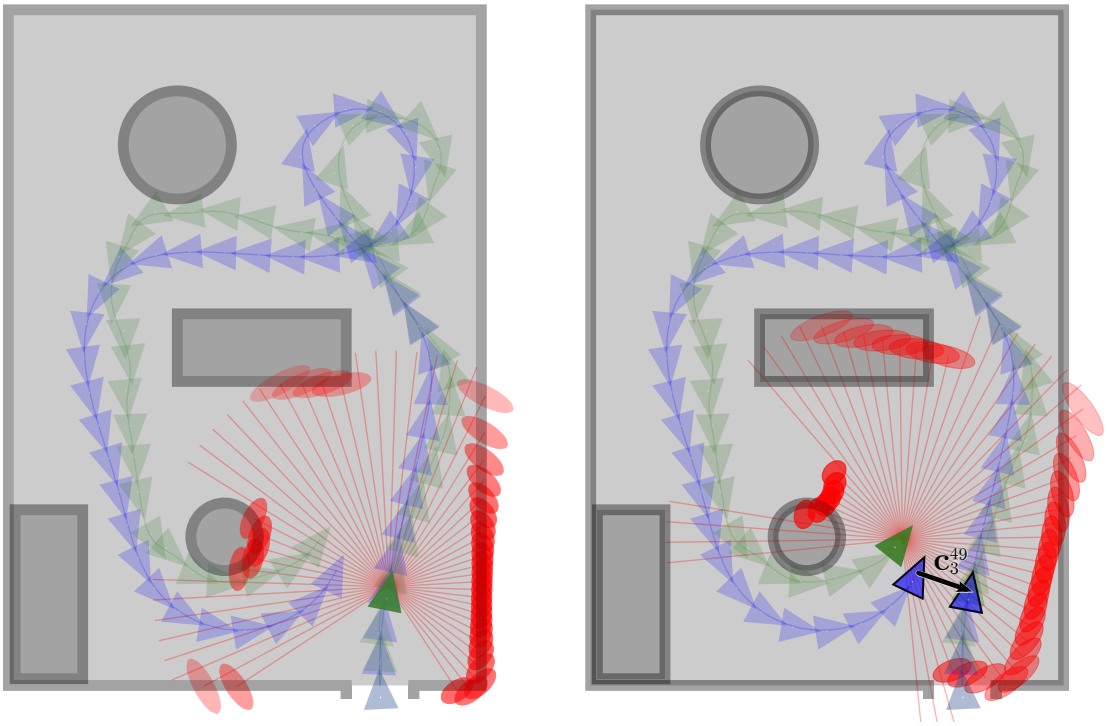


Figure 2.18. Loop closure constraints: A loop closure is found in the example problem. Blue triangles are the exact robot pose, green triangles are estimates, red ellipses are lidar measurements. **Left:** $t = 3$. **Right:** $t = 49$. After completing a loop of the room, a large amount of odometry noise has accumulated. Note the similarity between the two lidar scans (red): the loop closure constraint $\mathbf{c}_3^{49}$ is the estimated transform required to align them.

noisy pose estimate based purely on accumulated odometry. The virtual sensor model, $z\left(\mathbf{r}_{49}, \mathbf{r}_3\right) = \ominus\mathbf{r}_{49} \oplus \mathbf{r}_3$, predicts the constraint transform, which includes this accumulated noise. If the pose graph is well-formed, after a few iterations of the optimizer the constraint and sensor model prediction should be approximately equal, i.e. $\mathbf{c}_3^{49} \approx \ominus\mathbf{r}_{49} \oplus \mathbf{r}_3$ and the accumulated pose errors reduced.

During the maximum likelihood optimization, the spatial constraints act like *springs*. Each constraint pushes and pulls on the poses it connects to, attempting to make the relative $\mathbb{SE}\left(2\right)$ transform equal the constraint, i.e. $z\left(\mathbf{r}_j, \mathbf{r}_k\right) \approx \mathbf{c}_k^j$. The error between the sensor model prediction and the measured constraint defines the residual error function:

$$\mathbf{e}_{j,k} = e(\mathbf{r}_j, \mathbf{r}_k, \mathbf{c}_k^j) = \mathbf{c}_k^j - z\left(\mathbf{r}_j, \mathbf{r}_k\right) \tag{2.45}$$

Careful attention must be paid to ensure the residual is defined in the same frame as the constraint, i.e. in $\mathbf{r}_j$. During optimization the magnitude of the residual corresponds to the amount the constraint spring has been stretched or squashed from neutral. The

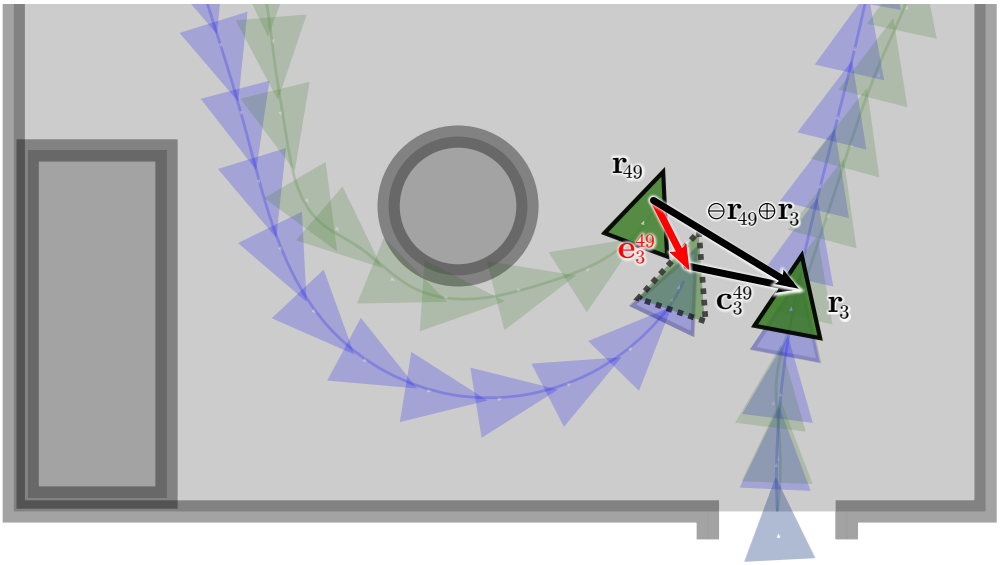


Figure 2.19. Residual errors in loop closure constraints: The example problem at the moment the loop closure is detected. Noisy pose estimates, $\mathbf{r}_3$ and $\mathbf{r}_{49}$, are shown in green. A "virtual" sensor model predicts the constraint transform $f(\mathbf{r}_{49}, \mathbf{r}_3) = \ominus\mathbf{r}_{49} \oplus \mathbf{r}_3$ based on the noisy poses. After optimization the prediction, $\ominus\mathbf{r}_{49} \oplus \mathbf{r}_3$, and the constraint measurement, $\mathbf{c}_3^{49}$, should be approximately equal, i.e. the residual error, $\mathbf{e}_3^{49}$, in red should be minimized.

strength of the spring, or its reaction force, is defined by the covariances $\mathbf{\Sigma}_{j,k}$, which if well modeled produce the residual distribution:

$$\mathcal{N}(\mathbf{e}_{j,k}, \mathbf{\Sigma}_{j,k}) = \frac{1}{(2\pi)^{3/2}\,|\mathbf{\Sigma}_{j,k}|^{1/2}} \exp\left(-\frac{1}{2}\mathbf{e}_{j,k}^T \mathbf{\Sigma}_{j,k}^{-1} \mathbf{e}_{j,k}\right) \tag{2.46}$$

2.5.4.5. Constraint Cost Function

Maximum likelihood estimation attempts to find the best pose graph configuration that satisfies all constraints simultaneously. Using the spring analogy from the previous section, the pose graph reaches an optimum (local or global) when all of the spring forces and torques are balanced and the poses are stable. Constraint residual errors cannot be compared directly because they have different spring strengths (covariances). Instead, each constraint is assigned a *cost function* that is the Mahalanobis distance squared [242]:

$$D^2 = \mathbf{e}_{j,k}^T \mathbf{\Sigma}_{j,k}^{-1} \mathbf{e}_{j,k} = \|\mathbf{e}_{j,k}\|^2_{\mathbf{\Sigma}_{j,k}} \tag{2.47}$$

In the special case where $\mathbf{\Sigma}_{j,k} = \text{diag}(\sigma_x^2, \sigma_y^2, \sigma_\phi^2)$, the cost function becomes the normalized Euclidean distance squared, i.e. $\|\mathbf{e}_{j,k}\|^2_{\mathbf{\Sigma}_{j,k}} \sim \frac{x_{j,k}^2}{\sigma_x^2} + \frac{y_{j,k}^2}{\sigma_y^2} + \frac{\phi_{j,k}^2}{\sigma_\phi^2}$. An intuitive interpretation of Equation 2.47 is that the constraint cost is the sum of the residual's components after being squared and normalized by the covariance. Extending this to the spring analogy: the constraint cost is how much the spring has been deflected, normalized by the spring's stiffness. Each constraint's cost provides a measure of how well its residual fits inside

its covariance. Constraint costs are expected to exhibit a *chi-squared* distribution of dimension 3, or $\chi^2(3)$.

In SLAM, the 99.7 % confidence interval, or 3-σ uncertainty bound, is often treated as a significant boundary between inliers and outliers. For $\mathbb{SE}(2)$ constraints this 3-σ boundary forms an ellipsoid-shaped volume in $\mathbb{R}^3$, where the boundary is given by $\|\mathbf{e}_{j,k}\|^2_{\mathbf{\Sigma}_{j,k}} = \chi^2_{0.997}(3) \approx 14.16$. To complete the spring analogy; a stiff spring equates to a small 3-σ covariance ellipsoid, while a more flexible spring has a larger 3-σ ellipsoid.

Similar to other SLAM algorithms, we assume that no correlations exist between measurements, and therefore constraint residuals and their costs are uncorrelated also. In reality there are sources of error that can produce correlated residuals. A slightly misaligned lidar scanner, for example, will produce biases in the headings indicated by all constraints. While biased measurements will affect pose graph convergence, in practice, there are often much larger sources or errors, such as odometry as discussed in Section 2.4.2.

2.5.4.6. Joint Probability Distribution

Pose graph optimization is generally performed using a maximum likelihood estimator on the joint probability distribution. Thus, the joint probability distribution, $\mathbf{x}$, for all T robot poses and constraints is formed first. We start by stacking the pose estimates to form the state vector:

$$\mathbf{x} = [\mathbf{r}_0, \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3 \cdots \mathbf{r}_T]^T \tag{2.48}$$

Similarly, the constraints are grouped into two vectors, odometry $\mathbf{u} = \left[\mathbf{u}_1^0, \mathbf{u}_2^1, \mathbf{u}_3^2 \cdots \mathbf{u}_T^{T-1}\right]^T$ and loop closures $\mathbf{c} = \left[\mathbf{c}_k^j \cdots\right]^T$. For conciseness, we redefine the constraint residual as a function of the entire state vector. For a single loop closure constraint:

$$e(\mathbf{x}, \mathbf{c}_k^j) \underset{\text{def}}{=} e(\mathbf{r}_j, \mathbf{r}_k, \mathbf{c}_k^j) = \mathbf{c}_k^j - z(\mathbf{r}_j, \mathbf{r}_k) \tag{2.49}$$

Since odometry and loop closure constraints have independent Gaussian distributions, the joint probability distribution for all poses, $\mathbf{x}$, can be described as a product of factors:

$$p(\mathbf{x}|\mathbf{u}, \mathbf{c}) = \underbrace{\prod_{t=1}^{T} \mathcal{N}\left(e(\mathbf{x}, \mathbf{u}_t^{t-1}), \mathbf{\Sigma}_u\right)}_{\text{Odometry Constraints}} \times \underbrace{\prod_{\mathbf{c}_k^j \in \mathbf{C}} \mathcal{N}\left(e(\mathbf{x}, \mathbf{c}_k^j), \mathbf{\Sigma}_{j,k}\right)}_{\text{Loop Closure Constraints}} \tag{2.50}$$

Recognizing that odometry and loop closures are both measurements of spatial constraints that use the same residual function, we combine them here into a single vector to simplify presentation:

$$\mathbf{z} = \begin{bmatrix} \mathbf{u} \\ \mathbf{c} \end{bmatrix} = [\mathbf{z}_0, \mathbf{z}_1, \mathbf{z}_2 \cdots \mathbf{z}_i \cdots \mathbf{z}_N]^T \tag{2.51}$$

Where each constraint, $\mathbf{z}_i$, is one of the N combined odometry and loop closure constraints in the pose graph. The joint probability distribution from Equation 2.50 reduces to:

$$p(\mathbf{x}|\mathbf{z}) \simeq p(\mathbf{x}|\mathbf{u}, \mathbf{c}) \tag{2.52}$$

$$= \prod_i \mathcal{N}(e(\mathbf{x}, \mathbf{z}_i), \mathbf{\Sigma}_i) \tag{2.53}$$

2.5.4.7. Nonlinear Least-Squares Form

This section derives the nonlinear *least-squares form* of the pose graph optimization problem. Pose graph optimization, or inference on factor graphs [100], is generally performed using maximum likelihood estimation (MLE). In the case of 2-D SLAM, with a pose graph of $\mathbb{SE}(2)$ constraints, this amounts to finding the configuration of poses that minimizes the sum of all individual constraint costs $\|\mathbf{e}_{j,k}\|^2_{\mathbf{\Sigma}_{j,k}}$ from Equation 2.47.

We first initialize the state vector, $\mathbf{x}_0$, from Equation 2.48 with the best estimate available. This is often achieved by fixing the first pose to zero and calculating the entire trajectory of poses by accumulating odometry measurements as in Equation 2.25. For a multi-robot pose graph, $\mathbf{x}_0$ can be initialized using a spanning tree that incorporates loop closure constraints between robots.

By definition [238], the maximum likelihood estimate of $\mathbf{x}$ is:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}}\, p(\mathbf{x}|\mathbf{z}) \tag{2.54}$$

Where $p(\mathbf{x}|\mathbf{z})$ is the joint probability distribution from Equation 2.53. Equation 2.54 is converted to its *negative log likelihood* form by substituting and taking the logarithm:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}}\ \log\left[p(\mathbf{x}|\mathbf{z})\right] \tag{2.55}$$

$$= \underset{\mathbf{x}}{\operatorname{argmax}}\ \log\left[\prod_i \mathcal{N}(e(\mathbf{x}, \mathbf{z}_i), \mathbf{\Sigma}_i)\right] \tag{2.56}$$

$$= \underset{\mathbf{x}}{\operatorname{argmax}}\ \sum_i \log\left[\frac{1}{(2\pi)^{3/2}|\mathbf{\Sigma}_{j,k}|^{1/2}} \exp\left(-\frac{1}{2}\mathbf{e}^T_{j,k}\mathbf{\Sigma}^{-1}_{j,k}\mathbf{e}_{j,k}\right)\right] \tag{2.57}$$

$$\simeq \underset{\mathbf{x}}{\operatorname{argmin}}\ \sum_i \left[\frac{1}{2}\mathbf{e}^T_{j,k}\mathbf{\Sigma}^{-1}_{j,k}\mathbf{e}_{j,k}\right] \tag{2.58}$$

$$\simeq \underset{\mathbf{x}}{\operatorname{argmin}}\ \sum_i \|e(\mathbf{x}, \mathbf{z}_i)\|^2_{\mathbf{\Sigma}_i} \tag{2.59}$$

In this frequently-used simplification, the logarithm cancels the exponential in the normal function to give the least-squares form in Equation 2.59 [100]. We now stack the N constraint residuals and their covariances:

$$\mathbf{e}(\mathbf{x}) = \underset{3N\times 1}{\begin{bmatrix} e\left(\mathbf{x}, \mathbf{z}_0\right) \\ e\left(\mathbf{x}, \mathbf{z}_1\right) \\ \vdots \end{bmatrix}} \qquad \mathbf{\Sigma} = \underset{3N\times 3N}{\begin{bmatrix} \mathbf{\Sigma}_0 & 0 & \cdots \\ 0 & \mathbf{\Sigma}_1 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}} \tag{2.60}$$

This allows us to express the least-squares form in Equation 2.59 even more compactly:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \left\|\mathbf{e}(\mathbf{x})\right\|_{\mathbf{\Sigma}}^2 \tag{2.61}$$

Thus the maximum likelihood of a pose graph is estimated by finding the value for $\mathbf{x}^*$ that minimizes $\left\|\mathbf{e}(\mathbf{x})\right\|_{\mathbf{\Sigma}}^2$, or the sum of all constraint costs simultaneously.

2.5.4.8. Gauss-Newton Algorithm

The Gauss-Newton algorithm is frequently used to find the maximum likelihood estimate of a pose graph using the least-squares form in Equation 2.61. Gauss-Newton is an iterative optimization algorithm that aims to reduce the total residual cost $\left\|\mathbf{e}(\mathbf{x}+\delta)\right\|_{\mathbf{\Sigma}}^2$ by solving for an incremental step δ.

The residual error $\mathbf{e}\left(\mathbf{x}+\delta\right)$ is nonlinear and so it is approximated by a first-order Taylor series expansion around $\mathbf{x}$:

$$\mathbf{e}\left(\mathbf{x}+\delta\right) \simeq \mathbf{e}\left(\mathbf{x}\right) + \mathbf{J}\delta \tag{2.62}$$

Here $\mathbf{J}$ is the Jacobians of the constraint residuals stacked, i.e.:

$$\mathbf{J} = \underset{3N\times 3T}{\begin{bmatrix} \mathbf{J}_0 \\ \mathbf{J}_1 \\ \vdots \end{bmatrix}} \tag{2.63}$$

Where each Jacobian, $\mathbf{J}_i$, is a sparse block matrix with non-zero blocks for the poses $\mathbf{r}_j$ and $\mathbf{r}_k$ that it connects:

$$\mathbf{J}_i = \left.\frac{\delta e(\mathbf{x}, \mathbf{z_i})}{\delta \mathbf{x}}\right|_{\mathbf{x}} = \begin{bmatrix} 0 & \cdots & 0 & \underbrace{\frac{\delta e(\mathbf{x}, \mathbf{z_i})}{\delta \mathbf{r}_j}}_{\text{Pose } \mathbf{r}_j} & 0 & \cdots & 0 & \underbrace{\frac{\delta e(\mathbf{x}, \mathbf{z_i})}{\delta \mathbf{r}_k}}_{\text{Pose } \mathbf{r}_k} & 0 & \cdots & 0 \end{bmatrix} \tag{2.64}$$

These Jacobians are derived in [243]. The least-squares form in Equation 2.61 becomes:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \left\| \mathbf{e}\left(\mathbf{x}\right) + \mathbf{J}\delta \right\|_{\mathbf{\Sigma}}^2 \tag{2.65}$$

Combining this with Equation 2.47, the problem is linearized and expanded:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \ \left(\mathbf{e}\left(\mathbf{x}\right) + \mathbf{J}\delta\right)^T \mathbf{\Sigma}^{-1} \left(\mathbf{e}\left(\mathbf{x}\right) + \mathbf{J}\delta\right) \tag{2.66}$$

$$= \underset{\mathbf{x}}{\operatorname{argmin}} \ \delta^T \underbrace{\mathbf{J}^T \mathbf{\Sigma}^{-1} \mathbf{J}}_{\mathbf{H}} \delta + 2 \underbrace{\mathbf{e}\left(\mathbf{x}\right)^T \mathbf{\Sigma}^{-1} \mathbf{J}}_{\mathbf{b}} \delta + \underbrace{\mathbf{e}\left(\mathbf{x}\right)^T \mathbf{\Sigma}^{-1} \mathbf{e}\left(\mathbf{x}\right)}_{\mathbf{k}} \tag{2.67}$$

Equation 2.67 is quadratic in the term δ, with the *quadratic form,* $\delta^T \mathbf{H} \delta + 2\mathbf{b}\delta + \mathbf{k}$. Thus the incremental step, δ, is found by solving the *normal equation* for this quadratic, or the linear system:

$$\mathbf{H}\delta = -\mathbf{b} \tag{2.68}$$

Where:

$$\mathbf{H} = \mathbf{J}^T \mathbf{\Sigma}^{-1} \mathbf{J} \tag{2.69}$$

$$\mathbf{b} = \mathbf{e}\left(\mathbf{x}\right)^T \mathbf{\Sigma}^{-1} \mathbf{J} \tag{2.70}$$

The incremental step δ in Equation 2.68 can be solved for by many different techniques, for example Cholesky decomposition [100]. In each iteration, the maximum likelihood estimate is calculated by applying the incremental step δ:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \delta \tag{2.71}$$

If the previous estimate, $\mathbf{x}_k$, was near a minima in the cost function, the new cost, $\|\mathbf{e}(\mathbf{x}_{k+1})\|_{\mathbf{\Sigma}}^2$, is likely to be less. The algorithm is iterated until convergence or a maximum iteration count is reached.

The matrix $\mathbf{H} = \mathbf{J}^T \mathbf{\Sigma}^{-1} \mathbf{J}$ is an adjacency matrix, where the non-zero elements indicate the pairs of poses in $\mathbf{x}$ that are coupled by constraints. Similar to information matrices (Section 2.5.3.5), $\mathbf{H}$ is sparse, positive-semidefinite and symmetric. Each constraint contributes four non-zero blocks to $\mathbf{H}$: two blocks on the diagonal for each pose, and two off-diagonal blocks between the poses [243]. The matrix $\mathbf{H}$ is additive; as additional constraints connect poses they accumulate information, or their covariances decrease.

The linear system in Equation 2.68 is often solved directly using sparse Cholesky decomposition. The structure of the problem is well understood, and highly efficient direct solvers

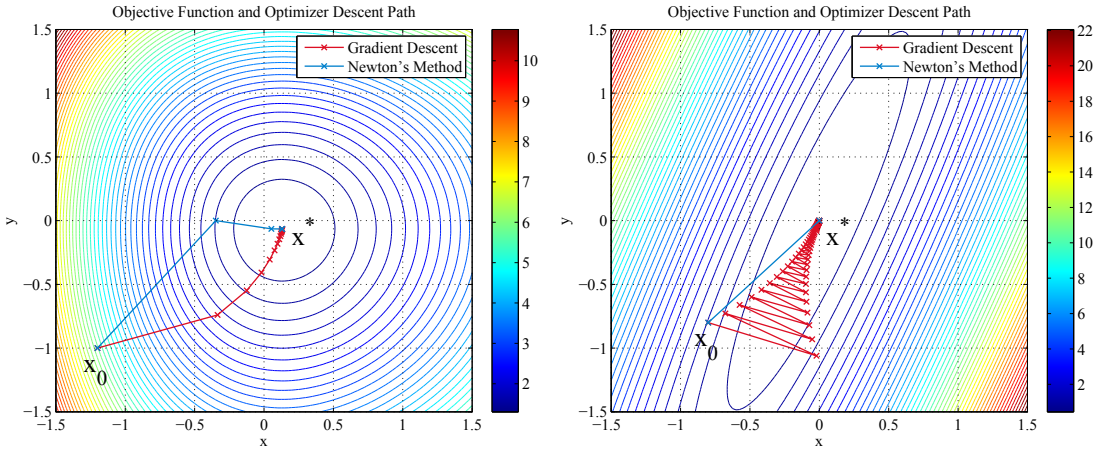


Figure 2.20. Convergence properties: Convergence for different optimization algorithms varies depending on the shape of the cost function. Here gradient descent and Newton's method are compared for a 2-D cost function $\|\mathbf{x}\|^2_{\mathbf{\Sigma}}$. Both circular (left) and ellipsoidal covariances (right) are common in pose graphs. Gradient descent converges slowly, and while Newton's method converges quicker, it may get stuck at saddle points [234]. Figure courtesy Niko Sünderhauf [244].

are available. Modern software libraries, such as CSparse, use memory storage techniques that maximize cache hits in modern processors [237, 238]. The structure of $\mathbf{H}$ and its symbolic decomposition are constant during each optimization, and as such are reused for efficiency.

2.5.4.9. Review of Optimization Algorithms

Several iterative and direct optimization algorithms are described here. Each takes a different approach to calculating the incremental step δ, used in the state update Equation 2.71. The $\|\mathbf{e}(\mathbf{x})\|^2_{\mathbf{\Sigma}}$ cost function in pose graph SLAM creates a large non-convex and nonlinear solution space. As described in Section 2.2.2.1, and similar to the other SLAM algorithms in Section 2.5.3, nonlinearity in the $\mathbb{SE}(2)$ constraints hampers convergence. Additionally, we typically cannot satisfy all constraints simultaneously, which results in local minima and no guarantee of convergence to the global optimum. Each algorithm has different complexity and convergence properties that generally relies on the initial state, $\mathbf{x}_0$, starting within some basin of convergence near global optimum.

Gradient descent: there are many iterative first-order algorithms that attempt to minimize cost by taking λ-sized steps in the negative gradient direction:

$$\delta = -\lambda \mathbf{b} \tag{2.72}$$

Large steps can jump over valleys in the cost function, resulting in worse solutions. Thus the step size, λ, is chosen using a line search algorithm. Gradient descent can be slow to

converge, Figure 2.20 shows two examples where the shape of the cost function and step size greatly affect gradient descent's rate of convergence.

Gauss-Newton: the Gauss-Newton algorithm was derived in the previous section. It is a modification to Newton's method that does not require second derivatives to be calculated. The Hessian $\mathbf{H} = \frac{\delta^2 \mathbf{e}(\mathbf{x})}{\delta \mathbf{x}^2}$ in Gauss-Newton is approximated by $\mathbf{H} = \mathbf{J}^T \mathbf{\Sigma}^{-1} \mathbf{J}$ given in the quadratic form, $\delta^T \mathbf{H} \delta + 2\mathbf{b}\delta + \mathbf{k}$, in Equation 2.67. The incremental step, δ, is found by solving the linear system:

$$\mathbf{H}\delta = -\mathbf{b} \tag{2.73}$$

Gauss-Newton is a direct, second-order method that converges quickly for simple least-squares problems. It typically converges faster than simple gradient descent methods, however can converge to saddle points [234].

Levenberg-Marquardt (LM): combines the guaranteed convergence of gradient descent and the speed of Gauss-Newton. The incremental step, δ, is found by solving the linear system augmented by the tunable damping factor $\lambda \mathbf{I}$:

$$(\mathbf{H} + \lambda \mathbf{I})\, \delta = -\mathbf{b} \tag{2.74}$$

As the damping factor is increased, $\lambda \to \infty$, the algorithm tends toward gradient descent, while reducing it, $\lambda \to 0$, reverts to Gauss-Newton. Step control is a complex topic [234]. Typically if the cost after an iteration, $\|\mathbf{e}(\mathbf{x}_k + \delta)\|^2_{\mathbf{\Sigma}}$, is lower, λ is decreased, whereas if the cost is higher, the update is rejected and λ is increased. As λ increases, LM switches to gradient descent to ensure convergence [111]. LM is widely used in pose graph optimizations due to its convergence properties [237, 238], and is used extensively throughout this research work.

Incremental smoothing and mapping (iSAM): is a direct second-order method. Kaess et al. use a QR matrix factorization to exploit the Hessian's sparsity, without explicitly forming the normal equations [235, 236]. Even though QR factorization is considerably slower than Cholesky [7], iSAM can reuse the factorization to provide an efficient incremental algorithm. Periodic re-linearization of the Jacobians and re-factorization of the Hessian are performed in batch updates, producing similar solutions to Gauss-Newton.

Hierarchical pose graphs (HOG-Man): Grisetti et al. estimate the structure of the pose graph by sparsifying it into a hierarchy varying from a very coarse pose graph to the original [245]. HOG-Man optimizes each level of the hierarchy separately, performing small Cholesky factorizations with Gauss-Newton on each level, enabling efficient incremental local updates.

Stochastic gradient descent (SGD): Olson et al. demonstrated an iterative gradient descent algorithm that converges quickly from poor initial estimates [233, 109]. By randomly optimizing individual constraints, SGD "pulls" the pose graph around randomly, and can "hop" over local minima. The step size, λ, is slowly decreased so the estimate is encouraged to settle into the global minimum. Olson reduces the step size using $\lambda = \frac{\lambda_0}{t}$.

2.5.5. Submapping Techniques

Submapping techniques exploit the structure of the SLAM problem to reduce computational cost and increase scalability. They divide the SLAM map into multiple partial *submaps,* which results in an approximation the full SLAM solution. The SLAM algorithms described in Section 2.5.3 provide insights into the structure of the SLAM problem. The success of constant-time sparsification in the Sparse EIF [199], and periodic thinning in the Thin Junction Tree Filter [204] suggest that parts of the full SLAM joint probability distribution can be simplified without introducing large errors.

A similar structural insight is used in submapping approaches: spatially close map features tend to be conditionally independent given the robot's pose, resulting in local distributions that are complex. These locally complex relationships tend to be less significant when considering the global structure of a SLAM map, however, making them well-suited for grouping into submaps. Thus the full SLAM joint probability distribution can be approximated by many independent local submaps that have poses joined by probabilistic constraints.

In a submapping system, a SLAM *front-end* constructs many small, local, submaps in approximately constant time. Each submap defines its own local coordinate frame, an important distinction compared to other hierarchical approaches described in Section 2.5.3. The submaps are embedded in a global frame and they are joined by a pose graph of $\mathbb{SE}(2)$ constraints that are optimized by a graph-based SLAM *back-end* to construct global maps. This modern graph-based submapping approach is described in more detail in Section 4.3.1. The approach enforces independence between submaps, an approximation that provides great efficiency in practice with minimal loss in accuracy (Chapter 7).

Prior to the development of efficient graph-based SLAM techniques (Section 2.5.4), approaches based on submapping were necessary to perform on-line SLAM at large scales. Bosse et al. first described their submap-based *Atlas Framework* in 2003 [246, 247, 248]. Their system enabled a robot to drive several loops in a 250×200 meter environment.

Despite the success of Atlas-like submapping systems, in the decade since, many SLAM researchers have continued pushing towards the "gold-standard" of full SLAM solutions, describing techniques that globally optimize every pose, lidar scan and, in some cases,

every lidar return [136]. For multi-robot SLAM systems that need to operate on-line and at large scales, however, these solutions are too computationally intensive, and it is impractical to transmit billions of lidar measurements over RF communications.

My research work builds on the core submapping concepts described in the Atlas Framework, adding several contributions that enable distributed multi-robot SLAM at large scales (Chapter 4). The remainder of this section describes several key submapping approaches described in the literature.

Atlas Framework (2003): Bosse et al. described Atlas as a "hybrid metrical/topological approach" [246, 247, 248]. In the Atlas submapping approach, submaps of bounded size and complexity are created using an EKF. Submaps are linked by $\mathbb{SE}(2)$ constraints, and the back-end optimizes their poses using nonlinear least squares. New loop closure constraints are validated using cycle verification to ensure they are in agreement. Atlas has been demonstrated with both lidar and sonar sensors, and with both landmark-based maps and scan matching. To minimize the number of submaps, Atlas allows robots to re-enter and update previously created submaps. When extending Atlas to multi-robot SLAM, however, this capability restricts it to operating in highly-centralized manner only. Section 4.3.3 explains why Atlas is not well-suited to decentralized multi-robot SLAM.

Hierarchical SLAM (2005): Similar to Atlas, Estrada et al. describe a two level map representation that includes independent local maps and a graph-based global constraint map [249]. They also use an EKF to build local maps, noting that the accumulated linearization errors were bounded. Estrada et al. allow previous submaps to be re-entered, and the submap EKFs to be updated; an operation likely to introduce inconsistencies when crossing coordinate frames. Unlike Atlas, Hierarchical SLAM optimizes graph cycles. Their approach has been demonstrated on-line in a 100×100 meter environment with large loop closures. In 2009 Estrada described a centralized MR-SLAM adaptation [250], demonstrating it on the Victoria Park dataset divided into 6 pieces.

Tectonic Smoothing and Mapping (2007): Ni and Dellaert describe their batch smoothing and mapping (SAM) approach in [251, 252]. They divide landmark-based maps into submaps, and perform small pose graph optimizations in each local frame. Importantly, they describe how grouping landmarks into rigid submaps, like tectonic plates, also groups the relative constraint linearizations. A global optimization of the $\mathbb{SE}(2)$ poses of these submaps avoids need to re-linearize measurements within submaps. In [251] they describe "out-of-core" optimization, where submaps can be paged to disk so that the entire SLAM problem does not need to fit in memory.

HMT-SLAM (2008): Blanco et al. describe a submapping approach called "hybrid metric-topological" SLAM [253]. Their approach builds local submaps using RBPF's, and while it was not designed to output global maps, their submaps can be arranged

into a global frame using their topological relationships and optimized with pose graph techniques. Their approach is unique since it can create multimodal constraints between submaps, however they are not used in pose graph optimization. Submap sizes are selected based on the sensor's properties such as noise and field of view; an approach also used in my research work (Section 4.4.1.3).

2.6. Problem Review

This chapter has introduced the SLAM problem and described its key aspects, such as data associations and loop closures, in detail. Table 2.1 summarizes these into various problem areas, along with references to relevant publications that address them. The problems with unique aspects that are directly addressed by my research contributions are indicated by chapter.

Area	Problem	Relevant Publications	Thesis
Data Association	Constraints/correspondences: baked-in or reversible	[144, 142, 254, 255, 218, 223]	Ch 4, 6
	Perceptual aliasing: corridors, repeating structures	[123, 183, 256, 184, 257]	Ch 5, 6
	Non-static environment, moving object tracking	[218, 258, 259]	Ch 4
	Multimodal constraints, multi-hypothesis	[104, 260, 117]	Ch 5, 6
Loop Closing	Large loops: detecting, minimizing false positives	[261, 256, 262, 263]	
	Visual place recognition	[176, 170, 171, 217, 172, 173]	
Localization	Motion models: odometry noise and failure modes	[151, 135, 47, 158, 150]	
	Coordinate frame: global, robo-centric, relative, submaps	[181, 220, 264, 247, 251]	Ch 4
	Global: GPS denied or intermittent	[79, 164]	Ch 4
	Global: aerial image priors	[265, 266, 267]	Ch 4
	Relocalization: kidnapped robot, place recognition	[268, 27, 269, 261, 166, 174]	
Mapping	Measurement models: sensors, accuracy and noise	[135, 130, 136, 128, 133, 134]	
	Structure: metric, topological, submapping	[140, 145, 149, 135]	Ch 4
	Map parameterization: landmarks, gridmap, point clouds	[137, 129, 140, 135, 149]	Ch 5
Bayesian Estimation	Estimator choice: EKF filter, RBPF, pose graph	[9, 6, 209, 212, 135, 193]	
	Nonlinearities in motion models and constraints	[194, 110, 195, 135, 181]	
	Consistency, optimistic covariance estimates	[155, 156, 157]	
	Computational complexity, memory usage	[180, 185, 188, 193, 216]	
Pose Graphs	Nonlinear least-squares optimization, sparse factorization	[224, 228, 233, 235, 245, 237]	
	Optimization convergence, convexity, approximations	[233, 270, 271, 272, 273]	
	Graph simplification, bounding complexity	[204, 207, 264, 274, 275, 263]	
	Robust constraints, constraint cycles	[250, 263, 244, 276, 117, 277]	Ch 5, 6
	Multimodal optimizations, non-Gaussian distributions	[57, 117, 278]	Ch 6
Submapping	Spatial constraint estimation	[279, 280, 281, 282]	Ch 5
	Map data fusion, simplification, deduplication	[247, 56, 53]	Ch 5
	Consistent global estimation, conditional independence	[283, 284, 188]	
	Optimal submap size, overlap	[253]	Ch 4
Large Scale	Large areas, large trajectories	[207, 229, 172, 266, 66, 53]	Ch 4, 5
	Highly cyclic graphs, high average node degree	[236, 53, 239]	Ch 4
	Long time scales, life-long mapping across seasons	[217, 275, 285, 172, 263]	
Multiple Robots	Relative pose initialization	[203, 27, 191, 50, 286, 31]	Ch 4
	Real-world robust deployable systems	[287, 53, 64, 58, 61]	Ch 4
	Wireless comms noise, drop-outs, bandwidth, latency	[288, 289, 290, 291, 53]	Ch 4
	Decentralized data fusion, state estimation	[292, 288, 293, 85, 294]	Ch 4
	Distributed processing, map building	[292, 295, 296, 297, 298]	Ch 4, 5
	Continuous loop closures, dense graphs	[236, 53, 299]	Ch 4
	Robust constraints, pose graph convergence	[300, 301, 302]	Ch 6

Table 2.1. Key SLAM problems and research contributions by chapter: Key problem areas are listed here along with relevant publications that have been referenced in this thesis. My research work was deployed into real-world environments as part of a working MR-SLAM system, therefore it had to solve *most* of these problems. Problems with unique aspects directly addressed by this research work are indicated by chapter.

3 Review: Large-Scale Multi-Robot SLAM

Multi-robot SLAM introduces a unique set of problems in addition to the single-robot problems described in Chapter 2. The complex distributed nature of MR-SLAM ensures that its solutions are not trivial extensions to existing single-robot SLAM algorithms. After providing some definitions in Section 3.1, Section 3.2 outlines the key MR-SLAM problems, introduces several high-level system architectures and reviews the MR-SLAM literature. Section 3.3 summarizes SLAM systems that have been demonstrated at large scales, and concludes by considering how MR-SLAM algorithms scale.

3.1. Definitions

Throughout this thesis I use the term "SLAM *algorithm*" to describe the math used for probabilistic estimation. Here I introduce the term "SLAM *architecture*" to indicate a high-level design abstraction and "SLAM *system*" to describe a particular instance of a SLAM architecture and algorithm.

For any on-line SLAM architecture it is convenient to divide the system's functionality into *front-end* and *back-end* roles. This terminology was used by Karlsson et al. in [303], and it has become widely used in the literature, e.g. [243, 237, 221, 53]:

- **Front-end:** runs on each robot where it acquires raw sensor data, often performing preprocessing on it. The front-end may simply filter sensor and odometry data to remove noise, compress redundant data, or to perform data association, e.g. [27]. More complex front-ends use a local SLAM algorithm to fuse sensor data, transmitting only compressed data with probabilistic estimates, e.g. [53, 58].

- **Back-end:** executes a MR-SLAM algorithm that maintains a probabilistic model of the robots and environment. The back-end fuses global maps and produces real-time localization data for other components of a MRS, such as global path planners. The back-end role is typically more computationally intensive than the front-end.

For brevity, the acronym *UGV* will be used to describe the unmanned ground robots that participate in a MRS, while the term ground control station, or *GCS*, will be used to describe the computers that interact with the MRS. Collectively, UGV and GCS computers are referred to here as the *participants* in a MRS.

In the context of MR-SLAM in the following discussions, the term *large scale* is used to describe demonstrations where the sum of accumulated distances driven by all UGVs is greater than 5 km. The accumulated distance is not the only factor that determine the complexity of the MR-SLAM problem, however, and the definition of "large scale" is considered more thoroughly in Section 3.3.

3.2. Multi-Robot SLAM

3.2.1. Problem Statement

While the multi-robot SLAM problem is a natural extension to the single-robot SLAM problem introduced in Chapter 2, its distributed nature makes it considerably more complex when deployed at the core of a coordinated MRS. The MR-SLAM problem requires more than a collection of single-robot SLAM algorithms: each robot needs to be localized in a single global coordinate frame, while large volumes of sensor data needs to be fused into maps in real-time. MR-SLAM problems frequently described in the literature include:

- **Consistent coordinate frames:** locally smooth coordinate frames for sharing global or relative spatial information.
- **Global map merging:** taking sensor data acquired by multiple UGVs and fusing into a single global or windowed map.
- **Relative pose initialization:** identifying $\mathbb{SE}(2)$ transforms between UGV coordinate frames, particularly after rendezvous when starting from different locations.
- **Decentralized data fusion:** enabling UGVs to perform MR-SLAM independently.
- **Global convergence:** ensuring optimization is robust to outliers and local minima.

For real-world deployments, the spatially and computationally distributed nature of the MR-SLAM problem introduces many unique aspects that require careful architectural considerations, including:

- **Wireless communications:** working with networks that are potentially intermittent and lossy, often with high-latencies and low-bandwidth.
- **Data reduction:** parameterizing and compressing large quantities of sensor data to enable efficient wireless communication and data fusion.
- **Distributed computation:** dividing SLAM algorithms and their computation over all UGV computers.
- **Distributed map building:** producing global or windowed maps on all UGV and GCS computers as needed.
- **Continuous loop closures:** handling dense and cyclical spatial constraints caused by multiple UGVs exploring together.
- **Map divergence:** preventing the distributed copies of global maps from becoming too different.
- **Double counting:** preventing overconfidence by ensuring measurements are only fused into the global map once.
- **Mutual observations:** exploiting information from direct observations between UGV during rendezvous.
- **Heterogeneous teams:** integrating sensor data from different UGV designs.
- **Race conditions:** avoiding timing-related problems due to latencies and messages arriving out of order.

Many of these problems are complex, ensuring that MR-SLAM is not a trivial upgrade over single-robot SLAM.

3.2.2. Architectures

For a stand-alone UGV, such as a household vacuum cleaner, the SLAM front-end and back-end roles and their associated data flows all occur within a single on-board CPU, leaving little scope for architectural variation. When multiple UGVs are considered, however, multiple additional distributed CPUs are available and architectural decisions must be made that determine where the back-end roles are performed and what data should flow over the wireless communications.

The MR-SLAM architectures described in the literature fall somewhere between *centralized* and *decentralized,* with many variations in-between. Similarly, the computation of a MR-SLAM algorithm can be performed on a centralized server, or *distributed* over all computers in a MRS. To avoid ambiguities that exists in the early literature, four broad classes of architecture are defined below, their relationship is illustrated in Figure 3.1.

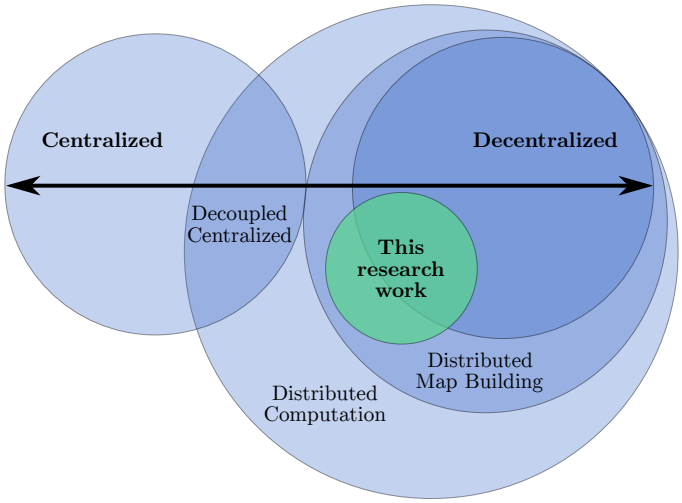


Figure 3.1. MR-SLAM architectures: Venn diagram showing different classes of MR-SLAM architectures spanning from centralized to decentralized. Not all architectures that distribute MR-SLAM computation are able to build distributed copies of the global map. For reference, the hybrid-decentralized architecture designed for my research provides fully distributed computation and map building. It is described in Chapter 4, and shown here in green.

- **Centralized architectures:** have a single computer at the GCS with a promoted role. Each UGV runs a minimal front-end that transmits all sensor data to this centralized computer, which runs a single MR-SLAM back-end algorithm that fuses the data from all UGVs. While this minimizes computation and storage requirements on each UGV, it requires considerable wireless communications bandwidth for each UGV. UGVs have limited spatial abilities and must be micro-managed by the GCS, so that even a minor communications issue will paralyze the UGV team.

- **Decoupled centralized architectures (DCA):** have a single centralized computer at the GCS, however each UGV maintains its own local coordinate frame that is locally smooth and independent of the back-end [29, 53, 58]. Each UGV front-end defines a *decoupled* coordinate system that uses a local filter to produce pose estimates from lidar scan matching and odometry. The UGV front-ends do not perform loop closures, ensuring that pose estimates in the local frame vary smoothly over time. Moore et al. describe the merits of this approach in [304], while Mei et al. survey several other relative SLAM coordinate systems approaches in [220]. We adopt Olson et al.'s terminology from [53], when referring to this approach as DCA. In DCA, a single MR-SLAM back-end runs on the centralized computer where it fuses preprocessed data from all UGVs. DCA requires part of the front-end SLAM computation to be distributed to each UGV, however it allows the front-end's computational requirements to be bounded by limiting the size or complexity of the filter (e.g. a SWF from Section 2.5.3.8). A significant benefit of DCA is that individual UGVs can continue to perform limited operations independent of the GCS

during brief communications outages. Without loop closures, however, this ability is limited by the rate at which odometry errors accumulate.

- **Distributed architectures:** describe algorithms or computation that is divided into parts and executed across a network of multiple computers. When a MR-SLAM architecture is distributed, the various parts of the MR-SLAM algorithm are executed on different computers in the MRS (both UGV and GCS computers). The simplest distributed architectures shift as much of the front-end SLAM computation onto the UGVs as possible (e.g. DCA), or use the UGVs to search for local loop closures. More complex architectures run distributed instances of the MR-SLAM back-end on each computer, allowing computation to be more evenly distributed. While this enables individual UGVs to produce their own copy of the map, or search for global loop closures independent of the GCS, it requires some, or all, of the MR-SLAM data to be distributed across the MRS computers. This data distribution may be expensive for wireless networks without broadcast capabilities.

- **Decentralized architectures:** describe systems that have no centralized controller and where all participant computers are equal. Thus in a decentralized MR-SLAM architecture, all UGVs perform MR-SLAM independently by executing local instances of the back-end. Each UGV maintains a copy of the global map and state estimates, however they may be incomplete, and typically no single UGV is aware of the complete network topology [288, 293]. While this independence provides robustness to many types of MRS failures, it comes at a cost of redundant computation and risk of global map divergence due to incomplete state information and overconfidence due to double-counting. Implementations with point-to-point communications require each UGV to retransmit SLAM data to every other UGV, e.g. [288]. Decentralized architectures may enable UGVs to plan paths and perform complex tasks without continuous communications with the GCS, however these paths may be globally suboptimal due to incomplete information.

Table 3.1 summarizes the advantages and disadvantages of these architectures, such as tolerance to failures and comparing capabilities such distributed map building and UGV navigation independent to the GCS.

3.2.3. Previous Work

This section reviews previous work, with a focus on large-scale MR-SLAM systems that are comparable to my research. This review is split into three parts: the decade before the 2010 MAGIC challenge, the MAGIC challenge, and the few years since. The section concludes with a timeline and summary of comparable systems in Table 3.2.

	Centralized	**Decoupled Centralized**	**Decentralized**	**Hybrid-Decentralized (This work)**
Distributed global map building	✖	✖	✖	✓
Distributed back-end computation	✖	Partly	✓	✓
Tolerant to failure at GCS	✖	✖	✓	✓
Decentralized local UGV navigation	✖	✓	✓	✓
Decentralized global UGV navigation	✖	✖	✖	✓
Avoids map divergence	✓	✓	✖	✓
Avoids double counting	✓	✓	Some	✓
Tolerant to comms interference	✖	Limited	✓	✓
Comms relative bandwidth	Very High	Low	High	Low
Comms est. KB/s for n UGVs	$100n$	n	$n(n-1)$	n
Front-end: filter or local SLAM	✖	✓	Optional	✓
Back-end: MR-SLAM algorithm	GCS only	GCS only	UGV & GCS	UGV & GCS
UGV processor/memory requirement	Low/Low	Med/Low	High/High	Med/High
GCS processor/memory requirement	V. High/High	High/High	High/High	High/High
References	[305, 31]	[27, 53, 58]	[288, 290, 296]	[66, 64]

Table 3.1. MR-SLAM architecture comparison: Capabilities and performance estimates for various levels of centralization. Relative estimates for communications, processor and memory requirements are given for a lidar-based UGV configuration that uses point-to-point communications. The hybrid-decentralized architecture is shown here for reference, it is described in detail in Chapter 4.

3.2.3.1. 1999-2009

While MR-SLAM was first described in the early 1990s [306], it was not until the late 1990s that it was first demonstrated. Howard and Kitchen described their Cooperative Localization and Mapping (CLAM) system in 1999 [227], which performed MR-SLAM using two robots. Vision-based front-ends transmitted range and bearing data to a centralized server. A single back-end used nonlinear least squares to optimize the trajectory to best fit the observations. This was an early description of a graph-based SLAM algorithm (Section 2.5.4.1).

Graph-based approaches were considered too expensive for the next several years, and instead the filter-based approaches that were popular for single-robot SLAM were shoehorned into MR-SLAM systems. Filter-based approaches, their limitations and scalability issues are reviewed in Section 2.5.3. Filters like the EKF, for example, are not well-suited for distributing over multiple computers in a MR-SLAM system, and typically require a highly-centralized architectures. Submapping approaches with periodic map-merging have been demonstrated [189], however for large areas and large teams of UGVs these approaches are still limited by either the quadratic nature of the filter, or inconsistencies resulting from "baked-in" nonlinearities [192].

In 2003 Makarenko and Durrant-Whyte demonstrated Decentralized Data Fusion (DDF) producing an occupancy gridmap for two UGVs [295, 293]. Originally designed for networks of sensors, DDF is a Bayesian data fusion technique; a comprehensive review is given by Liggens et al. in [292]. DDF has been demonstrated with graphical models (Section 2.5.2) in [296]. Sukkarieh et al. describe the application of DDF in the ANSER Project in [288], where four MAVs perform MR-SLAM using monocular cameras and landmark-based maps. The architecture they describe is fully decentralized, using information filters (described in Section 2.5.3.5) to share and fuse estimates. To avoid double-counting measurements they use channel filters to track information that has already been shared. To avoid duplicating information they use the information filter's convenient property that information can be removed by subtraction. In recent flight tests the team has demonstrated impressive results, including on-line MR-SLAM with two MAVs [85, 307].

One of the earliest mentions of distributed and decentralized graph-based MR-SLAM is by Konolige et al. in 2003 [308]. The UGVs in their MRS broadcast their lidar scans so that each UGV could build a local copy of the map using scan matching [224]. They demonstrated five UGVs operating on-line, however noted a quadratic increase in computation time. This work was a precursor to their distributed "Centibots" MR-SLAM system [28, 27, 49], designed for the DARPA SDR program in 2003 (described in Section 1.3.4). In their MR-SLAM architecture the front-end on each UGV transmitted odometry and lidar scans every 50 cm of motion. They described groups of UGVs moving in cliques, where one UGV was promoted to the MR-SLAM back-end role. The promoted UGV performed the expensive pose graph optimization and transmitted updates back to the remaining UGVs where distributed, local, copies of the map were generated. This impressive MR-SLAM system was demonstrated in a 45×25 m^2 indoor environment with three UGVs [27]. It also demonstrated robust relative pose initialization between pairs of UGVs using particle filters.

Howard et al. also demonstrated a MR-SLAM system in the 2003 DARPA SDR program [29, 51]. They described an architecture with separate local and global coordinate frames, similar to DCA (Section 3.2.2). Each UGV's front-end performed lidar scan matching with a SWF, transmitting lidar scans and decoupled pose estimates to the GCS. A centralized MR-SLAM back-end aggregated lidar scans from 3-10 m traverses into small local maps, or submaps. The back-end assembled these submaps into a pose graph which was optimized with MLE. Global occupancy-grids were produced at the centralized back-end. Howard et al. demonstrated four UGVs mapping in the SDR program. They tolerated noisy communications, with the exploration algorithm actively filling holes in the global map with other UGVs. They demonstrated relative pose initialization using mutual visual observations between pairs of UGVs.

Several key papers on single-robot pose-graph SLAM were published in 2006, including Thrun and Montemerlo's Graph SLAM [228], Olson et al.'s iterative Stochastic Gradient Descent [233], and Dellaert and Kaess' Square Root SAM [7]. In the years since, several graph-based MR-SLAM algorithms have been described. Many of these were demonstrated off-line with either prerecorded datasets or in simulations— likely due to the high cost of MR-SLAM research (Section 1.3.1). While many of these works describe algorithms that have been demonstrated on-line, they were often evaluated in centralized architectures, therefore avoiding many significant MR-SLAM problems by assuming perfect communications with zero-latency, no losses and infinite bandwidth.

Andersson and Nygårds described a landmark-based MR-SLAM algorithm that used SAM [305, 286]. Their method allowed MR-SLAM without initial knowledge of relative poses and handled map merges after a rendezvous. They demonstrated a centralized algorithm with two UGVs in a 12×6 meter environment. In [309], Chang et al. describe a MR-SLAM algorithm that generated topological/metric maps. Their highly-centralized submapping approach generated gridmaps after an off-line pose graph optimization. Estrada et al. described hierarchical MR-SLAM and demonstrated it off-line on the Victoria Park dataset divided into six parts [249, 250]. Pfingsthorn et al. implemented a manifold-based approach using a pose graph [310]. While they demonstrated their MR-SLAM system with the USARsim simulator (Section 1.4.1), they did not demonstrate loop closures in real-world datasets.

3.2.3.2. 2010 MAGIC Challenge

While five teams competed at the MAGIC challenge (refer to Section 1.3.4.2), only three of the teams demonstrated MR-SLAM systems that they later described in publications: Team Michigan (TM) [53, 54], University of Pennsylvania (Penn) [58, 311] and Team WAMbot [66, 64]. The latter is my research work that is described throughout this thesis.

Lee et al. demonstrated Penn's nine-robot MRS at the MAGIC challenge; their MR-SLAM system is described in [58, 311]. Their architecture was centralized however it used DCA to enable limited UGV operations independent of the GCS. They distributed all front-end SLAM computation to the UGVs, where they used Rao-Blackwellized particle filters (Section 2.5.3.7) to perform local SLAM and build 2-D occupancy gridmaps. The front-ends used a sliding window to remove old data and limit the accumulation of drift/errors in these gridmaps. To minimize communications bandwidth, only deltas to these gridmap cells were transmitted to the GCS. Their communications did not provide acknowledgements or delivery guarantees, however, and losses could cause parts of the global map to diverge from local maps. Their MR-SLAM back-end is described as pose-graph SLAM, however to the best of my knowledge, it was not implemented. Penn's global

maps, shown in [58] and [311], were generated from their datasets as an early part of my research work. More detail on their MRS architecture is given in Appendix A.

Olson et al.'s MR-SLAM system [53, 54], was designed to meet the same large-scale requirements as my research work, and is the most comparable. Their system was demonstrated on-line during the MAGIC challenge with 15 UGVs. Similar to Penn, Olson et al. used DCA to enable limited UGV operations independent of the GCS. The UGV front-ends performed localization in decoupled frames using a local SLAM algorithm with a "forgetful" 15 second sliding window. They used a servo-actuated "nodding" lidar scanner to acquire 3-D point clouds that were flattened into 2-D point clouds. Each 1.25 second scan was quantized and compressed into a "maplet," similar to a 2-D lidar scan. A single MR-SLAM back-end at the GCS assembled a centralized global map, and while lossy communications meant this map was often incomplete, the UGV's forgetful front-ends helped to ensure that map divergence did not paralyze the MRS. Their MR-SLAM back-end treated the maplets like 2-D lidar scans, performing scan matching for all 15 UGVs. Odometry and scan matching constraints were formed into a centralized pose graph that was optimized using Gauss-Newton (Section 2.5.4.7) and solved with Cholesky decomposition. They implemented a graph simplification algorithm that identified and removed redundant constraints [53], and a caching scheme to reduce redundant computation during gridmap building [56]. More detail on their MRS architecture is given in Appendix A.

3.2.3.3. 2010-2015

Since the MAGIC challenge, several authors have described MR-SLAM systems that are less-centralized. In [294, 298], Leung et al. describe a decentralized MR-SLAM approach that uses checkpoints to maintain synchronization of a distributed map. Their approach ensures each UGV's checkpoints are the same as the equivalent centralized estimate. While they demonstrate five UGVs ($N = 5$) building 2-D landmark-based maps, their approach is only evaluated in a 15×8 meter room with $M = 15$ landmarks. In highly-overlapped experiments, the computational complexity of their EKF-based approach scales with $\mathcal{O}\left(N^4 + N^3M\right)$, suggesting a complexity that is unlikely to scale to large areas or large teams of overlapping robots. Their 2-D landmark-based maps minimize communications requirements, however they lack the fidelity to describe most real-world environments.

Cunningham et al. described a distributed smoothing and mapping approach (DDF-SAM) that uses pose-graph SLAM to build small local graphs [297]. They condense local graphs into submaps and announce them to neighboring UGVs. Each UGV caches the condensed graphs and re-shares them, potentially sharing graphs between UGVs that are not in direct communications range. They demonstrate DDF-SAM on-line in a 100×100 m 2-D simulation and on Freiburg datasets [312], in both cases they use a single computer and avoid communication losses. In more recent work they demonstrate DDF-SAM using

RANSAC [313] to match local graphs and an updated design that prevents double-counting of information [314, 315]. Similar to Leung et al., their 2-D landmark-based maps lack the detail needed to describe real-world environments.

Kim et al. describe a centralized MR-SLAM system that uses iSAM (Section 2.5.4.1) to build pose graphs with lidar scan matching [31]. They solve the initialization problem by *anchoring* UGVs in separate coordinate frames and relating them with $\mathbb{SE}(2)$ constraints after a rendezvous. Their centralized architecture has been demonstrated processing datasets in real-time (but not on-line), and in one dataset they show a quadrotor MAV and UGV team building a gridmap in a 45×60 m area. The detected rendezvous using a camera and checkerboard, while they assumed perfect communications and unlimited bandwidth. Kim et al. describe improved initial convergence using anchors. For on-line systems, however, the ongoing cost of their coordinate transforms become expensive. Observations between UGVs' frames are coupled by the anchor variables, which adds additional non-zero elements in the Hessian matrix (Section 2.5.4.8). With this increased fill-in, Kim et al.'s anchoring approach would becomes expensive in MR-SLAM datasets with large numbers of loop closure constraints.

Several other systems with centralized architectures have been described over the last few years. While none have demonstrated MR-SLAM at scales larger than the MAGIC challenge, there are a few notable systems that address aspects of the communication problem. Guivant et al. describe the compression and transmission of 3-D SLAM data with a centralized server in [316]. Their well-designed system was initially designed for the MAGIC challenge, however it was not demonstrated in a MRS or at large scales. Dellaert et al. present simulations of a highly-centralized "map-server" design in [317]. They describe a submapping approach that they plan to use for MR-SLAM in a "cloud-based" centralized design. Other cloud-based designs have been described more recently by Riazuelo et al. [4] and Mohanarajah et al. [5].

Using an accumulated distance greater than 5 km as a threshold for large-scale MR-SLAM (Section 3.1), the only systems that have demonstrated *detailed* global maps at large scales were demonstrated at the MAGIC challenge [53, 54, 58, 311, 66, 64]. Similarly, the only system that has demonstrated fully-decentralized and distributed MR-SLAM at large scales was the DDF-based ANSER system [288, 85, 307], however it produced sparse, landmark-based, maps. These sparse maps, while large, lack the fidelity to represent real-world environments. Table 3.2 summarizes and compares the systems reviewed in this section.

Capabilities:	*Konolige et al. (SDR)*	*Howard et al. (SDR)*	*Andersson and Nygårds*	*Estrada et al.*	*Sukkarieh et al. (ANSER)*	*Olson et al. (TM, MAGIC)*	*Lee et al. (Penn, MAGIC)*	*Leung et al.*	*Cunningham et al. (DDF-SAM)*	***Reid et al. (This work)***
Multi-robot teams	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
On-line, real-time	✓	✓	✓	✖	✓	✓	✓	✓	✓	✓
Large scale	✖	✖	✖	✖	✓	✓	✓	✖	✖	✓
Detailed map representation	✓	✓	✖	✖	✖	✓	✓	✖	✖	✓
Avoids map divergence	✓	✓	✓	✓	✖	✓	✓	✓	✖	✓
Distributed map building	✓	✖	✖	✖	✖	✖	✖	✓	✖	✓
Distributed computation	✖	✖	✖	✖	✓	✖	✖	✓	✓	✓
Decentralized back-end	✖	✖	✖	✖	✓	✖	✖	✓	✓	✓
Details:										
Pose graph back-end	✓	✓	✖	✓	✖	✓	✓	✖	✓	✓
Occupancy gridmaps	✓	✓	✖	✖	✖	✓	✓	✖	✖	✓
Submapping	✖	✓	✖	✓	✖	✖	✖	✖	✓	✓
Year	2004	2004	2008	2009	2009	2010	2010	2011	2013	2015
Relevant publications	[27, 28]	[29, 51]	[305, 286]	[249, 250]	[85, 307]	[53, 54]	[60, 58]	[294, 298]	[297, 315]	[66, 64]

Table 3.2. MR-SLAM timeline: Significant MR-SLAM systems and their capabilities. Here, the term "large scale" describes a system that has been demonstrated with more than 5 km of continuous driving. My research work is shown for reference also.

3.3. Large-Scale SLAM

In Section 3.1 I used the accumulated distance driven by UGVs as a proxy for the scale and complexity of a SLAM problem. This provides an incomplete view, however, since the map extents, map resolution and trajectory loopiness are also important in determining the scalability of a SLAM solution. This section explores the state of the art in large-scale SLAM.

3.3.1. Large Areas and Trajectories

A comprehensive literature survey found many examples of "large scale" being used to describe SLAM algorithms. In 2000, a 9×3 meter area was considered large scale [186], while in the years since the term has been used to describe increasingly larger demonstrations. Table 3.3 summarizes the most significant publications over time. Over the last five years, many researchers have shifted their focus away from lidar to vision-based

SLAM systems such as Mei et al.'s RSLAM [220] and Cummins et al.'s FABMAP [172]. While they have demonstrated large trajectories, 142 and 1000 kilometers respectively, their SLAM outputs are sparse and lack the detailed information necessary for robot navigation.

For a MRS that operates in close proximity, SLAM must produce metrically accurate maps that are detailed enough for UGVs to plan paths through operational areas while navigating around each other. The extents of these maps are ultimately determined by the UGV range (e.g. speed, battery capacity and RF communication strength), while the resolution of these maps is determined by the footprint of the UGV and the smallest obstacles it could encounter (e.g. doorways or posts). Section 2.3.4 justifies the use of large gridmaps for MRS operations.

As summarized in Table 3.3, in 2003 the largest on-line MR-SLAM system that produced metrically-accurate gridmaps was demonstrated with four UGVs in a 45×25 meter indoor environment [51]. In 2010, a UGV and MAV pair were demonstrated in a 45×60 meter area [31], while at the MAGIC challenge up to 14 UGVs were demonstrated in a 210×150 meter mixed indoor/outdoor environment [53]. For the MAGIC challenge, where the ~50 cm UGVs had to navigate through doorways, the required mapping resolution was about 0.1 meters. Combined with the extents of the operational area (500×500 meters), this defined gridmaps of 5000×5000 cells; an order of magnitude larger than any other single or multi-robot SLAM system described in the literature.

It is interesting to note that global gridmaps output by these "large-scale" SLAM systems are unlikely to be useful beyond a certain size. For example, while my research work can produce 1,600×1,600 meter gridmaps at 0.1 meter resolution, it is questionable whether such maps would be useful in high-level tasks. Path planning in large gridmaps quickly becomes computationally expensive, and a city-scale MRS would instead use a hierarchical approach: global topological abstractions (i.e. road maps) for high-level planning, and smaller submaps for local fine-grained planning. This extension to my research is proposed in Chapter 8.

3.3.2. Large Teams of Robots

Section 3.2.1 outlined how the transition from single to multi-robot SLAM creates additional challenges. For state-of-the-art pose graph solutions, such as [53], the computational cost, memory storage and communications bandwidth requirements scale between $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$ for n UGVs that are constantly exploring. As discussed in [53], communications bandwidth is likely to be the factor limiting team size, particularly for lossy RF communications with large latencies.

Name	Test Environment	SLAM Algorithm	
Leonard and Feder, DSM 2000 [186]	9×3 m tank, $\frac{1}{100}$ scale AUV, sonar, 93 landmarks	Landmark-based, EKF divided into decoupled overlapping submaps	
Hahnel et al. 2003 [209]	28×28 m, Intel Research Lab, 491 m loopy trajectory	Lidar scan matching, FastSLAM/RBPF, 100 particles, real-time	
Bosse et al. Atlas, 2003 [246, 247, 248]	250×200 m, MIT Killian Court, 1,453 m trajectory, several loops	Lidar scan matching, 115 EKF submaps, global pose graph, cycle verification	
Konolige et al. Centibots, 2006 [28, 27]	45×25 m indoor, Fort AP Hill, DARPA SDR, 3 mapping robots	Lidar scan matching, pose graph optimization, multi-robot	★
Howard et al. 2006 [29, 50]	45×25 m indoor, Fort AP Hill, DARPA SDR, 4 mapping robots	Lidar scan matching, pose graph optimization, multi-robot	★
Frese et al. Treemap, 2006 [207]	350×350 m, 4× simulated 100 story buildings, 3,708,301 poses	Landmark-based, hierarchical graph, many independent floors, large cliques	
Thrun and Montemerlo, 2006 [228]	600×800 m, Stanford University, 100,000,000 landmarks	Landmark-based, pose graph, off-line EM-based optimization	
Estrada et al. Hierarchical SLAM, 2005 [249, 250]	100×100 m indoor/outdoor, 735 m trajectory, 3 loop closures	Landmark-based, MR-SLAM, EKF submaps, graph-based global opt.	
Kaess et al. 2007 iSAM [235, 236] 2012 iSAM2 [239]	250×300 m, Sydney University Victoria Park, trees only, 4 km trajectory, 6968 poses, 8 loops	Landmark-based SAM, pose graph, QR factorization, 7× real-time, library used in many other publications	
Ni et al. Tectonic SAM 2007 [251, 252]	4,856 poses, 18,000 landmarks, simulated block-world	Landmark, pose graph, batch SAM, submapping, out-of-core processing	
Piniés et al. CI-Graph 2008 [284, 318, 188]	300×300 m, 1,365 m trajectory, 9-loops, 34,173 trinocular images	Landmark-based, EKF visual SLAM, conditionally independent submaps	
Blanco et al. HMT-SLAM 2008 [253]	250×150 m, University of Malaga, 2 km trajectory, 5,000 poses	Topo-metric, submapping, RBPF, landmark or gridmap, optional global	
Konolige and Agrawal, FrameSLAM, 2008 [264]	10 km crusher dataset, two 5 km off-road loops, 42,000 images	Visual SLAM, pose graph, skeleton graph marginalization, real-time	
Dellaert et al. SPCG, 2010 [229]	35×25 km, central Beijing, 25,474 poses, 28,624 constraints	Hybrid pose graph, conjugate gradient descent, direct methods on subgraphs	
Kim et al. 2010 [31]	45×60 m, indoor loading dock, UGV and quadrotor MAV	Multi-robot, lidar scan matching, iSAM, multiple relative pose graphs	★
Cummins et al. FABMAP, 2011 [170, 172]	1000 km roadways urban and rural, 103,256 camera images	Topological, visual appearance-based PR, SURF features, bag-of-words	
Sibley et al. 2009 [319] Mei et al. 2011 [220]	142 km roadways, 850,000 images, multiple transport modalities	RSLAM continuous relative poses, no global frame, visual PR, real-time	
Kümmerle et al. g^{2}o 2011 [266, 238]	220×200 m, 890 m trajectory, indoor/outdoor, 445 3-D scans	Lidar scan matching, pose graph, gridmaps, MC localization with aerial.	
Olson et al. 2012 [53, 54, 55, 56, 57]	500×500 m, MAGIC 2010, indoor/outdoor urban, 14 robots	On-line, multi-robot pose graph, “maplets”, centralized MR-SLAM	★
Mapbuilder, Reid et al. (this research work) [66, 320, 64, 67]	500×500 m, MAGIC 2010, indoor/outdoor urban, 23 robot heterogeneous team	On-line, multi-robot pose graph, gridmap, submapping, distributed, hybrid-decentralized	

Table 3.3. Large-scale SLAM demonstrations: Various systems described in the literature as “large scale” from 2000 to 2015. Test environments have, on average, been getting larger and loopier. Several systems that are only demonstrated in simulations have been excluded along with sparse vision-based systems since 2011. Very few MR-SLAM systems have been demonstrated on-line in real-world environments. For reference, stars (★) highlight the systems that are most comparable with my research work.

"Very large scale" was used to describe the teams of UGVs deployed in the 2003 SDR program [28], see Section 1.3.4. While they demonstrated 100 UGVs localizing in a map, the initial MR-SLAM was only performed with four UGVs. The largest large-scale demonstrations of strongly coordinated MRS occurred at the MAGIC challenge: Olson et al. deployed 14 UGVs [53], while Lee et al. deployed 9 UGVs [311].

3.3.3. Large Average Node Degree

In state-of-the-art pose graph solutions (Section 2.5.4), the optimization cost grows approximately linearly when exploring new map areas. New poses are joined into the graph with a single odometry constraint and optionally one or more *local* loop closures, such that the average node degree remains approximately constant. When multiple UGVs explore together, many more local loop closures occur between the UGVs, creating local cliques of densely-constrained poses. Section 2.5.4.8 discusses how the Hessian matrix, $\mathbf{H} = \mathbf{J}^T\mathbf{P}^{-1}\mathbf{J}$ gains two new non-zero blocks for each additional constraint. While teams of overlapping UGVs contribute to increased Hessian fill-in, while exploring new map areas the matrix structure typically grows along the block-diagonal and is relatively easy to reorder and factorize.

Large loop closures occur in pose-graph SLAM when previously explored areas are revisited: these loop closures cause step increases in optimization cost because they add non-zero blocks further away from the Hessian's diagonal. Each large loop closure makes it increasingly difficult to find an optimum variable ordering and factorize the Hessian. Even with state-of-the-art matrix factorization approaches, this remains an NP-hard problem [239]. Teams of UGVs are more likely to create pose graphs with higher average node degrees and Hessian structures that are difficult to factorize, particularly when they are exploring in groups and frequently performing large loop closures. This creates an additional "large-scale" MR-SLAM problem. Techniques for simplifying pose graphs and reducing the average node degree have been described in [204, 207, 274], however only in the context of single-robot SLAM.

4 Hybrid-Decentralized and Distributed Multi-Robot SLAM

Several research contributions are described in this chapter that combine to form a novel hybrid-decentralized and distributed MR-SLAM system called Mapbuilder. The Mapbuilder MR-SLAM system enables decentralized teams of robots to build distributed global gridmaps and navigate large urban areas. My research is presented as a system architecture and design document for Mapbuilder: Section 4.1 starts with the high-level requirements and assumptions, while Section 4.2 outlines the high-level architecture, including design rationale. In Section 4.3 I describe the conceptual design, which includes a submapping technique and algorithms that enable decentralized and distributed MR-SLAM, while Section 4.4 presents the logical design describing each of the distributed software components and their interactions with other components. Section 4.5 verifies that the design addresses key requirements.

4.1. Introduction

The research work in this chapter was motivated by the MAGIC challenge (Section 1.3.4.2), and the need for a distributed MR-SLAM system that could enable teams of UGVs to be rapidly deployed in urban environments. Referring to Table 3.2, the previous state of the art did not include systems that could produce maps at large scales and with sufficient detail for UGV navigation, while being sufficiently decentralized to allow UGV operation independent of the GCS.

4.1.1. Research Contributions

Together, the research contributions in this chapter describe a novel MR-SLAM system architecture and design that distributes the MR-SLAM back-end functionality over all UGV and GCS participant's computers. Contributions include:

1. **Distributed MR-SLAM back-end:** allows computationally expensive MR-SLAM algorithms to be distributed over all UGV and participant computers. The back-end design adapts to the available computational resources (Section 4.4.2).

2. **Distributed global map building:** allows each participant to build its own local copy of the global occupancy gridmap. This distributed spatial awareness enables decentralized global MRS operations (Section 4.3.1).

3. **Hybrid-decentralized pose graphs:** enables cliques of participants to operate independent of the GCS. Submap pinning and priority-based filters ensure that local pose graph copies converge to the global solution (Section 4.3.9).

4. **Immutable submapping:** minimizes overall system complexity, in particular, making the distributed and decentralized parts of the design both simple and robust. Immutable, or "read only," submaps, retain the ability to re-evaluate the most uncertain data associations (loop closures), while minimizing communications bandwidth (Section 4.3.3).

5. **Human-in-the-loop MR-SLAM:** increases mapping accuracy and convergence by allowing a human operator to interact with the pose graph and submaps in real-time. This is essential in real-world environments where Gaussian assumptions do not always hold (Section 4.4.3).

Together these highly-coupled contributions form a functional MR-SLAM system— instead of describing them separately, this chapter is presented as a design document for the Mapbuilder system architecture. The Mapbuilder system architecture is presented here in a top-down manner: System Architecture (Section 4.2), Conceptual Design (Section 4.3), Logical Design (Section 4.4) and System Verification (Section 4.5). Results demonstrating these contributions are given in Chapter 7.

Before describing the Mapbuilder system, however, this section outlines various high-level dependencies, requirements and assumptions that helped to bound the scope of the research problems addressed in this work.

4.1.2. Dependencies

Section 1.2.3 suggested that any coordinated MRS that is deployed into unstructured environments should be built on top of a MR-SLAM system. The Mapbuilder MR-SLAM

system was initially designed as a core component of the WAMbot MRS, and their requirements are closely related. The WAMbot MRS is described in detail in Appendix A. Most notable is its distributed architecture, which required a distributed MR-SLAM architecture also. Figure A.1 on page 227 shows a deployment diagram indicating how Mapbuilder integrates with the WAMbot MRS.

In a distributed MRS deployment, multiple software components on each participant's computer will depend on Mapbuilder's localization and mapping outputs in real-time. Typical dependencies at the GCS include global path planners, task allocation, high-level autonomy, and Graphical User Interfaces (GUIs) for operator situational awareness and control. On each UGV dependencies include global and local path planners, local navigation, and local autonomy algorithms such as exploration.

4.1.3. Requirements

Mapbuilder's high-level functional requirements, including those from its various dependencies, can be summarized:

- Input all SLAM-related sensor data.
- Output occupancy gridmaps in real-time (global or windowed).
- Output localization data in real-time.
- Convert UGV local coordinates into global coordinates, e.g. OOI observations.
- Convert global coordinates into UGV local coordinates, e.g. operator commands.
- Edit underlying map structure, e.g. to correct for UGV sensing errors.

Similarly, Mapbuilder's low-level functional requirements are:

- Distributed map building: produce global or windowed maps on all participants.
- Decentralized map state: allows independent operation of small cliques of UGVs.
- RF communications: tolerant to intermittent and lossy communications.
- Heterogeneous: operates with different types of UGVs.
- Single-level environments: no overlapping terrain such as bridges.
- Height obstacles: detects and maps ramps, positive and negative hazards.
- Consistent coordinates: local and global frames for commands and observations.
- Flexible global localization: GPS-denied localization and aerial map georeferencing.
- No clocks: real-time computer clocks are not accurately synchronized.

Mapbuilder's performance requirements are:

- Real-time: delivers maps and updates to local processes at $> 1\,\text{Hz}$.
- Limited CPU usage: less than 25% of each UGV's processing power.
- Low bandwidth: limited to $1\,\text{KB/s}$ per UGV.

- Duration: 3.5 hours maximum, 3 km total odometry per UGV.
- Gridmap resolution: 10 cm, allowing a 50 cm UGV to navigate a 70 cm doorway.
- Large areas: 500×500 meter operational boundary.
- Large teams: 3-23 UGVs deployed into the same global map.
- Large overlaps: UGV sensor data densely overlapping in high-traffic areas.
- Robust to sensor noise: corrects loop closures of paths up to 250 m long.
- Unstructured environments: indoor and outdoor; arbitrarily cluttered or sparse.
- Robust to moving objects: tolerates OOIs and humans moving at up to 6 km/h.

These requirements could be considered quite generic for a MRS that is deployed into large-scale urban environments.

4.1.4. Assumptions

Mapbuilder is designed for "one-time" deployments into complex, unstructured, environments where unexpected problems or unmodeled errors are both likely to occur, and likely to result in mission failure. Therefore, while Mapbuilder is designed to handle >99% of MR-SLAM data automatically, we assume that a human operator at the GCS will correct occasional errors using a GUI. For UGV sensor configurations that primarily use 2-D lidar and wheel odometry (see Appendix A), non-Gaussian errors combined with ambiguities in lidar scan matching make these errors much more likely (discussed in Section 6.1.2.2).

To bound the scope of my research work, I made the following assumptions so I could focus on the core aspects of the distributed and decentralized MR-SLAM problem:

1. **Global UGV pose initialization:** UGVs are assumed to start in the same area, either outdoors or indoors with regular building geometry, e.g. roof structure and eaves. If GPS-based bootstrapping fails the operator can use the Mapbuilder GUI to add ground-truth constraints that align 2-D lidar data with aerial imagery.

2. **Aerial image priors:** matching 2-D lidar data to aerial imagery is a complex visual perceptual problem that is addressed in other works such as [265, 266, 267]. Here I assume that a trained operator can visually identify areas that are suitable to match (the Mapbuilder GUI automates extracting and aligning geometry).

3. **Global UGV relocalization:** UGVs are assumed to drive continuously through the environment and are never "kidnapped" or manually transported. Map-based relocalization has been addressed in many publications (see Section 2.4.4.2).

4. **Gaussian odometry errors:** odometry errors accumulated along small loops are assumed to not exceed a few meters. Small loops can be closed automatically by matching 2-D lidar data.

5. **Unmodeled and non-Gaussian odometry errors:** large unexpected errors are corrected by an operator. To help close large loops, the operator can add ground-truth constraints that align 2-D lidar data to either aerial imagery or other lidar data. Operators can delete submaps corrupted by non-Gaussian wheel slippage (Section 4.4.3).

6. **Long-term deployments:** only short-term deployments are assumed such that on-line operation is only required for 3.5 hours.

4.2. System Architecture

Mapbuilder is a hybrid-decentralized and distributed MR-SLAM system that is a solution to the MR-SLAM problems listed in Section 3.2.1, while satisfying the high-level requirements in Section 4.1.3, and given the assumptions made in Section 4.1.4.

4.2.1. High-Level Decisions and Rationale

When designing a MR-SLAM system, there are several high-level architectural decisions that are affect all aspects of the lower-level design and implementation. This section summarizes and explains these decisions.

2-D occupancy gridmaps: Many MRS software components that depend on MR-SLAM, particularly path planners and exploration algorithms, require detailed maps that can accurately describe open spaces, solid obstacles and unmapped areas of the environment. Mapbuilder is built around occupancy gridmaps because they are well-suited to unstructured environments (Section 2.3.4), and they explicitly describe unmapped areas. While the architecture described here could work with landmarks or line features, most high-level consumers would first convert them to gridmaps anyway. Occupancy gridmaps offer various implementation advantages that are explored in Chapter 5.

Graph-based: Mapbuilder's architecture leverages the graph-based SLAM techniques described in Section 2.5.4; they are central to its robustness, flexibility, and on-line performance. Pose graph data structures, formed by spatial constraints joining poses, provide a clean interface that is well-suited for incorporating SLAM data from multiple heterogeneous robots simultaneously. Unlike recursive Bayesian filters, poses and constraints can be added and removed from anywhere on the graph structure at any time. Large-scale loop closures and other alterations to the graph are possible[1], such that data associations and linearization errors are not "baked-in" into the estimate.

[1] Note that in practical implementations care must be taken to limit the size of errors in newly added constraints. Large step changes can cause numerical instabilities during optimization.

Distributed MR-SLAM back-end: The need to be robust to both intermittent and complete communication loss requires the Mapbuilder back-end to be distributed across all participants. For a MRS to handle intermittent communications, each UGV requires the ability to plan its own paths and operate in a decentralized manner without continuous low-level control from the GCS. This decentralized planning, in turn, requires each UGV to build and update its own local copy of the global gridmap (global maps are generally too large to be downloaded from a centralized server, particularly at the large scales considered here and after large structural changes). Distributed map building requires participants to run their own back-ends that maintains a local copy of the pose graph structure and submap data. While this introduces redundant computation, it also introduces the ability to distribute other back-end computation across participant computers.

Decentralized MR-SLAM back-end: With the same fully-functional MR-SLAM back-end running on each UGV, decentralized data fusion (DDF) can be performed on the distributed copies of the pose graph. This enables cliques of UGVs to continue functioning independent of the GCS during extended communications outages. With each back-end optimizing its own local pose graph, however, there is a risk that the various pose graphs could diverge catastrophically (a single dropped constraint message could produce very different global maps). A novel *hybrid* decentralized approach is described in Section 4.3.9 that is tolerant to lossy communications and minimizes the chance of divergence.

Wireless mesh communications: To enable the decentralized operation of cliques of UGVs, independent of the GCS or wireless infrastructure, Mapbuilder is designed to be deployed over a multi-hop wireless mesh network (described in Appendix A). To provide fault-tolerant message delivery, it uses a data distribution system (DDS) [321]. The DDS middleware provides a publisher-subscriber framework that enables robust real-time communications with variable quality-of-service (QoS) policies.

Submapping: Submaps, described in Section 2.5.5, are a key aspect of Mapbuilder's design that enables distributed map building and computation. They minimize MR-SLAM communications bandwidth by compressing dozens of lidar scans into a single 2-D occupancy gridmap. Submap gridmaps are a minimal representation that is very efficient for storage and transmission. Chapter 5 explores how they can be efficiently aligned and fused into global maps.

4.2.2. Software Components and Deployment

Mapbuilder's graph-based MR-SLAM algorithm is divided into front-end and back-end roles as defined in Section 3.1. The Mapbuilder system architecture defines three software components that are introduced briefly here:

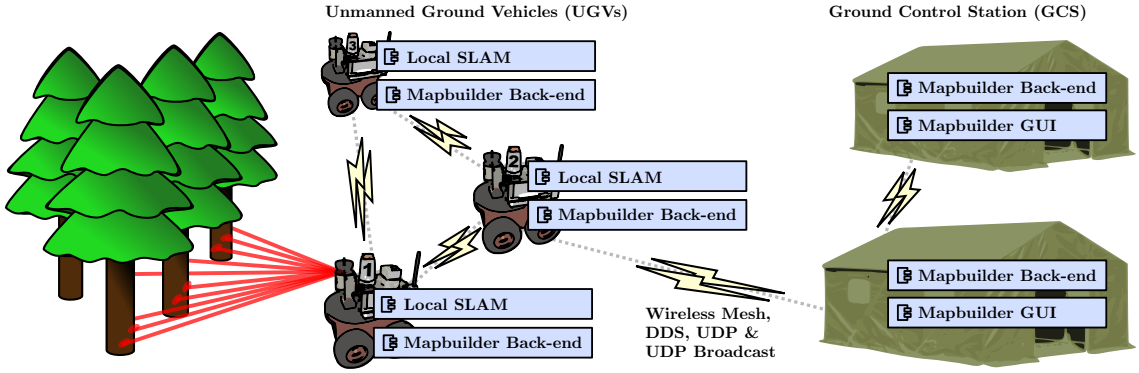


Figure 4.1. Mapbuilder architecture and software deployment diagram: Instances of the Local SLAM component execute on each UGV, processing sensor data. Instances of the Mapbuilder back-end execute on all participants (UGV and GCS computers) where it produces distributed copies of the global map. The Mapbuilder GUI software is used by operators at the GCS.

- **Local SLAM:** implements the SLAM front-end. Instances execute on each UGV where they use a single-robot SLAM algorithm to process sensor data and build submaps. Submap and constraint data is broadcast efficiently across the wireless mesh network (Section 4.4.1).
- **Mapbuilder Back-end:** implements the MR-SLAM back-end. Instances are executed on participants that require real-time mapping and localization data. Each instance stores a local copy of all submaps and constraints. Local pose graphs are periodically optimized and submaps fused to build global maps. The back-end searches for new loop closure constraints between submaps (Section 4.4.2).
- **Mapbuilder GUI:** implements a point-and-click user interface that allows operators to view the global map and interact with the pose graph. The GUI executes on one or more GCS computers where it enables operators to view and precisely manipulate individual submaps (Section 4.4.3).

Figure 4.1 shows a typical deployment diagram that illustrates the distributed nature of the architecture. The same Mapbuilder back-end software component executes on each of the UGV and GCS computer participants.

4.3. Conceptual Design

This section provides a conceptual overview of the Mapbuilder MR-SLAM system.

4.3.1. Graph-based SLAM with Submaps

To provide distributed copies of the global gridmap on every UGV and GCS participant, this design divides the global gridmap into *overlapping* submaps. Each submap defines a

rigid, locally Euclidean, occupancy gridmap and a $\mathbb{SE}(2)$ pose, $\mathbf{p}_a^{\mathcal{W}}$, that describes where it is located in the global frame, $\mathcal{W}$. This approach to multi-robot mapping can be explained with a table top analogy: if a submap is like a playing card with a gridmap printed on it, then each UGV lays down a sequence of overlapping cards as it explores (this is illustrated in Figure 4.2). Even a small team of UGVs exploring an area will build a large pile of overlapping cards, often dozens of layers high.

Submaps are key to Mapbuilder's ability to provide distributed map building; once each submap gridmap and its pose have been broadcast across the network, any back-end can fuse them to produce local copies of the global gridmap. This submap fusion searches for consensus in overlapping gridmaps to determine whether each of the global gridmap's cells are occupied, free or unknown. While fusing multiple distributed copies of the global gridmap introduces redundant computation, making this computation efficient is easier and less expensive than the alternative, which is to repeatedly broadcast or incrementally update global maps from a centralized server. Chapter 5 describes how submaps can be efficiently fused into global gridmaps in real-time, making distributed mapping with submaps possible.

Each submap's pose, $\mathbf{p}_a^{\mathcal{W}}$, is constrained by one or more $\mathbb{SE}(2)$ spatial constraints (refer to Section 2.5.4.4). Sets of constraints, $\mathbf{C} = \left\{ \mathbf{c}_b^a \cdots \mathbf{c}_a^{\mathcal{W}} \cdots \right\}$, join submaps to form large pose graphs that are mathematically equivalent to those described in Section 2.5.4. Two types of constraints are used: the first, $\mathbf{c}_b^a$, joins submaps with the residual error function $\mathbf{e}_b^a = \mathbf{c}_b^a - (\ominus \mathbf{p}_a^{\mathcal{W}} \oplus \mathbf{p}_b^{\mathcal{W}})$, given in Equation 2.45. The second, *ground truth* constraints, $\mathbf{c}_a^{\mathcal{W}}$, align submaps in the global frame with the residual error function $\mathbf{e}_a^{\mathcal{W}} = \mathbf{c}_a^{\mathcal{W}} - \mathbf{p}_a^{\mathcal{W}}$. These pose graphs are optimized with the maximum likelihood techniques, described in Section 2.5.4.7 and Section 4.4.2.3.

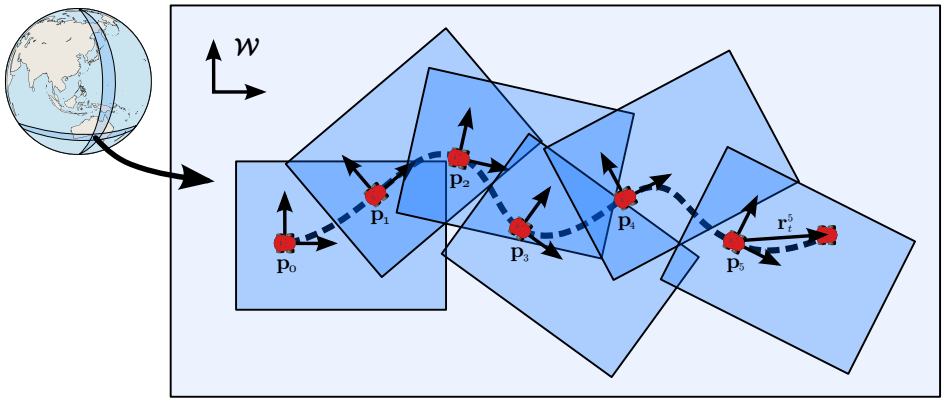


Figure 4.2. Graph-based SLAM with submaps: A single UGV (red) creates a sequence of overlapping submaps while driving (blue). Each submap contains a gridmap, where the pose $\mathbf{p}_a^{\mathcal{W}}$ indicates the gridmap's origin. The UGV in this example is currently in the right-most submap near $\mathbf{p}_5^{\mathcal{W}}$. The UGV's time-varying pose in the submap frame is $\mathbf{r}_t^5$, which in the global frame is $\mathbf{r}_t^{\mathcal{W}} = \mathbf{p}_5^{\mathcal{W}} \oplus \mathbf{r}_t^5$.

Extending the table-top card analogy, constraints are like mechanical springs that join pairs of submaps, causing them to slide and rotate into alignment (illustrated in Figure 4.3). Occasionally called *relaxation* in the literature [230], pose graph optimization, is analogous to releasing the submaps and their interconnecting network of springs and allowing them to slide and relax into the lowest energy state (where all the spring forces and torques are balanced). In a pose graph, this is equivalent to simultaneously minimizing the sum of all constraints' residual errors.

New constraints are created either by the SLAM front-end (odometry fused with lidar scan matching, Section 4.4.1), or when the MR-SLAM back-end identifies spatial alignments (correlation between gridmaps, Section 5.4). New ground-truth constraints are either created when an operator uses the GUI to drag submaps into alignment (Section 4.4.3), or when the operator matches submaps to aerial imagery (Section 4.4.3.2). Ground-truth constraints can also be created from GPS measurements, typically with large covariances.

Several authors have described graph-based SLAM approaches that use submapping (reviewed in Section 2.5.5), however, no submap-based MR-SLAM solutions have been described that can build detailed maps at large scales and in real-time. From the previous work summarized in Table 3.2, Howard et al.'s MR-SLAM system [29, 51] is the closest to the design described here, however their system was not distributed, relied on a centralized server and was only demonstrated at a small scale.

This submapping technique is highly effective for distributed MR-SLAM, particularly when closing large loops: every distributed copy of the global gridmap can be updated using minimal communications bandwidth by broadcasting only changes to the pose graph. To the best of my knowledge, building distributed global maps by fusing submap gridmaps has not previously been demonstrated. In early work, Makarenko and Durrant-Whyte

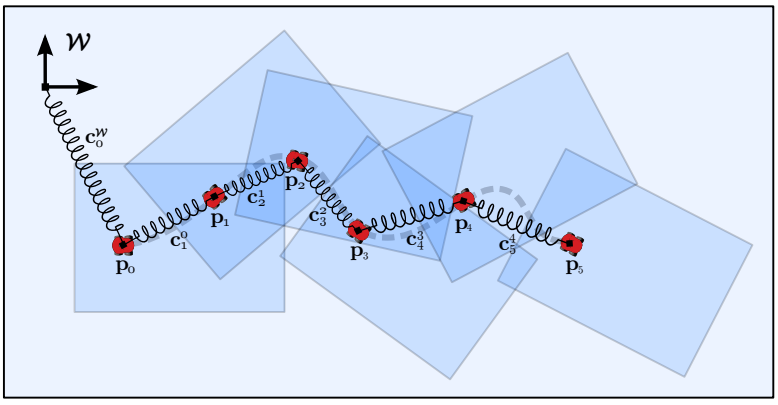


Figure 4.3. Submap constraints: Constraints are analogous to mechanical springs that connect pairs of submaps. During optimization they slide and rotate each submap's pose, $\mathbf{p}_a^{\mathcal{W}}$, until all forces and torques balance. This pose graph has a single ground-truth constraint, $\mathbf{c}_0^{\mathcal{W}}$, which locates the first submap in the world frame. The five subsequent submaps have their positions determined by the relative constraints $\mathbf{c}_b^a$.

built decentralized gridmaps with DDF in [293], however their simulations used external localization and did not perform SLAM.

4.3.2. Coordinate Frames

Pose graph optimization estimates each submap's pose in a global Euclidean coordinate frame, $\mathcal{W}$. This frame is necessarily Euclidean, since we use $\mathbb{SE}(2)$ pose compositions and optimization. The frame's origin can be arbitrarily selected for rapid deployment, however for precise georeferencing against orthorectified aerial imagery we uses a UTM grid [322]. Other globally-referenced coordinates, such as latitude and longitude [323], are converted to coordinates in the selected UTM grid.

Each submap defines a local coordinate frame that is anchored to its origin and located in the global frame at the submap's pose, $\mathbf{p}_a^{\mathcal{W}}$. Each locally-Euclidean submap coordinate frame is then used for all UGV-related operations. For example, the UGV pose, $\mathbf{r}_t^a$, is always broadcast relative to the current submap pose, $\mathbf{p}_a^{\mathcal{W}}$. Since each submap is created with the UGV at its origin, the UGV's submap pose is always initialized to zero, i.e. $\mathbf{r}_0^a = [0, 0, 0]^T$. The UGV's time-varying pose in the global frame is then given by the composition $\mathbf{r}_t^{\mathcal{W}} = \mathbf{p}_a^{\mathcal{W}} \oplus \mathbf{r}_t^a$. This configuration is also shown in Figure 4.2. The Local SLAM front-end on each UGV defines and maintains its own sequence of submaps, and hence it is responsible for defining these local submap coordinate frames.

4.3.3. Submap Life Cycle

A key design concept is that submaps become *immutable*, or "read-only," once they are closed. By preventing UGVs from re-entering old submaps, the aspects of the architecture that enable distributed MR-SLAM are greatly simplified, while robustness is increased. Submaps exist in one of two states:

- **Open:** submaps are currently being built by a single UGV; or
- **Closed:** submaps are immutable and cannot re-entered by any UGV.

Each UGV's Local SLAM front-end creates a continuous sequence of new submaps. The front-ends maintain a single "open" submap that the UGV currently occupies, connected to a sequence of older "closed" submaps that were previously completed. While submaps may physically overlap, each UGV only occupies a single submap at any moment in time.

Each Local SLAM front-end uses a single-robot SLAM algorithm to build submaps (various algorithms are reviewed in Section 2.5). The front-ends continuously perform local SLAM, resetting the map and pose uncertainty to zero when each new submap is started. The

submap closing criteria is described in Section 4.3.6. For reference, the WAMbot UGVs create new submaps every ~3 meters of travel.

While a closed, immutable, submap's gridmap contents cannot be changed, its pose may still be updated at any time. Thus once a closed submap has been broadcast across the network, any instance of the MR-SLAM back-end can update the submap's pose and fuse it into a global gridmap[2]. This allows the back-end to be designed with the assumption that a closed submap's gridmap contents will never be updated, enabling many efficiencies explored in Section 5.2. This distributed architecture's robustness and simplicity comes at the cost of increased computation and storage requirements. Chapter 5 shows how this additional cost can be offset with an efficient parallelized implementation.

To the best of my knowledge, the use of immutable submaps has not been described in the literature. Several submapping techniques were reviewed in Section 2.5.5. Early approaches, such as Bosse et al.'s Atlas [246], aimed to minimize pose graph complexity by limiting the number of submaps created. Bosse et al. described a graph traversal technique that allows UGVs to re-enter submaps that had been created earlier. Although their approach would work for centralized MR-SLAM, it does not extend cleanly to a decentralized MR-SLAM solution. Consider, for example, multiple UGVs driving in the same submap: decentralized updates would require an implementation of DDF for each submap, while lossy communications would introduce the potential for map divergence within a single submap and the potential for various race conditions; all complex engineering challenges. Mapbuilder uses immutable submaps to avoid these complex timing issues and race conditions.

4.3.4. Submap Uniqueness

Every submap created by the Local SLAM front-ends is assigned a universally unique identifier (UUID). These UUIDs are used in all MR-SLAM messages to uniquely identify submaps. This uniqueness is critical to the Mapbuilder design concept, since two submaps with same UUID would cause the distributed pose graphs to diverge catastrophically. While a convenient human-readable UUID scheme might be to concatenate each UGV's unique identifier with an incrementing counter, e.g. "`WAMbot_4_Submap_53`", collisions are still likely. Despite disk-based data persistence, battery failures and unexpected power cycling can still produce duplicate UUIDs.

To avoid problems, Mapbuilder uses a standard approach to generating UUIDs [324]. Each UUID is a 128-bit number carefully generated to avoid collisions. A random example, represented in hexadecimal, is "`5984ca82-b665-11e4-8ab6-6f95453c7342`". Using Version 1 UUIDs,

[2] If an object moves and the static-world assumption is broken, overlapping submaps may disagree. These disagreements are resolved when they are fused by the back-end (see Section 5.3).

uniqueness is guaranteed if each participant's computer has a unique IEEE 802 MAC address and the system clock increases monotonically.

4.3.5. Submap Gridmap Representation

The primary reasons to use submapping techniques are to minimize bandwidth, storage and redundant processing across the distributed MR-SLAM back-end. To achieve this, each UGV's front-end reduces the flood of incoming sensor data into a sequence of independent submaps joined by constraints. Submaps need to represent the "full" dataset as closely as possible, since the redundant sensor data is discarded.

In this architecture submaps represent the environment with gridmaps, however it is worth rationalizing this decision by considering other minimal representations. The simplest data reduction approach is to decimate lidar scans using a heuristic, such as storing one scan for every 50 cm of travel. This has been described in many pose-graph SLAM publications [28, 51, 31]. It provides a minimal representation that has been shown to work well in structured indoor environments. In outdoor environments, however, where objects are often more distant, the lidar scan measurements may become sparse to the point where this decimation discards useful information. This can be seen in the lidar scans in Figure 2.6 on page 40, where simple geometry such as straight walls become sequences of disconnected points.

Rather than decimating lidar scans, the Local SLAM front-end collects all of the scans acquired while driving through a submap. It aligns and fuses them into a single 2-D occupancy gridmap using the ray tracing technique described in Section 2.3.4.3. These gridmaps maintain an accurate representation of the environment, while requiring 10-30 times less data storage compared to the original set of scans. They are also several times more compact than the decimation approach described above. For reference, Figure 4.4 shows two typical submaps created by the WAMbot UGVs.

The gridmap representation rounds each lidar measurement to the nearest grid cell, naturally introducing small amounts of noise due to quantization. To prevent this noise from accumulating when each scan is ray traced into the gridmap, the front-end collects the lidar scans and aligns them with scan matching before ray tracing them into the gridmap as a batch. While gridmap cell sizes can be reduced to limit this quantization noise, the processing and storage costs increase quadratically, as discussed in Section 2.3.4.3. The largest gridmap cell size is selected that ensures quantization noise will not interfere with high-level functions, such as planning paths through narrow doorways.

4.3.6. Firewalling Pose Uncertainty

UGVs start in new submaps with zero pose uncertainty, and irrespective of the SLAM algorithm chosen, their pose uncertainty will always grow (Section 2.4.2.2). In most environments, horizontally mounted lidar sensors produce fairly consistent views (2-D slices) of the environment that change minimally over small UGV motions (e.g. 10 cm). These consistent views allows scan matching to align pairs of scans and reduce the growth in pose uncertainty due to odometry noise. The resulting rate of accumulation depends largely on the shape of the environment: a small rectangular room allows for precise scan matching and minimal uncertainty growth, while in long corridors or sparse outdoor environments scan matching is generally unable to reduce uncertainty.

The key to this submapping approach is to start a new submap whenever the UGV pose uncertainty becomes too high. By tracking the current angular pose uncertainty and average distance to obstacles in the environment, combined with the linear pose uncertainty, the amount of "blurring" in distant gridmap cells can be an estimated. If this estimate exceeds a fixed multiple of the gridmap cell size, a new submap is triggered.

This heuristic allows minimal distortions into the submap gridmap, while prevention the large distortions that could result from a badly aligned lidar scan. If a new submap is triggered, the current lidar scan is not fused. Rather the submap is closed, a new submap is opened and the lidar scan is fused into the new submap instead.

To prevent uncertainty from accumulating without bound, the pose uncertainty is reset to zero at the start of each submap. This effectively "firewalls" the uncertainty and prevents

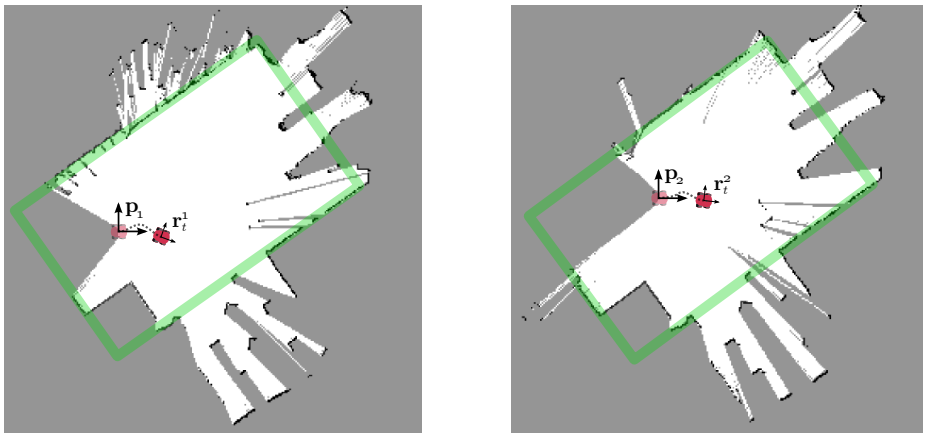


Figure 4.4. Local SLAM submaps: Two submaps created by WAMbot UGVs in the Old Ram Shed Challenge (ORSC). These 10 cm occupancy gridmaps represent the environment: Free space (white), obstacles (black) and unknown (gray). In both submaps the UGV (red) is overlaid twice: First at the submap origin, and second at the UGV's final pose where each submap is closed. The unmapped wedge behind each UGV occurs because of the 270° lidar FOV. These submaps can be compared to TM and Penn's in Figure 7.27 on page 212 where the same 20×12 meter area is highlighted in green.

it from distorting the contents of future submaps. For completeness, the accumulated uncertainty is not discarded, instead it is transferred into the constraint that is created to join the old and new submaps (the UGV's pose uncertainty covariance becomes the new constraint's covariance). Thus each UGV's local pose uncertainty is periodically passed into the pose graph where it is handled efficiently with maximum likelihood estimation.

The main insight in this design is that many lidar scans captured from a similar vantage point can be collected and fused so that the distortions due to pose uncertainty are small enough to be hidden in each gridmap's quantization noise. The front-end produces a sequence of submaps connected by constraints, where each submap is a spatially consistent representation of the environment and each constraint captures the UGV pose uncertainty that accumulated in the drive through the submap.

4.3.7. Loop Closures with Submaps

Each UGV front-end produces its own simple, acyclic, pose graph: long sequences of submaps connected by constraints and without loop closures. The best configuration of these disconnected pose graphs can be calculated with a simple spanning tree, without any SLAM or pose graph optimization.

The SLAM problem actually begins when the back-end identifies loop closures, between pairs of overlapping submaps with gridmaps that appear spatially similar. These loop closures are broadcast as new constraints that form cycles in the distributed pose graphs—it is the residual error in these newly-created cycles that requires optimization. Loop closure constraints include a $\mathbb{SE}(2)$ transform, a 3×3 covariance matrix and a weighting parameter that reflects the confidence in the match. The search for submap matches is performed by the back-end, described in detail in Section 4.4.2 and Section 5.4.

The nature of the loop closing problem changes considerably when multiple UGVs are operated in an overlapping area. To prevent errors from accumulating, and to keep the UGV team well-localized, the back-ends must search for local loop closures between the current "open" submaps and other overlapping submaps in real-time. These local loop closure constraints create many small cycles that greatly increase the pose graph's mean node degree. Ideally this increases the quality of pose estimates, at the cost of making pose graph optimization more computationally expensive. It is worth noting that the equivalent pose graph built from individual lidar scans would have orders of magnitude more nodes, thus Mapbuilder's submapping approach offsets much of the additional expense.

4.3.8. Robust Wireless Communications

Appendix A describes how the DDS communications middleware can be configured to provide various quality-of-service (QoS) policies over a wireless mesh network. In the

Mapbuilder DDS *publisher-subscriber model*, publishers buffer up to n messages, attempting to deliver them until their subscriber has confirmed receipt. The oldest messages are dropped if more than n messages become buffered. Each of the message types published by the front-end and back-end components are assigned various "best-effort" delivery policies that determine the buffer size n. Larger buffers are more expensive, since publishers require more memory during lossy communications, and consequently more bandwidth when synchronizing after an outage.

Publishers are separated into *local* and *global partitions*. Most MR-SLAM message types are broadcast to the global partition (all participants) only when required to avoid overloading the network (rules for broadcasts are given in Section 4.3.9.2). The local partition is used for high-rate inter-process communication within a participant (e.g. 10 Hz). Local partition messages passed between the front-end, back-end and other high-level MRS software components are efficiently transported using shared memory.

QoS policies and buffer sizes are tuned based on the total bandwidth available and the number of UGVs in the team. They are fixed configuration parameters in Mapbuilder, however it would be possible for the MRS to dynamically tune buffer sizes based on the current data throughput and bandwidth requirements of other MRS components.

Wireless mesh networks of moving UGVs often drop data packets and occasionally lose the physical link. Even with a robust communications framework, and "best-effort" message delivery policies of DDS, there are a few possible scenarios that the MR-SLAM system must still cope with:

a) **Out-of-order messages:** noise on the physical link may cause data to be retransmitted. Delayed messages may cause sets of messages to arrive out of order.

b) **Brief outages:** the message buffer size n is large enough to store all undelivered messages during the outage. After the outage each publisher is able to redeliver missing messages to each subscriber.

c) **Dropped messages:** the buffers become full during an outage and the oldest messages are dropped. For each message type, subscribers receive only the most recent n messages.

4.3.9. Hybrid-Decentralized Pose Graphs

Central to the Mapbuilder architecture is a set of *priority-based message filters* that control the flow of information between distributed copies of the pose graph. These filters are particularly important when UGVs move out of range and the wireless network splits into smaller cliques, and then when they rejoin and the network becomes fully connected again.

Each UGV and GCS participant executes an instance of the Mapbuilder back-end that builds and optimizes its own local copy of the pose graph and is capable of completely decentralized operation. In the ideal case, if wireless communications were instantaneous and flawless, these distributed copies of the pose graph would remain synchronized and decentralized pose-graph MR-SLAM would be simple. In reality, wireless communications are lossy and intermittent, and messages are frequently lost, delayed or reordered, causing the distributed pose graphs and their estimates to diverge. For large loop closures, in particular, a single missing constraint can produce very different estimates.

To synchronize distributed pose graphs it is *not* sufficient to make sure every constraint has been copied to every back-end instance. Large pose graph optimizations are susceptible to local minima, and unless the constraints are added in the same order, different pose estimates will result. Thus when wireless communications are lossy and intermittent, decentralized MR-SLAM requires additional logic to ensure global convergence.

This section first introduces a *naive* design to avoid pose graph divergence, and after pointing out its shortcomings, describes a *hybrid-decentralized* design that allows cliques of participants to continue operating independent of the GCS. Its implementation is described in Section 4.4.2.1.

4.3.9.1. Naive Decentralization

To keep the distributed copies of the pose graph synchronized, participants share their current pose graph estimates by broadcasting batches of `SubmapPose` messages. By design, these messages are relatively small and inexpensive to communicate since the QoS policy does not guarantee delivery (see Section 4.4.2.6). For reference, 30 `SubmapPose` messages transmitted as a batch are about the same size as an average submap gridmap.

To maintain synchronization in a naive decentralized system there are two rules that describe when a participant should broadcast batches of `SubmapPose` messages:

- **Rule 1: After optimization**: all participants broadcast `SubmapPose` messages for any pose estimates that are updated in the local pose graph.
- **Rule 2: After two cliques merge:** all participants broadcast `SubmapPose` messages for their entire pose graph estimate to synchronize all participants.

While these two rules create a functional decentralized system that can converge to a single global solution, it is not tolerant to the loss of constraints. Consider a MRS that splits into two cliques: if one or more constraints are dropped during an extended split, the submap pose estimates are likely to oscillate after rejoining (likely overloading the network with `SubmapPose` messages).

4.3.9.2. Hybrid-Decentralization with Priority-Based Filters

Each participant in a MRS is assigned a unique identifier that provides a *priority* that can be used to sort the participants deterministically. These identifiers, and thus priorities, are assigned so that GCS participants are ranked higher than UGV participants. During deployment, each participant tracks the priorities of the other participants in its local clique by querying the wireless mesh network.

If a participant recognizes that it has the highest priority in its clique, it automatically *promotes* itself. This typically happens when a small clique splits away from the main clique. Conversely, when two cliques merge only one participant has the highest priority; the other will automatically *demote* itself. As such, each participant's promotion and demotion is performed independent of the clique; there is no time-dependent two-way signaling and no centralized controller. Ideally, each clique has a single promoted participant, however if two participants self-promote, convergence is still guaranteed at the cost of some redundant communications.

Using this technique to determine self-promotion, the hybrid-decentralized approach refines the two rules used in the naive approach:

- **Rule 1: After optimization**: the promoted participant broadcasts `SubmapPose` messages for any pose estimates that are updated in the local pose graph.
- **Rule 2: After two cliques merge:** both promoted participants broadcast their pose graph estimates as a batch of `SubmapPose` messages.
- **Rule 3: After a clique splits in two:** the newly promoted participant broadcasts its entire pose graph estimate.
- **Rule 4: After receiving stale messages:** low-priority participants broadcast their non-stale copies of `SubmapPose` messages after a deterministic time delay.

Rule 3 and Rule 4 are added for robustness and are explained in Section 4.3.9.4. For revision tracking, each `SubmapPose` message is augmented with the priority and local timestamp of the participant that last updated the pose estimate (see Figure 4.7). When a participant accepts a `SubmapPose` message, it stores the priority and timestamp along with the pose estimate.

To ensure convergence, two *priority-based filters* determine whether `SubmapPose` messages are accepted and which submaps in the local pose graph are eligible for optimization. The filters are deterministic: information flows in a single direction so that the distributed pose estimates converge on a single, global, solution without oscillations or race conditions:

- **Filter 1:** is used by participants to decide which `SubmapPose` messages to accept: if the incoming priority is lower than the current priority, the message is ignored. If the incoming priority is higher, the message is accepted, while if the priorities match (originates from the same participant), the pose estimate with the most recent timestamp is kept. Using this filter, each participant's pose estimates will converge to the newest estimate from the highest priority participant.

- **Filter 2:** is used by participants to decide which submaps in the local pose graph to optimize. The participant's priority is compared against the local pose graph, and only submaps that have pose estimates with equal or lower source priorities are optimized. Any submaps with pose estimates generated by higher priority participants are "pinned" so they cannot move; and any constraints between these pinned submaps are omitted from the optimization.

Using these rules and filters, the pose estimates in the distributed copies of the pose graph tend to converge to the promoted participants' estimates over time. Lower priority participants accept most `SubmapPose` messages, resulting in the majority of the submaps in their local pose graphs becoming pinned. Since UGVs have lower priorities than the GCS, the only submaps they tend to optimize are the ones they create locally with their own front-end (until they are pinned by a higher priority participant).

Within each clique, the highest-priority participant will optimize the unpinned parts of the pose graph, broadcasting updates that the lower-priority participants accept. The lower-priority participants accept these updates and pin their submaps, avoiding the majority of the redundant pose graph optimization that would occur in the naive design. Pinning also minimizes the quantity of `SubmapPose` messages, helping to avoid situations where participants flood the wireless network.

4.3.9.3. Convergence Example

A brief example is given here to show how these rules and filters perform together. In a typical MRS deployment, the UGV team starts near the GCS so that the mesh network is fully-connected and every participant is part of a same clique. As the UGV team starts to explore, the main GCS participant has the highest priority and all of the UGV participants remain synchronized to its pose estimates. Figure 4.5 (a) illustrates this situation.

When a clique of UGVs drive out of communications range, however, the mesh network splits as shown in Figure 4.5 (b). The main clique (shown in red) splits to form the new clique (blue). The participant in the blue clique with the highest priority promotes itself and begins optimizing newly-created submaps in the blue shaded area (submaps in the red shaded area remain pinned).

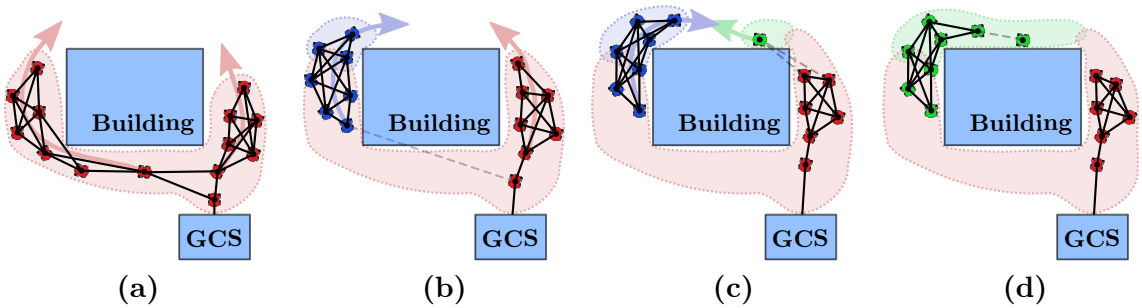


Figure 4.5. Decentralized pose graph case study: (a) A clique of 14 participants (red UGVs) exploring outside a building. Wireless network links between participants are indicated in black. Note that these figures are not pose graphs; the red shading indicates areas containing submaps. In (b) the mesh network splits into two cliques; the highest-priority participant in the new clique (blue UGVs) optimizes their newly created submaps (blue shaded area). In (c) a participant from the red clique loses its network connection and proceeds alone (green). Finally in (d) the green and blue cliques merge and the green participant is promoted. The design handles this worst-case situation robustly.

In Figure 4.5 (b) and (c) the participants in the red clique continue to explore and the main GCS participant continues to optimize the submaps in the red shaded area. In (c), however, a single UGV participant loses communications with the red clique and splits to form the green clique by itself (it is carrying pose updates for the pinned submaps in the red area).

A worst-case scenario is created in Figure 4.5 (d) when the UGV in the green clique merges into the disconnected blue clique and becomes the highest priority participant. At this moment, the promoted green UGV knows nothing about the existence of the submaps in the blue shaded areas in (c), while it is carrying stale updates to the pinned submaps in the red area. Following Rule 2, both the green UGV and the promoted blue UGV broadcast their submap estimates. The UGVs that were blue in (c) accept the updated pose estimates and at this moment a discontinuity may be created between the submap estimates in the red and blue areas. By design, the blue UGV that was promoted in (b) is still able to optimize the submaps in the blue area; it quickly repairs the discontinuity and continues to optimize these submaps until the missing constraints are delivered to the promoted green UGV. At this point in (d), the promoted green UGV is able to optimize all of the submaps in the larger green area. This scenario illustrates how the design can handle worst-case edge cases effectively.

4.3.9.4. Robustness

The DDS middleware may take seconds or minutes to communicate batches of `SubmapPose` messages to the entire mesh network. If the mesh network configuration and cliques of participants are changing frequently, partial sets of messages may occasionally be received by some participants. To avoid discontinuities from appearing in the distributed gridmaps,

these messages are sent in batches that are applied atomically (i.e. only if the entire batch is received).

DDS makes no guarantee of the order in which subscribers are serviced. When cliques merge the synchronization is deterministic, however, if cliques split and merge repeatedly it is possible for a promoted participant to have stale pose estimates. Rule 3 and Rule 4 (Section 4.3.9.2) are designed to provide robustness in this situation.

Rule 3 causes a newly promoted participant to broadcast its pose graph as a batch of `SubmapPose` messages. Each of the low-priority participants process them using Filter 1, ignoring any messages that are stale. According to Rule 4, the ignored messages are flagged by the low-priority participant, which then re-shares its local estimates (they have higher priorities and/or newer timestamps).

Since multiple participants can identify the same stale pose estimates according to Rule 4, broadcasts are delayed to avoid flooding the network. Each participant calculates its delay deterministically by considering its priority ranking within the local clique; the second highest priority participant waits one second before broadcasting, while the n-th priority participant waits $(n-1)$ seconds. These subsequent broadcasts are likely to clear many, or possibly all, of the other participants' flags, avoiding large batches of redundant `SubmapPose` messages from overloading the network.

Several edge cases can occur, particularly when participants are slow at detecting changes to their local clique[3]. When cliques merge, for example, more than one participant can be promoted at the same time. In this case, Filter 1 consider all messages regardless of the source, and only the highest priority, and newest update is stored. This approach ensures that each participant's pose graph estimate converges to the same global solution, even when batches of `SubmapPose` messages arrive in different orders. This design deliberately avoids two-way message passing to decide which participants to promote and demote, this prevents race conditions and avoids a form of the "Two Generals Paradox" [325].

4.3.9.5. Considerations

This hybrid-decentralized approach is designed to allow extended periods of decentralized activity. While it can handle difficult scenarios like the one shown in Figure 4.5 (c) and (d), they should generally be avoided when out of communications range of the GCS. Large-scale loop closures should ideally occur either while supervised by an operator, which in this scenario requires one or more UGVs to be repositioned to act as a wireless relays so that the network remains fully connected. Alternatively, unsupervised large-scale loop closures could be confirmed by a more precise data association technique (e.g. visual fiducials as in Section 2.4.4.2).

[3]The mesh network used in WAMbot (Appendix A) took up to a minute to detect topology changes.

4.4. Logical Design

The logical design presented here is derived from the conceptual design in the previous section. Each UGV and GCS participant performs the same roles, which leads to a logical design with only three software components. The software components' functional requirements are:

Local SLAM: (Section 4.4.1)

- **Input:** sensor data (lidar, odometry, IMU, GPS).
- **Behavior:** performs local SLAM, creates sequence of submaps.
- **Output:** broadcasts submap data, constraints, real-time UGV pose estimates.

Mapbuilder Back-end: (Section 4.4.2)

- **Input:** submap data from all UGVs, submap constraints, ground-truth constraints.
- **Behavior:** optimize pose graphs, fuse submap data, search for constraints.
- **Output:** global or windowed maps, submap pose estimates, submap constraints.

Mapbuilder GUI: (Section 4.4.3)

- **Input:** all MR-SLAM messages, GUI events, e.g. keystrokes and mouse clicks.
- **Behavior:** display global maps, interpret operator commands.
- **Output:** messages that alter graph structure, e.g. ground-truth constraints.

Instances of the Mapbuilder back-end component execute on all participants, while the Local SLAM front-end executes only on UGVs, and the Mapbuilder GUI executes only on GCS participants. Figure 4.1 on page 103 shows a typical deployment diagram. A minimal view of Mapbuilder's logical design, considering only a single UGV and GCS participant, is shown in Figure 4.6. This diagram summarizes the inputs and outputs of the Local SLAM front-end and Mapbuilder back-end software components.

The various MR-SLAM message types used by Mapbuilder are shown in a class diagram in Figure 4.7. All messages are derived from the `Submap` message class, which ensures that all messages include the source participant's priority and timestamp, along with the submap UUID.

This research work uses several open source libraries. Common to all three software components is the Eigen library[4], which is used extensively for matrix math. $\mathbb{SE}(2)$ pose compositions (Section 2.2) are implemented using templates provided by the g^{2}o graph optimization library[5] [238]. While the selection of software libraries is typically an implementation detail, these are listed here since their capabilities are leveraged heavily in the design.

[4] http://eigen.tuxfamily.org
[5] http://github.com/RainerKuemmerle/g2o

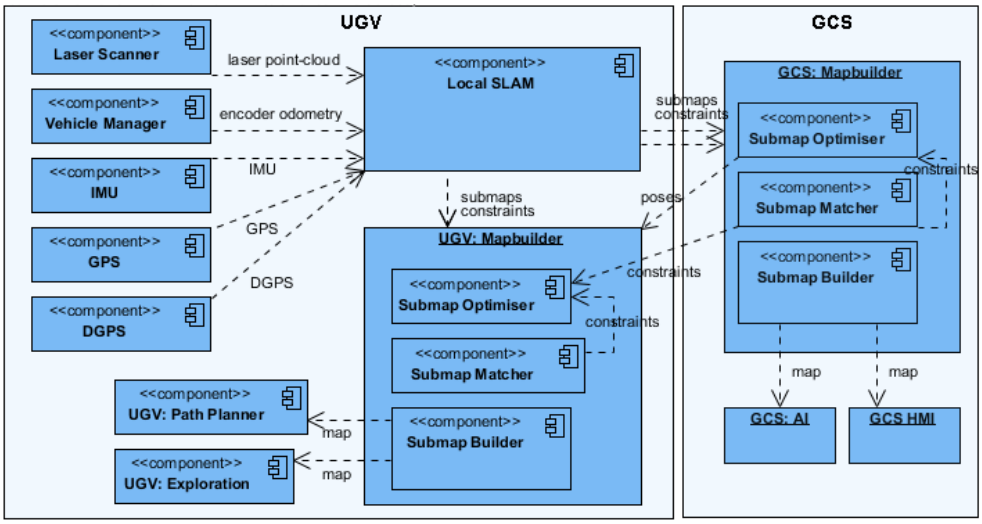


Figure 4.6. Mapbuilder logical design: Software components and data flows on a typical UGV participant (left) and GCS participant (right) are shown. Decentralized communications are performed using the DDS middleware.

4.4.1. Local SLAM Front-end

The role of the Local SLAM front-end is to create submaps, constraints and real-time pose estimates and broadcast them over the wireless network. Put simply, it is a single-robot SLAM algorithm combined with a heuristic that decides when to start each new submap. An instance of the front-end executes on each UGV, where it takes all sensor data as inputs and creates a sequence of submaps that provide an overlapping gridmap representation of the environment. The front-end's requirements, based on the WAMbot MRS, are to:

- Estimate UGV pose, broadcast in real-time >10 Hz locally, 1 Hz globally
- Build 10 cm submap gridmaps, broadcast >1 Hz locally, >0.2 Hz globally
- Robustly handle moving objects, including mobile OOIs, humans walking at 6 km/h
- Robustly handle difficult sensing conditions: sparse areas, featureless corridors
- Detect odometry errors to minimize corruption in submaps
- Compress submap gridmaps before broadcast
- Use less than 25% of available computation and memory requirements

The Mapbuilder architecture distributes the front-end computation fully— each UGV that is added to the team includes the necessary computing power to process the additional sensor data it UGV creates. The computation and storage requirements are bounded by the maximum size of each submap, which is ultimately bounded by the maximum distance a UGV drives in its submap and the range of its lidar sensors.

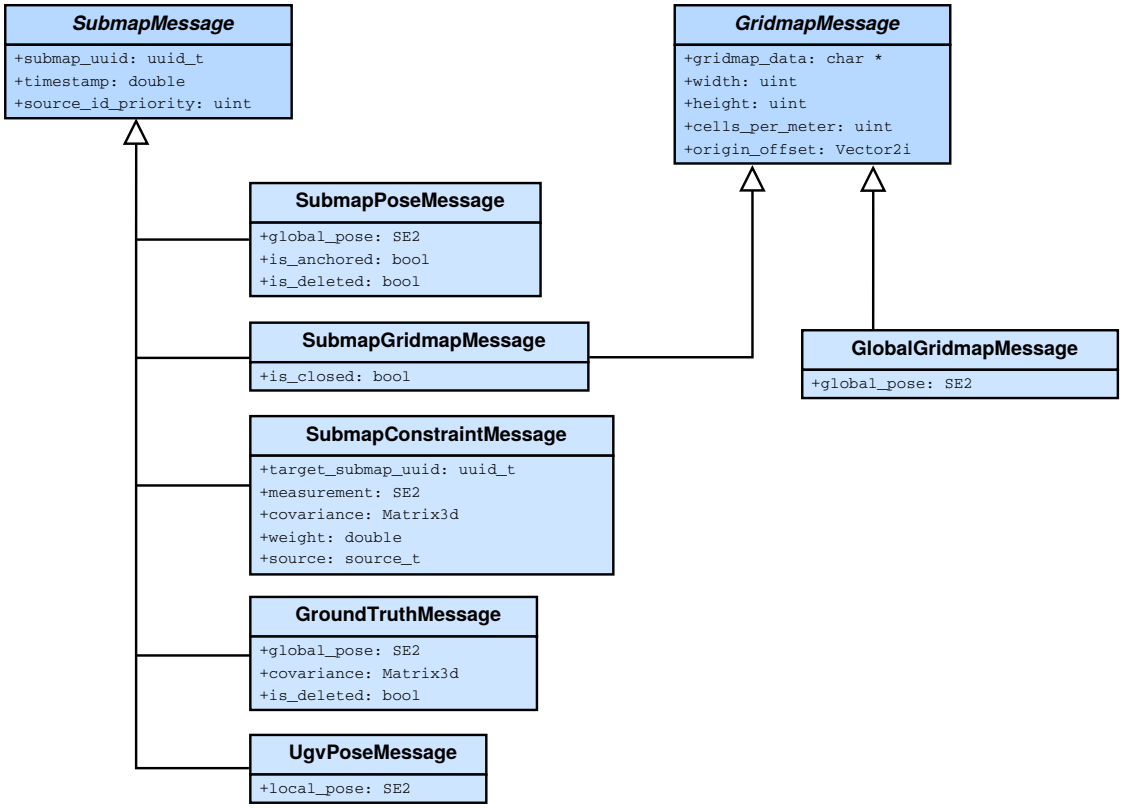


Figure 4.7. Mapbuilder message types: UML class diagram of demarshalled message types.

4.4.1.1. SLAM Algorithm

The architecture is flexible to the choice of single-robot SLAM algorithm implemented in the front-end. Many of the algorithms reviewed in Section 2.5 are suitable; the main requirement is that the algorithm can produce occupancy gridmaps while estimating UGV pose and accumulated pose uncertainty. The various single-robot SLAM algorithms have different strengths and weaknesses when considered for submapping front-ends:

- **Kalman Filter SLAM:** approaches, such as the EKF (Section 2.5.3.2), typically have bounded computational requirements for a fixed-size environment. While generally used to generate landmark-based maps, lidar scans can be rendered to create gridmaps [103, 104]. An inability to revisit previous data associations, combined with nonlinearities, results in small errors being "baked" into the estimate. For fixed-size submaps, however, the accumulation of errors can be managed and firewalled from subsequent submaps.

- **Particle Filter SLAM:** approaches, such as RBPF's (Section 2.5.3.7) are ideal for submapping, since the number of particles can be selected to suit the target submap size, environment complexity and processing budget [208, 209, 135]. Each particle stores an estimate for the UGV's path through the current submap, thus the filter

has the ability to maintain multiple pose hypotheses. When the submap is closed the final distribution of particles can be converted into either unimodal or multimodal constraints (Section 6.2.2).

- **Pose-graph SLAM:** approaches (Section 2.5.4) produce solutions that are closer to the full-SLAM solution. Depending on the shape of the environment, periodic scan matching and optimization offers the best opportunity to smooth the UGV's path and batch-align lidar scans into locally consistent configurations [224, 225, 226, 243]. While the path length, and computational requirements, are potentially bounded, smoothing a graph with dozens of poses can requires orders of magnitude more computation than filter-based methods; this cost is amortized in on-line incremental optimizations. In difficult environments, optimization can cause large step changes in UGV pose estimates, and pose uncertainty estimates recovered from marginal covariances are likely to be overly optimistic [223].
- **Sliding Window Filter:** approaches (Section 2.5.3.8) maintain a fixed-size set of historical UGV poses that are either filtered (e.g. an EKF) or smoothed (e.g. pose-graph SLAM). The oldest poses are marginalized out, bounding computational requirements. Within the Mapbuilder submapping framework, SWFs can optionally provide more stability to the Local SLAM estimate by *not* clearing the map at the start of each submap. This can assist with lidar scan matching and localization, particularly in degenerate environments with perceptual aliasing. Care is taken to ensure lidar scans from previous submaps are marginalized-out before closing the current submap. This maintains the ability to firewall uncertainty between submaps while avoiding double-counting measurements.

The algorithm selection depends on the target environment, the UGV sensor package and UGV computation available. Three different configurations are used in my research work, emphasizing the flexibility of the Mapbuilder design. The WAMbot front-end uses an EKF with ICP, while TM uses a SWF and Penn uses RBPF techniques. These configurations are described in Appendix A.

4.4.1.2. EKF SLAM with ICP

The Local SLAM front-end for the WAMbot UGVs is described here briefly, while he sensors and hardware are described in Appendix A. To minimize computational requirements the front-end uses an EKF (Section 2.5.3.2) with scan matching (Section 2.3.4.1). Submaps are built by aggregating lidar scans every 20 cm of motion or 20 degrees of rotation. To maintain an upper bound on the error within each submap, the heuristic described in Section 4.3.6 is used to decide when to close the current submap. This heuristic is augmented by a threshold on the percentage of lidar returns that are successfully aligned; this detects scan matching failures, particularly in sparse environments.

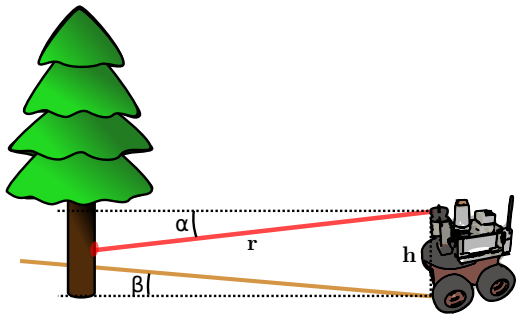


Figure 4.8. Local SLAM lidar prefilter: The terrain is assumed to be locally flat with inclinations less than β, while a "Manhattan World" assumption treats all objects as vertical. For each measurement r from the horizontal lidar scanner, the declination, α, is calculated from the pitch and roll of the UGV. Using trigonometry, the corrected lidar range is $r\cos\alpha$, while the inequality for accepting individual measurements is $0 < h - r\sin\alpha - r\cos\alpha\tan\beta$.

UGV pose is estimated using the EKF: each cycle begins by *predicting* the current pose using the latest wheel odometry and IMU data. Using this prediction the latest lidar scan is aligned against the current submap using scan matching [326]. Moving objects are detected using a RANSAC outlier rejection step [313]. The resulting scan matching alignment is used to *update* the EKF and pose estimate. The odometry noise in the EKF update is altered according to the local ground slope; this is estimated using UGV pitch and roll measurements from the IMU. As the slope increases, the model assumes that odometry noise due to wheel slippage increases also, this switches the filter to prefer scan matching over odometry as the slope increases. The filter is updated at the lidar scan rate of 25 Hz; UGV pose messages are sent at 25 Hz to the local partition and broadcast at 1 Hz globally.

Uneven terrain, particularly in outdoor environments, can introduce a range of scenarios that result in erroneous returns from a lidar scanner that is mounted horizontally. The ground grazing hits described in Section 2.3.2 are a good example. Although it is difficult to detect all of these errors, many can be filtered using UGV pitch and roll measurements from the IMU. To do this we make two assumptions: the first is the local flatness of the terrain, e.g. the inclination is never greater than β_{max}. The second is about the shape of the objects the environment— here we use a "Manhattan World" assumption [327]. Together, these assume the terrain is mostly flat and objects are mostly vertical.

The prefilter becomes a simple test based on trigonometry. Referring to Figure 4.8, each lidar measurement range r is first corrected to account for the declination, α, of the lidar measurement (caused by the pitch and roll of the UGV). The corrected lidar measurement, $r\cos\alpha$, and the height of the lidar above the ground, h, are used to create the inequality:

$$0 < h - r\sin\alpha - r\cos\alpha\tan\beta_{max} \tag{4.1}$$

Any measurement r with declination α that does not pass Equation 4.1 is filtered (removed) because there is a chance that it hit (or grazed) the ground. In practice, individual lidar measurements are first projected in 3-D using the UGV's pitch and roll to obtain r and α. It is trivial to apply this filter individually to each measurement in a lidar scan. The corrected, filtered, lidar measurements $r\cos\alpha$ are passed into the SLAM algorithm.

This prefilter assumes that obstacles in the environment have infinitely high walls. While this is not true, it is difficult to make a more principled assumption— the prefilter works well in practice for UGV pitch/roll angles below 30 degrees.

4.4.1.3. Submap Extents and Overlap

Submap gridmaps are rectangular, and their dimensions are dynamically increased to encompass the extent of all lidar returns. Thus the size of each closed submap gridmap is determined by the maximum range of the lidar (R), the shape of the environment and the heuristic that decides when to start the next submap. This heuristic is described in Section 4.3.6; here we will consider an average distance between submaps (D).

For a sequence of overlapping rectangular submaps created by a single UGV driving in an outdoor area, the maximum overlap between submaps separated by D can be calculated by the ratio $\frac{2R}{2R+D}$. Using the WAMbot UGVs as an example (Appendix A), the lidar range ($R = 20$ m) and average distance ($D = 3$ m) produces a maximum submap overlap of 93%. A single UGV could therefore produce submaps that overlap the same area up to 15 times. For reference, in the MAGIC challenge datasets (Appendix B) the overlap varied between 10 and 22 times per UGV.

While it might seem that submapping creates a large quantity of redundant submap data and computation, in the decentralized MR-SLAM case this spatial overlap between submaps is *required* to enable the distributed back-ends to compare and align map data. Given that each submap contains dozens of compressed lidar scans, this submapping approach results in less redundant computation than the equivalent decentralized MR-SLAM solution using lidar scan matching. Chapter 5 explains how this overlapping, redundant, data can be handled as efficiently as possible. For reference, histograms and statistics for real-world submaps are given in Figure 7.9 on page 183, Figure 7.14 on page 189 and Figure 7.19 on page 195.

4.4.1.4. Output Message Types and Policies

Three different message types are broadcast by the front-end, each timestamped by the local clock. When submaps are closing, the front-end broadcasts their compressed gridmap data along with a constraint linking the closed submap to a newly created one. This

minimal set of messages is sufficient for the distributed instances of the Mapbuilder back-end to recreate the pose graph and produce global gridmaps.

If a UGV is driving infrequently, such as performing a surveillance task, it might remain in a single submap for extended time periods. Therefore to produce real-time global maps, the front-end also broadcasts incomplete gridmaps for the currently open submap. This gridmap data is sent to the local partition at 1 Hz and the global partition every five seconds to keep the distributed global gridmaps up-to-date. A flag in the message indicates that the submap is still open, and is not yet immutable (Section 4.3.3).

The front-end's publishing QoS policies vary depending on message type. The DDS buffer size, n, is specified for each:

1. `SubmapConstraint` ($n = 1000$): Constraint messages are assigned the highest priority, since they define the entire pose graph structure. A large buffer size is appropriate, since the messages are both small and comparatively valuable.

2. `SubmapGridmap:` Gridmap messages are assigned the next highest priority, since they encode the actual shape of the environment. While gridmaps are compressed, they form the bulk of the MR-SLAM data.

   a) **Open** ($n = 0$): Gridmaps that are open are disposable since each front-end re-broadcasts them periodically, and stale messages are not useful. An advanced implementation only re-broadcasts parts of the gridmap that are changing. For local UGV navigation, open gridmaps can be sent to the local partition at the lidar's scan rate.

   b) **Closed** ($n = 100$): Gridmaps that are closing are higher priority, since they are only transmitted once. There is a trade off when choosing n: if it is too low there is a risk of small holes appearing in the global gridmap, if n is too high the network may be overwhelmed after an extended communications outage.

3. `UgvPose` ($n = 0$): Real-time UGV pose estimates in the current submap's frame, are broadcast frequently. They become stale immediately and are disposable. For local UGV navigation, pose messages can be sent to the local partition at the lidar scan rate.

To create `SubmapGridmap` messages, a rendering algorithm ray traces the accumulated lidar scans into an empty gridmap using the techniques described in Section 2.3.4.3. For robustness and efficiency the gridmap data is broken into smaller 32×32 cell gridmap tiles that can be broadcast as UDP packets without fragmentation. The occupancy data in these tile cells is quantized and compressed using run length encoding. This basic compression scheme provides around 50:1 compression ratio, reducing gridmaps to 2 % of their original size.

4.4.2. Mapbuilder Back-end

Distributed instances of the Mapbuilder back-end provide all of Mapbuilder's MR-SLAM capabilities. Using the DDS middleware, each back-end subscribes to *all* of the MR-SLAM messages listed in Figure 4.7 to maintain a local copy of the pose graph and submap gridmap data. The back-end's requirements, based on the WAMbot MRS, are to:

- Optimize pose graphs robustly and efficiently, < 5 seconds per iteration.
- Output large (5000×5000) gridmaps to the local partition at >1 Hz.
- Match submaps to generate robust constraints, < 5 seconds per match.
- Broadcast `SubmapPose` messages globally to maintain decentralized pose graph.
- Output `SubmapPose` updates to the local partition at >1 Hz.

To meet these requirements, the back-end computation is performed in three tasks:

- **Optimizer:** performs incremental pose graph optimization.
- **Builder:** fuses the submap gridmaps to output global or windowed gridmaps.
- **Matcher:** searches for new $\mathbb{SE}(2)$ constraints that align overlapping submaps.

Task scheduling depends on the computational resources allocated to the back-end. In this research work, the Optimizer, Builder, and Matcher tasks are executed in parallel in separate CPU threads. This adds complexity by requiring an extensive use of mutexes and semaphores, however it allows the back-end to take advantage of the multiple processing cores available in modern CPUs. Any spare processing time the UGV and GCS participants have can be dynamically allocated to the back-end, which typically runs the Matcher task to perform additional submap alignments. This section describes the back-end Optimizer (Section 4.4.2.3), Builder (Section 4.4.2.4), and Matcher (Section 4.4.2.5) tasks in detail.

4.4.2.1. Maintaining Distributed Pose Graphs

For efficiency, each Mapbuilder back-end instance uses a single set of DDS subscriptions and a single set of message call-back handlers. To ensure pose graph updates are performed atomically and in a thread-safe manner, messages are not processed when they are received. Instead they are demarshalled and placed in temporary queues (the demarshalled messages are shown in Figure 4.7). These queues are processed in a batch update to the pose graph, directly before the Optimizer is executed. The batch update uses Filter 1, from Section 4.3.9, to decide which messages to accept or ignore.

To ensure pose graph updates are deterministic, special care is taken to only compare timestamps embedded in messages that originated from the same participant. This avoids

the need for participants' real-time clocks to be closely synchronized (the only requirement is for the clocks to increase monotonically).

Recognizing that messages can arrive out of order, incoming messages are handled in a greedy manner. For example, if a `SubmapConstraint` message arrives that joins submaps that do not exist in the local pose graph, the missing submaps are automatically created in the graph.

To avoid double-counting measurements, a common problem in decentralized MR-SLAM architectures, incoming submap constraints are reversed as needed so that they are ordered deterministically. Reversing is performed using $\mathbf{c}_b^a = \ominus \mathbf{c}_a^b$ so that the submap UUIDs are ranked $a < b$. This design ensures that submap pairs have only has one constraint, and allows for multimodal constraints.

The $\mathbb{SE}(2)$ pose graph is stored using submap pose and constraint classes that are derived from the g^{2}o library's, `g2o::VertexSE2` and `g2o::EdgeSE2` types, respectively. Ground truth constraints are derived from `g2o::EdgeSE2Prior`. Refer to [238, 243] for an introduction to the g^{2}o optimization library. Deriving classes from g^{2}o types allows the pose graph optimization to be performed directly on the graph and matrix decompositions to persist between invocations, greatly improving efficiency.

4.4.2.2. Constraint Types

The Mapbuilder architecture can build and optimize pose graphs with various spatial constraint types and both direct or iterative methods. Potential types have residual error functions with Jacobians that can be derived analytically, or calculated numerically ([243]). Three $\mathbb{SE}(2)$ constraint types are used in this research:

- **Unimodal:** the typical Gaussian $\mathbb{SE}(2)$ pose graph constraint described frequently in the literature [238]. Jacobians are calculated analytically.
- **Max-Mixture:** Olson and Agarwal's multimodal Gaussian approach was implemented for comparison in Chapter 6 [57]. Jacobians are calculated analytically also.
- **COMBO:** this novel multimodal Gaussian constraint type is described in Chapter 6. Jacobians are calculated with a combination of numerical and analytical techniques.

4.4.2.3. Optimizer: Incremental Pose Graph Optimization

The Optimizer task uses the standard pose graph optimization techniques described in Section 2.5.4.7. The problem formulation is almost identical with submaps: the submap poses, $\mathbf{p}_a^{\mathcal{W}}$, are estimated instead of the UGV poses, $\mathbf{r}_t^{\mathcal{W}}$. Either direct or iterative methods

can be used to find the maximum likelihood estimate (MLE)— the Levenberg Marquardt algorithm (Section 2.5.4.9) with Cholesky factorization is used in this work.

To limit the Optimizer task execution time, each invocation performs an incremental MLE optimization with a upper limit to the number of iterations (e.g. ≤ 10). Most of the time, however, the pose graph will be close to an optimum; the optimization will converge in one or two iterations and terminate early. Between invocations, additional poses and constraints can be added to the pose graph that could, for example, result in large loop closures. These may require dozens of iterations and several invocations of the Optimizer (i.e. several batches of iterations) to converge..

Care must be taken to ensure that the optimization is well-formed, particularly when using direct methods with matrix decompositions, such as the Cholesky factorization used here:

- All constraint uncertainties must be positive semidefinite.
- Every submap must be connected by at least one constraint.
- Every connected set of submaps must have at least one ground-truth constraint.

When messages arrive out of order, it is highly likely that the pose graph will not be fully-connected. Submaps that are not connected to the main pose graph are omitted from the optimization to avoid *gauge freedom* (refer to [234]). Gauge freedom must be avoided, since the resulting Hessian matrix will be rank deficient such that it cannot be factorized (Section 2.5.4.9).

It is important that the initial pose of any new submap is "close" to the global optimum, since numerical optimization using floating-point numbers can become unstable with large constraint residual errors (Section 2.5.4.4). A simple spanning-tree algorithm can be used to initialize any submap pose estimates, provided the pose graph has at least one ground-truth constraint.

To implement the hybrid-decentralized approach described in Section 4.3.9, the Optimizer uses Filter 2 to determine which submaps in the pose graph are pinned, and which can be freely optimized. For lower-priority participants, most submaps will be pinned resulting in a very minimal pose graph optimization. For efficiency, only unpinned submaps are added to the optimization along with the constraints they are directly connected to.

The four rules listed in Section 4.3.9.2 determine if the back-end should broadcast any `SubmapPose` messages to maintain synchronization with the other participants. These pose updates are filtered first, however, since most incremental optimizations result in numerically small changes to the global gridmap. A simple filter projects the four corners of each submap gridmap into the global frame: if any corner moves by more than half a cell-width, the update is included.

4.4.2.4. Builder: Occupancy Gridmap Fusion

The Builder's role is to efficiently create occupancy gridmaps by fusing thousands of submap gridmaps together. It may produce global or windowed gridmaps depending on the high-level requirements and available processing resources. Each execution of the task starts with a blank output gridmap, where the probability of each cell being occupied is $p(\mathrm{m}_i) = 0.5$. The Builder iterates over all submap gridmaps, fusing them into the output gridmap. For each submap gridmap, the individual cells are transformed into the output gridmap's coordinate frame and the corresponding cells are fused using a binary Bayes filter (Section 2.3.4.3).

Although it is computationally inexpensive to fuse individual cells, submaps are often very densely overlapped. For reference, in the Phase 2 dataset in this research work 175 million submap cells are fused into 2.5 million output gridmap cells. It is nontrivial to achieve this scale of gridmap fusion in real-time. Chapter 5 describes an efficient gridmap fusion algorithm that can fuse 45 thousand submaps, and 3.9 *billion* submap cells per second, enabling the distributed submapping approach described in this chapter. The output gridmap data is efficiently delivered to the local partition (high-level consumers) over shared memory.

4.4.2.5. Matcher: Submap Constraint Search

As discussed in Section 4.3.7, to connect sequences of submaps from individual UGVs into a fully connected multi-robot pose graph, constraints need to be added that connect spatially overlapping pairs of submaps. The Matcher task searches for these $\mathbb{SE}(2)$ constraints, finding both large and small loop closures that make the pose graph increasingly connected, while refining submap pose estimates.

For the distributed architecture to function, each UGV should be designed with sufficient computational resources to match, at least, each submap that it creates to one other submap. For example, if each UGV creates a new submap every ~3 meters, and has a top speed of 0.5 meters per second, then searches should take less than 6 seconds to keep the MR-SLAM pose graph connected.

The odometry error that accumulates as UGVs explore is assumed to be bounded by ± 3 meters and ± 20 degrees. This assumption, explained in Section 4.1.4, defines a volumetric search space between pairs of submaps that must be considered by the Matcher[6]. Section 2.3.4.1 describes several algorithms that could perform this type of search; the main requirement is that the algorithm returns a relative constraint in $\mathbb{SE}(2)$ and a principled estimate of the constraint's covariance. Consideration should be given to

[6] Note that constraints in $\mathfrak{se}(2)$ have three degrees of freedom that define a volume.

the algorithm's robustness in different environments. Chapter 5 describes an algorithm that samples the entire search volume to estimate constraints with multimodal Gaussian distributions.

The Matcher task maintains a list of submap pairs that require matching. This list is formed by iterating over submaps in the local pose graph looking for pairs of submaps that overlap spatially. An overlapping submap pair is added to the list if it has: 1) no existing constraint, 2) an existing constraint where their relative pose has changed significantly, or 3) either of their states have changed from open to closed.

A heuristic is used to decide which submap pairs in the list a participant should prioritize matching, it is designed to prevent participants from performing redundant searches. Existing constraints prevent most redundant searches, except for the occasion where two participants begin matching the same pair at the same time. To avoid this, this each participant sorts and prioritizes the list:

1. Submap pairs that include a submap that was created by the participant itself in the last 10 seconds. This gives UGV participants the opportunity to quickly connect their newest submaps into the pose graph, while preventing other participants from redundantly performing the same search.

2. Older submaps are matched by any participant, however to distribute the search spatially across the pose graph, and computationally across the entire clique, the local list of submap pairs is sorted deterministically using the last 32 bits of both submaps' UUIDs. These bits are uniformly distributed random numbers, and once sorted they are used to slice the list of submap pairs into even work-loads. Each participant starts matching submap pairs at an offset in the list determined by its priority ranking in the local clique (see Section 4.3.9).

Each search is performed ± 3 meters and ± 20 degrees around the current relative pose. The search results are broadcast as a `SubmapConstraint` message that includes a weighting parameter that reflects confidence in the match. Locally-created constraints are integrated into the local pose graph via the back-end's main message queue (Section 4.4.2.1) to remain deterministic.

4.4.2.6. Output Message Types and Policies

The back-end's publishing QoS policies vary depending on message type. The DDS buffer size, n, is specified for each:

1. `SubmapPose:`

   a) **Global Partition** ($n = 0$): Submap pose estimates are broadcast globally to perform pinning in the hybrid-decentralized pose graph. Since synchronization between distributed participants is handled by by the priority-based filters described in Section 4.3.9, delivery guarantees are not required .

   b) **Local Partition** ($n = 0$): DDS uses shared memory to deliver real-time submap pose estimates to local software components. QoS is not required.

2. `SubmapConstraint` ($n = 1000$): Constraint messages are assigned the highest priority, since they define the entire pose graph structure. A large buffer size is appropriate, since the messages are both small and comparatively valuable.

3. `GlobalGridmap` ($n = 0$): DDS uses shared memory to deliver global gridmaps to local software components. QoS is not required.

4.4.3. Mapbuilder GUI

The GUI implements a point-and-click user interface that allows operators to interact with the MR-SLAM pose graph and submaps[7]. Its main purpose is to enable operators to interactively correct for the non-Gaussian odometry errors described in Appendix B. To correct these errors the operator needs to be able to modify the pose graph. Operator interaction requirements include:

- Select, move and rotate a submaps (`GroundTruth` messages).
- Select and delete submaps (`SubmapPose` messages with deleted flag set).
- Select and anchor submaps (`SubmapPose` messages with anchored flag set).
- Create submap priors from aerial imagery (`SubmapGridmap` messages).

These four operator interactions are described in this section, since they affect all UGV and GCS participants and their distributed copies of the pose graph. For reference, design requirements that allow the operator to interact with the graphical displays are:

- Display, pan and zoom the global gridmap output.
- Display and hide configurable overlays (real-time UGV pose, historical UGV tracks, pose graph structure and aerial imagery).
- Highlight individual submaps, highlight submap groups filtered by source UGV.
- Highlight unanchored parts of the global map.
- Save and restore pan and zoom presets for efficient interaction.

[7] Videos of the GUI being used are available online: http://reid.ai/thesis

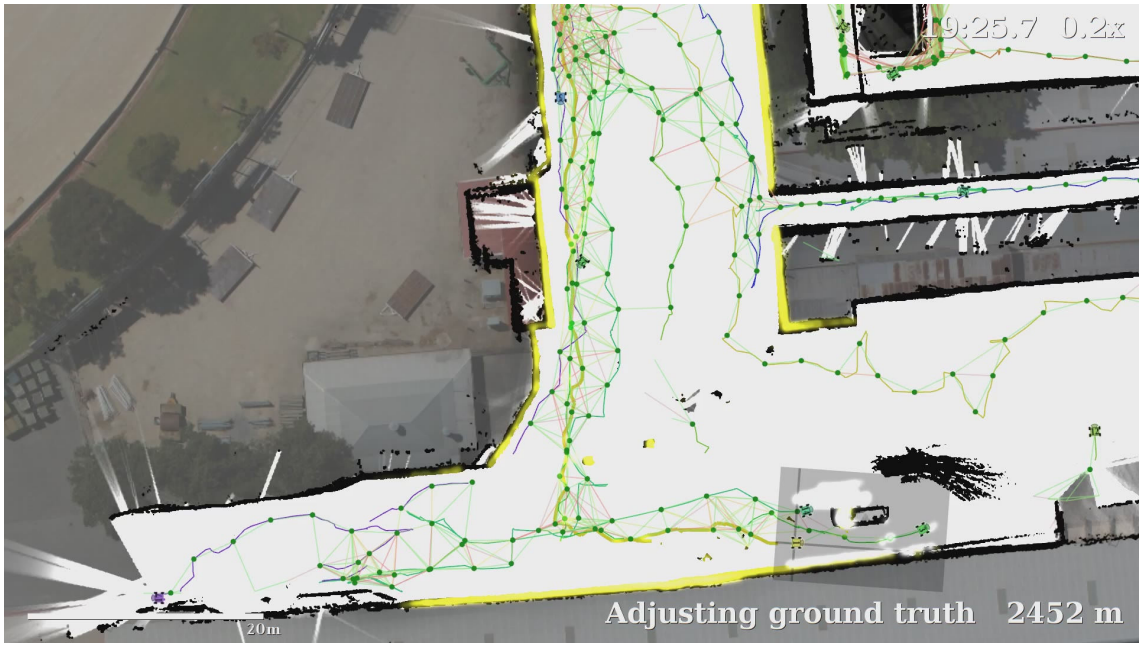


Figure 4.9. Mapbuilder graphical user interface: Screen-shot showing ten of the UGVs exploring Phase 2 of the MAGIC challenge. A global occupancy gridmap is blended with an aerial image and various configurable overlays. Markers indicate current UGV poses, while lines indicate their trajectories. Green dots indicate submaps and green lines the active pose graph constraints. Animations are used extensively to minimize clutter. In this screen-shot, the submaps for a selected UGV are highlighted (yellow, pulsating), while in the lower right a single submap has been selected with the mouse (highlighted with a gray rectangle). Errors can be seen in these submaps resulting from many non-Gaussian odometry errors caused by a long ditch, the operator is manually deleting them (click to view video).

The GUI maintains its own local copy of the pose graph by subscribing to all MR-SLAM message types. It displays an aerial image for the operator, over which it draws the current global gridmap, the pose graph structure, highlighted submaps and UGV pose markers. The global gridmap is delivered efficiently over the local partition so that the display updates interactively and in near real-time. A screen-shot of the GUI is shown in Figure 4.9.

While a GUI is an important part of any deployable MRS, the only system I could find in the literature that allows operator interaction with the MR-SLAM pose graph is described by Olson et al. in [53]. I could not find any GUIs that enable interactions with distributed pose graphs.

4.4.3.1. Distributed Operator Interactions

For the GUI to affect the other participants' distributed pose graphs, operator interactions are broadcast as messages (summarized below).The participant running the GUI must have the highest priority for the messages to pass through every other participant's priority-based filters (Section 4.3.9).

Submaps are moved and rotated into alignment by sending `GroundTruth` constraint messages. The operator performs these alignments by clicking and dragging on submaps with the mouse. Typically the operator will align submaps with either the aerial image, or other submaps.

To assist the operator managing dozens of UGVs, particularly during initialization, the concept of *anchoring* submaps is introduced. Unanchored submaps float on top of the map display, visually pulsating to draw the operator's attention; they are ignored by the Builder and Matcher tasks. The operator can move and rotate unanchored chains of submaps into alignment, before issuing a command to anchor them (`SubmapPose` messages are broadcast with their "anchored" flag set). The anchoring process is greedy, since it propagates from the selected submap through all connected submaps in the pose graph. In this way, every UGV and its submaps begin unanchored from the global map, and are only fused and matched with other anchored submaps after the operator's command. Thus submaps generated by uninitialized UGVs are excluded from the output gridmap and they are not incorrectly joined to other submaps.

If the operator identifies a submap that has been damaged by non-Gaussian errors, the operator selects it and issues a delete command. To propagate the change to other participants, submap deletions are broadcast via `SubmapPose` messages that have a "deleted" flag set. The other participants delete the submap's gridmap data and any constraints connected to it, however they leave the submap in the pose graph. This removes all influence in the optimization and output gridmap, while allowing the deleted flag to propagate to the other hybrid-decentralized participants as in Section 4.3.9.

Deleting submaps can cause "islands" of submaps to be disconnected from the main pose graph. If these disconnected submaps are not constrained by any ground-truth constraints, they are automatically unanchored and begin pulsating to draw the operator's attention. These newly unanchored submaps are excluded by the Optimizer, which is crucial for the gauge freedom rules described in Section 4.4.2.3.

4.4.3.2. Aerial Image Priors

Section 4.1.4 explains how a trained operator can manually add submap priors from aerial imagery. This is performed in the GUI by creating *synthetic* submaps. Synthetic submaps are generated by simulating a 360 degree lidar scan at the selected location; the simulated lidar measurements are ray cast into the aerial image and stop when an edge in pixel intensities is detected. The simulated lidar scan data is rendered into a gridmap using the same techniques as the Local SLAM component (Section 4.4.1). The GUI broadcasts the synthetic submap as a closed `SubmapGridmap` message, with a fixed `SubmapPose`, so that the distributed back-ends can then perform loop closures to them.

This technique assumes the edges in the aerial images are aligned with actual objects in the environment. Figure B.4 (b) on page 237 shows two adjacent buildings with very different eaves that appear almost identical when viewed in the aerial image. Building eaves such as these, and other features like shadows on the ground, lead to difficulties using this approach, even for a trained operator. To assist, a 2-D edge map image is precomputed off-line that can be manually edited to remove shadows and other clutter. Matching 2-D lidar data to aerial imagery is a complex visual perceptual problem that is addressed in other works such as [265, 266, 267].

4.4.3.3. Output Message Types and Policies

The GUI's publishing QoS policies vary depending on message type. The DDS buffer size, n, is specified for each:

1. `SubmapPose` ($n = 0$): Submap pose messages are broadcast globally to perform pinning in the pose graph. Since synchronization between distributed participants is handled by by the priority-based filters described in Section 4.3.9, delivery guarantees are not required.

2. `GroundTruth` ($n = 1000$): Ground truth messages are assigned the highest priority, since they allow the operator to tie down the pose graph structure. They are small and valuable messages.

3. `SubmapGridmap` ($n = 100$): Submap priors are generated from aerial imagery and broadcast as closed `SubmapGridmap` messages.

4.5. System Verification

This research work and the Mapbuilder system are designed to meet a long list of requirements (Section 4.1.3) under several assumptions (Section 4.1.4). This section addresses three of these requirements, highlighting how they are achieved. Results shown in Chapter 7 demonstrate that the remainder of the MR-SLAM requirements have been met.

4.5.1. Flexible Global Localization

The pose graph approach described here is flexible with respect to the method of global localization. If a UGV team is deployed indoors, the first UGV submap pose can be initialized to $\mathbf{p}_0^{\mathcal{W}} = [0, 0, 0]^T$, and the system will function normally without global localization. For a georeferenced UGV deployment in GPS-denied environments, such

as in dense urban settings or when electronic countermeasures are in place, submaps (and hence their UGVs) can be aligned to a globally-referenced aerial image (Section 4.4.3.2).

If reliable GPS sensor data is available, it can be used to create loose ground-truth constraints that gradually "tie down" a pose graph so that it becomes georeferenced. In ideal situations GPS can provide position and heading accuracies better than ±5 meters and ±10 degrees outdoors. For each GPS ground-truth constraint the uncertainty is estimated using the GPS sensor's horizontal DOP value (Section 2.4.3). The DOP model is overconfident, however, and does not account for non-Gaussian errors such as multipath reflections [328]. GPS data is *not* used in the results in Chapter 7, since the UGVs were deployed in urban areas that experienced frequent multipath errors. Refer to Appendix B, and Figure B.7 on page 239 for more detail.

4.5.2. Consistent Coordinate Frames

Teams of UGVs should perform spatial tasks with respect to their local environment, not a global coordinate frame that is maintained by pose graph optimization. Large loop closures, for example, can cause submaps (and the UGVs that are building them) to "jump" in the global frame, causing inconsistencies in global coordinates.

To maintain consistent coordinate frames, *all* UGV operations are performed relative to one or more submap coordinate frames. Navigation waypoints, for example, are issued as coordinates with respect to one or more submap frames. This is essential, since globally-referenced waypoints may become inaccessible if a loop closure causes an obstacle such as a wall to move. Section 8.3 describes an approach for global navigation that plans directly in submaps— this is future work.

To ensure consistency during operator GUI interactions, care needs to be taken when converting global coordinates into submap-relative coordinates. If the operator provides a global waypoint, $\mathbf{r}^{\mathcal{W}}$, it is converted to a submap-relative coordinate with $\mathbf{r}^{a} = \ominus \mathbf{p}_{a}^{\mathcal{W}} \mathbf{r}^{\mathcal{W}}$, where $\mathbf{p}_{a}^{\mathcal{W}}$ is the closest submap to the waypoint with a gridmap that includes the waypoint.

4.5.3. Heterogeneous UGVs

The submap-based approach described in this chapter is well-suited to MR-SLAM with heterogeneous UGVs. The back-end can incorporate data from UGVs with different types of sensors, provided the Local SLAM front-end can generate:

- 2-D occupancy gridmaps that appear similar from similar vantage points.
- Spatial constraints between submaps with principled uncertainty estimates.
- Real-time submap-relative pose updates.

To generate these outputs, each UGV sensor package would typically include a lidar scanner. Results in Chapter 7 demonstrate heterogeneous UGVs performing MR-SLAM with fixed, nodding and sweeping lidar configurations (see Appendix A).

5 Efficient Occupancy Gridmap Fusion and Matching

The distributed multi-robot SLAM approach described in Chapter 4, produces thousands of overlapping occupancy gridmaps that become computationally expensive to fuse into global gridmaps and to search for loop closures. To maintain on-line performance in large-scale MR-SLAM deployments, two algorithmic contributions in this chapter show how to utilize the massively-parallel processing capabilities of modern graphics processing units (GPUs). In Section 5.1 I introduce GPUs and their programming model, while in Section 5.2 I describe how submap gridmaps are similar to the 2-D textures that GPUs are optimized for. Section 5.3 shows how the binary Bayes filter can be parallelized into an algorithm that can fuse tens of thousands of gridmaps per second. Finally, in Section 5.4 I describe a correlative submap matching algorithm that extracts multimodal Gaussian constraints from pairs of submaps.

5.1. Introduction

Chapter 4 describes Mapbuilder, a distributed architecture for MR-SLAM that generates thousands of densely overlapping submap gridmaps. The research work in this chapter shows how these submap gridmaps can be efficiently manipulated and describes two algorithms that enable *on-line* distributed MR-SLAM.

5.1.1. Research Contributions

These contributions were required to realize the Mapbuilder MR-SLAM system:

1. **GPU-based occupancy gridmap fusion:** merges tens of thousands of submaps per second using an efficient highly-parallelized adaptation of the log odds binary Bayes filtering algorithm (Section 5.3).

2. **GPU-based multimodal constraint generation:** extracts descriptive, yet minimalistic, representations of the spatial relationships between submaps. A highly-parallelized gridmap correlation algorithm calculates likelihood volumes and extracts multimodal Gaussian distributions (Section 5.4).

The computers in a typical UGV operate without any visual display and the on-board GPU is often unused. The GPU-based gridmap algorithms described in this chapter allow the most computationally expensive parts of the Mapbuilder back-end (Section 4.4.2) to be off-loaded to the unused GPU, thus freeing the CPU for other tasks.

The occupancy gridmap fusion algorithm described here is demonstrated in the Mapbuilder back-end's Builder task (Section 4.4.2.4), while the multimodal constraint generation algorithm is demonstrated in the Matcher task (Section 4.4.2.5). The distributed nature of the back-end design allows each instance to use 100 % of the participants GPU: the Builder task executes periodically, while the Matcher task is allocated all spare GPU computation time to search for loop closure constraints.

5.1.2. Graphics Processing Units

While computer microprocessors (CPUs) have become exponentially more powerful over the last few decades [329], physical limits have prevented CPU clock speeds from increasing further. Instead, recent advances in computational power have been enabled by packaging multiple processors into a single silicon chip. Two or four "cores" are frequently packaged into these multi-core CPUs in a manner that allows them to share the execution of multi-threaded programs. Even using the advanced vector operations on modern CPUs, however, four processing cores running at 2-3 GHz are still too slow at manipulating the *billions* of gridmap cells per second required for the Mapbuilder back-end to run in real-time[1].

GPUs have evolved over the last two decades into multi-core processors with thousands of cores. Their arithmetic and logic cores are highly-optimized for 3-D computer graphics operations such as rendering 2-D textures. They are well-suited for algorithms where the computation can be broken into thousands or millions of threads that can be executed in

[1] The memory caching techniques used in modern CPU's are not well-suited to the types of sequential read-write memory access patterns that occur when building large global gridmaps.

parallel. Their massively-parallel designs have memory architectures that are optimized for sequential read-write access patterns.

Driven primarily by 3-D computer graphics and games, GPUs are now commodity hardware. For reference, the 1664-core GPU used in this research retails for USD $320 and has more than twice the computational power necessary for on-line MR-SLAM with 23 UGVs (it can process over 100 billion textured pixels per second). The research problem considered here, therefore, is how to reformulate the gridmap fusion and constraint generation algorithms to leverage these massively-parallel capabilities.

5.1.3. Programming Model

GPUs are not programmed with traditional languages and compilers because of the special functionality required to synchronize memory access and execution across groups of cores. While proprietary languages have been created by various manufacturers [329], two notable open standards for programming GPUs are OpenGL GLSL [330] and OpenCL [331]. For manipulating gridmaps, both languages are equally well-suited, however OpenGL GLSL is an older standard and is available on a a wider range of GPUs (OpenCL was not available on the WAMbot UGV computers).

In this research work, the GPU is interfaced using OpenGL version 3.0 , and the gridmap algorithms are implemented in the OpenGL GLSL shader language version 1.30 (an open standard since 2008) [330]. For reference, most commodity GPU hardware, including modern mobile phone processors, provide the OpenGL GLSL capabilities necessary to run these algorithms.

5.2. Submaps as Textures

The insight that makes Mapbuilder's submapping approach computationally feasible is that rectangular submap gridmaps are very similar to the 2-D textures that can be efficiently manipulated by GPUs. The thousands of cores in a modern GPU are very efficient at transforming coordinates, reading pixels from 2-D textures and output memory buffers, blending the red, green, blue and alpha (RGBA) color components and writing the result back into the output memory buffer— all while synchronizing the output buffer so that the reading and writing by thousands of cores at the same time do not interfere. These capabilities are designed for applications like computer games, where millions of 2-D textured polygons are drawn into output buffers to render views of 3-D scenes. When considering rectangular gridmaps as 2-D textures, a single gridmap cell is equivalent to a single pixel in the texture.

In this work, OpenGL is used to store submap gridmaps as 2-D textures, which depending on the platform, will reside either in dedicated high-speed GPU memory or in memory that is shared with the CPU. Since the Mapbuilder architecture specifies that closed submaps are immutable, they are only converted to textures and uploaded to the GPU's memory *once*— this is an important aspect of the design. Given each UGV maintains only a single open submap, the number of new submap textures being uploaded to the GPU each second is less than the number of UGVs in the team.

If the GPU's computational power is limited, or if only smaller "windowed" gridmap outputs are required, it is trivial to limit the extents of the gridmap output. Similarly, if a participant's GPU has limited memory available, the OpenGL drivers will move any unused submap textures back to the host CPU's memory. Furthermore, if the host CPU's memory is limited, then submap textures can be buffered to permanent storage (hard disk). Submaps that do not overlap the current gridmap output window can be trivially filtered and off-loaded to permanent storage.

It is inexpensive to store additional 2-D spatial data alongside the occupancy gridmap data in a submap texture. GPUs are designed for rendering color images, and therefore their memory access is optimized for reading and writing pixels with multiple color components. A typical format is RGBA, with 8 bits per channel, or 32 bits total per pixel. Textures in this format provide four spatially-aligned layers of data with 8-bit accuracy.

Mapbuilder utilizes the four color channels in each submap texture. Before uploading submap textures to the GPU, the RGBA channels are precomputed and set to:

- **Red: Occupancy gridmap**, refer to Section 2.3.4.3 and Figure 4.4 for examples. Occupied cells ($p(\mathrm{m}_i) = 1$) are stored as zero, free-space cells ($p(\mathrm{m}_i) = 0$) are stored as 255, and unknown cells ($p(\mathrm{m}_i) = 0.5$) are stored as 127. This is the main channel used by the gridmap fusion algorithm.

- **Green: Likelihood gridmap**, estimates the location of objects in the submap. Likelihood data is not broadcast by the front-ends, as such it is approximated using the occupancy data. The likelihood map is zeroed, all occupied cells are set to 255 and then a Gaussian blur with $\sigma = 2$ pixels (0.2 m) is applied to approximate lidar sensor noise.

- **Blue: Height cost gridmap**, stores navigation height cost information for each cell. This channel was not utilized in this research work because the three different UGV front-ends produced height cost information in different formats that could not be fused.

- **Alpha: Navigation cost maps**, stores precalculated cost maps to accelerate path planning. The first step in many path planning algorithms is to build a cost

map by repeatedly dilating and smoothing the global occupancy gridmap. This computationally expensive step can be avoided by fusing submaps with precalculated cost maps.

5.3. GPU-based Occupancy Gridmap Fusion

5.3.1. Problem Statement

While occupancy gridmaps are well-suited for distributed MR-SLAM and provide a rich representation of the environment, at large scales the computational cost of fusing gridmaps becomes prohibitive. Although it is inexpensive to fuse individual gridmap cells with the binary Bayes filter (Section 2.3.4.3), submaps are often densely overlapped. In the Phase 2 dataset in this work, for example, 175 million submap cells are fused into 2.5 million output gridmap cells. It is nontrivial to achieve this scale of gridmap fusion in real-time. This section shows how the binary Bayes filter can be reformulated into a gridmap fusion algorithm that exploits the massively-parallel capabilities of modern GPUs.

At the core of this problem is the fact that the cells in sets of submaps are rarely aligned. Gridmaps quantize the environment into square cells (e.g. $10\times10\,\text{cm}$), however pose graph optimization ensures that submaps and their cells overlap at arbitrary offsets and angles. When fusing cells into an output gridmap, each cell will typically overlap 1-6 output cells. Figure 5.1 illustrates the case of a single submap cell overlapping four output cells; the probabilistically correct approach here is to perform four weighted binary Bayes filtering operations, however the contention in memory access that occurs when reading and writing the output buffer becomes prohibitive. A simpler approach might be to choose the closest output cell, however this introduces spatial noise into the gridmap's probability distribution.

5.3.2. Previous Work

Many authors have recognized the potential for GPU-based algorithms in robotics and SLAM. Limiting this discussion to SLAM in $\mathbb{SE}(2)$, Yguel et al. demonstrated rendering lidar measurements into gridmaps using GPUs [332], however only with a single gridmap in a simulated environment. Other authors have demonstrated building gridmaps with GPUs in both indoor, [329], and outdoor environments [333].

Several authors have demonstrated fusing sets of gridmaps from multiple UGVs [334, 280, 335], with the largest real-world dataset being 45×25 meters. These authors did not attempt to optimize the fusion algorithm. Only Strom and Olson [56] have demonstrated fusing thousands of gridmaps at scales comparable to this research work. They optimize

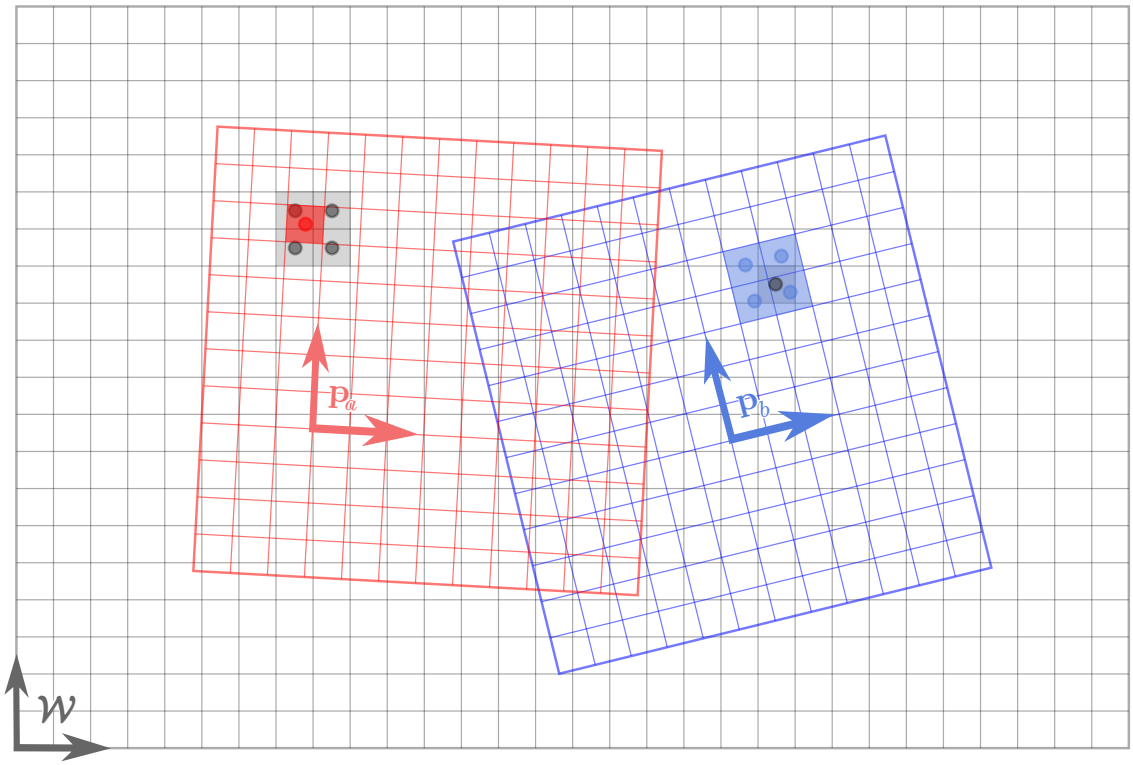


Figure 5.1. Occupancy Gridmap Fusion: Two submap gridmaps with origins at $\mathbf{p}_a^{\mathcal{W}}$ (red) and $\mathbf{p}_b^{\mathcal{W}}$ (blue) are being fused into the global output gridmap $\mathcal{W}$ (gray). In the naive approach, the highlighted cell in $\mathbf{p}_a^{\mathcal{W}}$ (red square) must be fused into the four cells in the output gridmap (gray squares). This becomes inefficient when submaps are densely overlapped. The algorithm described here reads four cells from the submap gridmap instead (blue squares in $\mathbf{p}_b^{\mathcal{W}}$) and fuses them into a single cell in the output gridmap (gray square). Note that typical submaps are 400×400 cells, and the typical output gridmap is 5000×5000 cells. Cell occupancy values are not shown for clarity.

global gridmap rasterization by avoiding the fusion of redundant information (their work is compared in Section 7.4.6). As far as I am aware no other researchers have described how to fuse multiple occupancy gridmaps at large scales, or have demonstrated how to handle the misalignment of gridmap cells in a probabilistically sound manner.

5.3.3. Naive Algorithm

To avoid numerical issues and for computational efficiency, it is best to perform occupancy gridmap fusion using the log odds ratio. If $p(\mathrm{m}_i)$ is the probability of the cell m_i being occupied, then its log odds ratio is $l_i = \log \frac{p(\mathrm{m}_i)}{1-p(\mathrm{m}_i)}$. The cells of an unknown gridmap (i.e. $p(\mathrm{m}) = 50\,\%$) in the log odds form begin with all cells initialized to zero. Applying the binary Bayes filter in the log odds form is trivial: cells are fused using *addition*. For example, if there is 95 % certainty that two overlapping cells are occupied ($l_i = 3$ and $l_j = 3$), then the fused log odds value ($l_{i+j} = l_i + l_j = 3 + 3 = 6$) represents a 99.8 % certainty after fusion. Refer to Section 9.2 in [135] for an in-depth explanation.

Given that submap gridmaps are typically misaligned with the output gridmap, as in Figure 5.1, and each submap cell overlaps multiple output cells, the log odds value for each submap cell must be divided and fused to multiple output cells. One technique is to use a reverse form of bilinear filtering [336]. Using this approach, each submap cell value is divided and fused to its the four closest output cells (the log odds value is weighted by the submap cell's bilinear distance to the four output cells). While the math used for cell fusion is trivial, as the problem scales to thousands of submaps and large output gridmaps (e.g. 5000×5000), the memory access patterns in both CPUs and GPUs can become inefficient[2].

The naive approach to gridmap fusion is a simple nested loop: for each submap, iterate over all of its cells, transforming them into the output gridmap's coordinates and fusing them into the four nearest output cells. For each submap this approach requires the overlapped output cells to be read from memory, modified and then written back to memory four times— eight times where the pair of submaps overlap in Figure 5.1, and on average 272 times in the datasets used in this research. While this naive approach can be parallelized by fusing gridmap cells in multiple threads of execution at the same time, it is difficult and expensive to synchronize memory reads and writes to the output gridmap. The fusion operation must complete atomically, that is, two threads cannot read, modify and write to the same output cell at the same time. Heuristics might gain some efficiencies by spatially staggering the cell fusion operations, however the naive approach cannot escape the memory access pattern of reading one submap cell and then reading and writing to four output cells in succession.

5.3.4. Proposed Algorithm

The proposed algorithm flips this pattern up side down: an outer loop iterates over each submap, however for each submap the inner loop iterates over the output gridmap cells that are overlapped by the submap instead (refer to Figure 5.2). For each overlapped cell in the output gridmap the 2-D coordinates of the center of the cell is transformed into the submap's coordinate frame. This submap coordinate is then used to read the corresponding submap cell value, which is then fused into the output cell.

The cell alignment problem is now reversed: each output cell now overlaps with multiple submap cells instead. Bilinear interpolation [336] is used to avoid introducing spatial noise: the weighted sum of the four closest cells in the submap gridmap is fused into each output cell (these four cells are shown in blue in Figure 5.1).

Flipping the inner loop reduces the number of synchronized memory reads and writes to the output buffer by a factor of four. To achieve this it increases the number of reads

[2]For large problems (e.g. 5000×5000 cells), the fast memory caches in modern CPU and GPUs are too small to cache the entire output gridmap (100 MB).

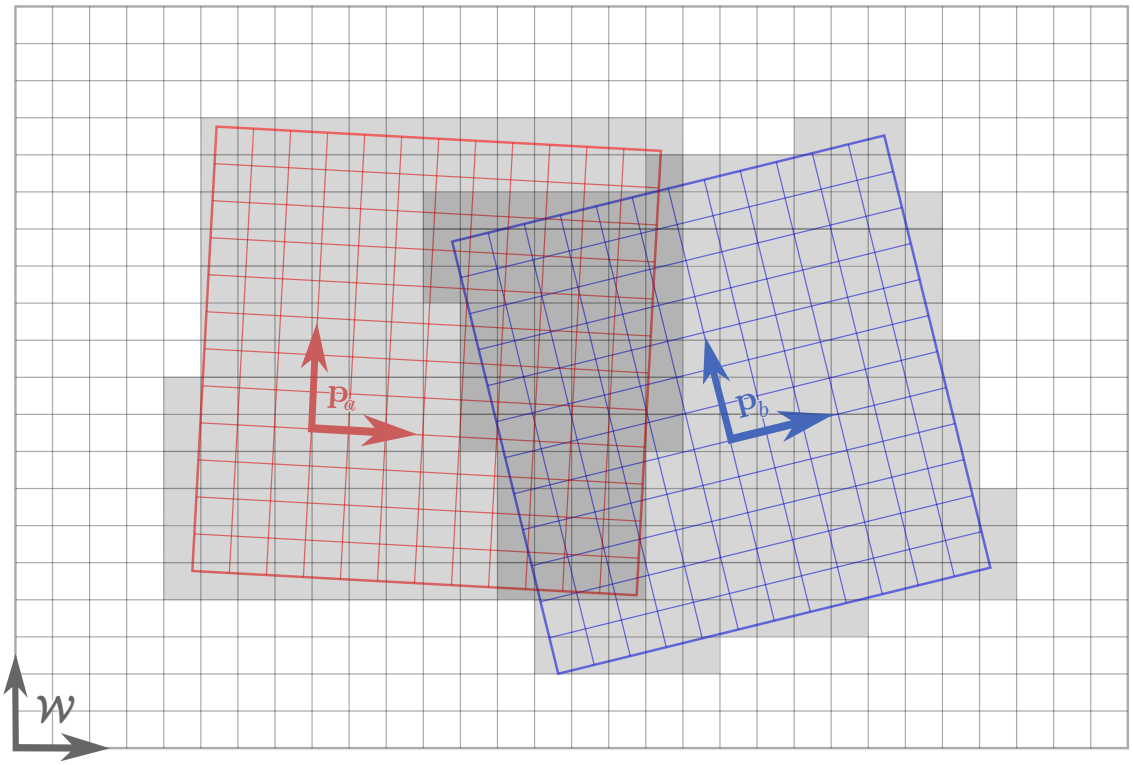


Figure 5.2. Occupancy Gridmap Fusion: Two submap gridmaps with origins at $\mathbf{p}_a^{\mathcal{W}}$ (red grid) and $\mathbf{p}_b^{\mathcal{W}}$ (blue grid) are being fused into the global output gridmap $\mathcal{W}$ (gray grid). The inner loop of the algorithm described here iterates over the output gridmap cells that are covered by the submap gridmaps (gray squares). This is a typical rasterization operation in computer graphics. Cell occupancy values are not shown for clarity.

from the submap gridmap by four times, however this adds negligible cost— reads from immutable submap gridmaps (read-only memory) do not require synchronization, and crucially, the entire submap gridmaps are small enough to fit in the L2 cache in modern GPUs.

5.3.5. Implementation

The proposed gridmap fusion algorithm is very similar to the rasterization that occurs when a GPU is used to draw a rectangular texture. In OpenGL [336], a rectangular texture is rendered into an output Frame Buffer Object (FBO) by drawing a `GL_QUAD` primitive. The corners of the `GL_QUAD` are transformed into FBO output pixel coordinates and a rasterization algorithm iterates over the output cells the `GL_QUAD` overlaps (refer to Figure 5.2). In a modern GPU, this causes thousands of processing cores to calculate and blend pixel values into the output FBO in parallel.

To implement the proposed algorithm in OpenGL, a 5120×5120 pixel FBO is allocated to store the 500×500 meter output gridmap at 0.1 meter resolution[3]. The FBO is

[3]Modern GPU hardware can easily create 16384×16384 pixels FBOs, i.e. 1.6×1.6 km output gridmaps.

```
1  #version 130
2  in vec3           submap_coord;
3  uniform sampler2D submap_tex;
4  void main() {
5          vec4 x = texture2D(submap_tex, gl_TexCoord[0].xy ); //sample gridmap
6          float cell = 20*(0.5 - x.r);   //convert: 10.0=occupied -10.0=free
7          float radius_sqr = dot(submap_coord.xy,submap_coord.xy);
8          float blend = 0.2 + 10.0*exp( -radius_sqr/40.0 ); //Gaussian blend
9          gl_FragColor.r = cell;         //classical gridmap
10         gl_FragColor.g = cell * blend; //blended gridmap
11         gl_FragColor.b = 1.0;          //accumulate submap count
12         gl_FragColor.a = x.g;          //sum of gaussians
13 }
```

Table 5.1. GPU-based Occupancy Gridmap Fusion: algorithm implemented in OpenGL GLSL fragment shader code. For each submap gridmap, this algorithm is executed for each output gridmap cell that is overlapped. Most of the computation in this algorithm is performed in the texture lookup (line 5) and when the output value (`gl_FragColor`) is blended into the output buffer (this is hidden in the OpenGL implementation).

created with four 32-bit floating-point channels (described in the next section), with the red color channel storing the fused output gridmap occupancy values as log odds ratios (initialized to zero, i.e. unknown). Log odds values are fused into the FBO by accumulating them— this is accomplished in OpenGL by setting the *output blending function* to `glBlendFunc(GL_ONE, GL_ONE)` and `glBlendEquation(GL_FUNC_ADD)` [336]. Two FBOs are actually allocated that are used in an alternating manner: the output gridmap is built into one FBO while the other one is being used for other high-level functions (also called double buffering).

Each build cycle starts with the host CPU swapping the output FBOs and configuring the OpenGL projection matrices for 2-D orthographic projection. The algorithm loops quickly over all submaps, instructing the GPU to fuse their submap textures (Section 5.2) using `GL_QUADS` positioned at the current submap pose. The GPU rasterizes each of the submap textures: for each cell in the output gridmap that is overlapped by the submap gridmap, the GPU executes a GLSL *fragment shader* that performs the cell fusions in parallel. The fragment shader code is similar to pseudocode, it is listed in Table 5.1.

For each output cell fusion, the core executing the GLSL code starts by transforming the FBO's output pixel coordinates back into a coordinate in the submap's texture space (line 2). The submap gridmap is sampled by the GPU's texture units; bilinear filtering reads and blends four cells (line 5). The filtered sample is already in log odds format, however it is first remapped because the submap textures are stored as 8-bit unsigned integers (line 6). To build a "classic" gridmap (as in Section 2.3.4.3), the output red channel is simply set to the cell value (line 9). After finishing the GLSL code, the GPU performs the OpenGL output blending function. The blending function accumulates the log odds values in the output FBO with a synchronized read-add-write to the output FBO memory.

Once the GPU has finished fusing all of the submap gridmaps (a call to `glFinish()` returns), the output gridmaps can be downloaded from the GPU or reused in-place for other high-level functions. The probabilities can be recovered from the log odds form with $p(\mathrm{m}_i) = 1 - \frac{l_i}{1+\exp(l_i)}$.

5.3.6. Additional Output Gridmaps

Four color channels can be accumulated in the output FBO, enabling the GLSL fragment shader code in Table 5.1 to implement different gridmap fusion techniques in parallel. The gridmap fusion algorithm is very efficient on modern GPUs, and the extra three channels can be calculated with almost no additional cost.

The log odds occupancy values can be scaled to increase the certainty of gridmap cells that are close to the UGV (and submap origin), while decreasing the certainty of distant gridmap cells. A 2-D Gaussian-like scaling function is used to implement this (line 7 and 8). This scaling function reflects the fact that close lidar returns have less noise, and more certainty, while distant lidar returns are subject to many sources of error (refer to Section 2.3.3). The long tail on this blending function ensures that the distant gridmap cells are included in the output, however the log odds values in these cells are easily overpowered by information fused by a closer UGV. This blending is particularly important to ensure that doorways are not obscured by noisy data from distant UGVs.

The code in Table 5.1 produces output gridmaps with these color channels:

- **Red:** (line 9) accumulated log odds cell values. This builds the "classical" gridmap as described in Section 2.3.4.3.
- **Green:** (line 10) accumulated log odds cell values that have been adjusted by the Gaussian scaling function. This blending algorithm is used in the results presented in Chapter 7 and the accompanying videos.
- **Blue:** (line 11) counter incremented for every submap that overlaps the output cell. This counter is used for diagnostics and statistics.
- **Alpha:** (line 12) accumulated pre-blurred likelihood maps (refer to Section 5.2). This channel emphasizes occupied cells and produces the "x-ray" view shown in Figure 7.15 (e) on page 191.

This GPU-based implementation of the gridmap fusion algorithm is demonstrated in Chapter 7. If height or navigation cost information is available in the submap gridmaps (blue or alpha channels from Section 5.2), it could be fused into these output channels.

5.4. GPU-based Multimodal Constraint Generation

5.4.1. Problem Statement

The problem considered here is how to evaluate and characterize a 3-D likelihood volume that describes the relative spatial alignment of a pair of submap gridmaps in $\mathfrak{se}(2)$. This problem arises when searching for relative constraints, or loop closures, between submaps in Mapbuilder's graph-based SLAM formulation. As explained in Section 4.1.4, each UGV's accumulated odometry error is assumed to be bounded by ± 3 meters and ± 21 degrees. From this, we assume that submaps are roughly aligned and the likelihood volume search can be bounded to the same ± 3 meters and ± 21 degrees around the pair's current relative pose.

The problem has two parts: the first is *evaluating* the likelihood distribution across the search volume, i.e. sampling the likelihood at thousands of (x, y, ϕ) relative poses, the second is *characterizing*, or summarizing the likelihood distribution in a form that can be readily used in a pose graph optimization, e.g. extracting unimodal Gaussian constraints.

The likelihood volume should be sampled with sufficient density to ensure that the resulting constraint distribution is representative. In complex environments it is likely that multiple potential alignments exist between submap pairs (for example Figure 5.3), and insufficient sampling could miss the correct alignment. Chapter 6 describes a novel technique that optimizes multimodal Gaussian distributions, thus this chapter considers the problem of characterizing complex likelihood volumes with multimodal Gaussian constraints also.

5.4.2. Previous Work

The constraint likelihood volume could be evaluated using existing scan matching approaches such as ICP (Section 2.3.4.1). Given enough uniformly distributed samples the peaks in the distribution could be found and their covariances estimated [326]. Approaches like ICP that use hill climbing tend to get stuck in local minima, however, and it is difficult to know *a priori* how dense the sampling should be. It is difficult to obtain accurate (non-conservative) covariance estimates with iterative approaches, without sampling large volumes around each peak.

While exhaustive uniform sampling of the likelihood volume is computationally expensive, it *will* find each peak given a sufficiently small sampling interval. Many authors have proposed using exhaustive correlation-based approaches in $\mathbb{SE}(2)$ SLAM. Konolige and Chou describe CPU-based approaches as early as 1999 [279]. Bailey describes scan correlation experiments in his thesis [142], while others describe correlation using histograms in [281, 280, 337]. An example of exhaustive correlation-based matching is shown in Figure 5.3.

Olson described the first GPU-based exhaustive correlation approach in [282], which he demonstrated to be superior to both ICP and ICL algorithms (Section 2.3.4.1). Olson used OpenGL GLSL shaders to perform correlative lidar scan matching. His approach matched a 1-D texture representing a single lidar scan to a 2-D texture that represented the gridmap in log-likelihood form. Olson also showed how covariances could be extracted from the likelihood volume around a peak, to produce unimodal constraint distributions.

An early version of this research work was the first to describe exhaustive correlation of gridmaps at large scales using a GPU-based algorithm [66], however it only extracted unimodal constraint distributions. Since then Rodriguez-Losada et al. demonstrated a local submap matching approach using GPUs [329], however instead of extracting Gaussian constraints, they blur the submap gridmaps and perform gradient descent directly on a correlation cost function. They have demonstrated their approach directly aligning 28 submaps in 1.23 seconds.

The GPU-based algorithms described by Olson [282] and Rodriguez-Losada et al. [329] are closely related to this work, however they solve a different problem. I could not find any publications describing exhaustive correlative submap matching using GPUs, or how to extract multimodal Gaussian constraint distributions from likelihood volumes.

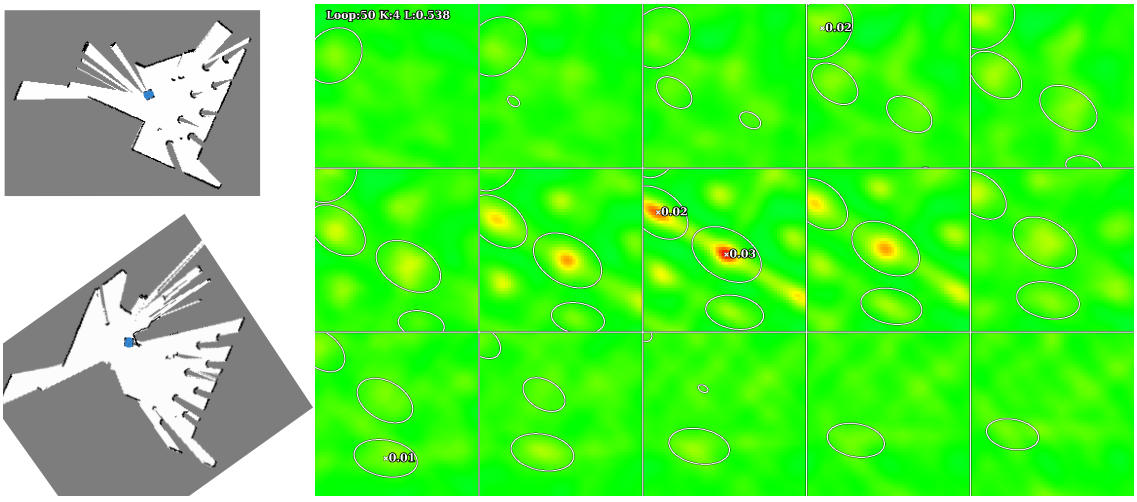


Figure 5.3. Multimodal constraint output: example with perceptual aliasing. Correlation results for an area in the Old Ram Shed Challenge where repetitive geometry causes perceptual aliasing. An overlapping pair of submaps is shown (left), along with 15 slices through their $(x, y, \phi) \in \mathfrak{se}(2)$ constraint likelihood volume (right). Each slice represents the 2-D correlation between the submaps varied over $\pm 3\,\mathrm{m}$ translation in x and y, at a fixed angular rotation, e.g. the middle row corresponds to -6, -3, 0, 3 and 6 degrees. A Gaussian mixture has been fit to the likelihood volume, modes are indicated by their 3-σ covariances (2-D ellipses). Due to occlusions, only a small fraction of the occupied cells (black) in this submap pair actually overlap. An array of columns in the environment dominate the matchers output.

5.4.3. Multimodal Constraint Generation

5.4.3.1. Evaluating the Likelihood Volume

If the pair of submaps being evaluated have global poses $\mathbf{p}_a^{\mathcal{W}}$ and $\mathbf{p}_b^{\mathcal{W}}$, the current relative $\mathfrak{se}(2)$ transform between them is $\mathbf{q}_b^a = \ominus\mathbf{p}_a^{\mathcal{W}} \oplus \mathbf{p}_b^{\mathcal{W}}$. The likelihood volume is evaluated around this relative pose, i.e. $\mathbf{q}_b^a + [x, y, \phi]^T$, where the translational parameters x and y are both varied ± 3 meters and the angular parameter ϕ is varied ± 21 degrees. The sampling parameters $[x, y, \phi]^T$ define a 3 DOF volume in $\mathfrak{se}(2)$. For each sample, the algorithm evaluates the 2-D correlation between the two gridmaps with the relative transform $\mathbf{q}_b^a + [x, y, \phi]^T$. The likelihood volume is sampled at discretized intervals, or *steps*, in the x, y and ϕ parameter space.

The step sizes are first calculated based on the gridmap's spatial resolution: The 2-D correlations calculated for each sample cannot use the occupancy gridmap values (they are log odds ratios), and likelihood cost gridmaps are used instead (Section 5.2). The likelihood cost gridmaps are created by applying a 2-D Gaussian blur with $\sigma = 0.2$ meters to each of the occupied cells. While this blurring approximates some of the lidar sensor's noise, it also helps to smooth the resulting likelihood volume which is advantageous because it allows the step size between correlation samples to be increased. With a blur of $\sigma = 0.2$ meters, the smallest peak in the likelihood volume has $\sigma = 0.26$ meters (correlation between two 2-D Gaussians), which requires a linear step size of 0.1 meters to sample adequately.

The angular step size is more difficult to determine in a principled manner, since it depends on the shape of the environment and on the amount of blurring, σ. Even a small rotation will move distant grid cells through a large arc. A step size of 3 degrees is chosen in this work, since a single rotation step will move a cluster of cells that are 20 meters from the origin through a one meter arc. This motion is small enough for the cluster of cells to be aligned by the ± 3 meter linear translation across five angular steps. For larger objects that are closer to the submap's origin, the same 3 degree step combined with the $\sigma = 0.2$ meter blurring allows objects up to eight meters long to be aligned.

To evaluate the likelihood volume on a GPU, the samples (each one a 2-D submap correlation evaluated at $\mathbf{q}_b^a + [x, y, \phi]^T$) are calculated in a OpenGL GLSL algorithm executed in parallel by thousands of GPU cores. To formulate the problem, the submap gridmaps are loaded into 2-D textures as described in Section 5.2, while the output likelihood volume is created as a floating-point output FBO. To represent the 3-D likelihood volume in the 2-D output FBO, the volume is sliced into fifteen 2-D tiles. Each 60×60 pixel tile captures the ± 3 meter translation in the x and y directions (0.1 meter steps) for a fixed angle ϕ. The fifteen tiles each evaluate the range of translations at a different angle ϕ,

varied ±21 degrees in 3 degree steps. Figure 5.4 shows an example of two submaps and their likelihood volume as calculated in the output FBO.

To start the GPU cores performing the correlations for each sample in the likelihood volume, fifteen `GL_QUADS` are rendered into the output FBO for each of the fifteen angular offsets. Using the OpenGL model projection matrix, the input coordinate for each 60×60 pixel `GL_QUAD` is translated to the center of the current set of samples ($\mathbf{q}_b^a + [0, 0, \phi]^T$). OpenGL rasterizes each precisely aligned `GL_QUAD`, iterating over the 60×60 pixels/samples (x and y translations), starting a single GPU core processing each sample.

For each sample, a GLSL fragment shader performs the 2-D submap correlation algorithm listed in Table 5.2, between the "target" submap (line 4) and the "base" submap (line 3). The algorithm uses a double for loop (lines 20 and 23) to iterate over every cell in the base submap's gridmap. It calculates the base gridmap's cell coordinate in the target gridmap's frame (line 26), and then reads the target gridmap cell value (line 29), sampling it with bilinear filtering [336] to avoid introducing spatial noise (similar to Section 5.3.5). The correlation is performed with a multiplication and accumulation of the two cells' likelihood values (line 30).

The correlation sum is normalized before being saved as the sample likelihood value in the output FBO (line 37). This normalization is a unique aspect of the algorithm that is designed to reduce "greedy" matching. Typical correlation approaches are greedy because they return higher correlation values as more geometry overlaps; this tends to

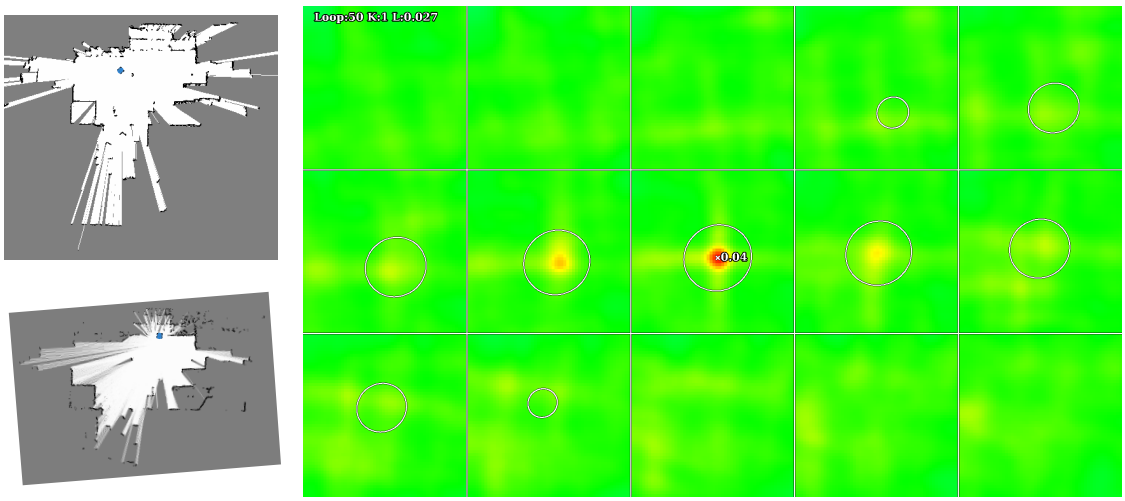


Figure 5.4. Multimodal constraint output: example with a single mode. Correlation results for a well-constrained area in the Old Ram Shed Challenge. An overlapping pair of submaps is shown (left), along with 15 slices through their $(x, y, \phi) \in \mathfrak{se}(2)$ constraint likelihood volume (right). Each slice represents the 2-D correlation between the submaps varied over ±3 m translation in x and y, at a fixed angular rotation, e.g. the middle row corresponds to -6, -3, 0, 3 and 6 degrees. These submaps are from different UGVs (TM's top left, Penn's lower left). This example shows how a single Gaussian mode appears when its 3-σ covariance ellipsoid is plotted as 2-D slices (white ellipses).

```
1  #version 130
2  in vec3 rel;
3  uniform sampler2D base_submap_tex;
4  uniform sampler2D target_submap_tex;
5  uniform vec2 base_bottom_left;
6  uniform vec2 base_top_right;
7  uniform vec2 target_bottom_left;
8  uniform vec2 target_top_right;
9  void main() {
10     const float cell = 0.1;  //cell size in meters
11     const float half_cell = cell/2.0;
12     const float scale = 2.0*3.1415 * 2.0 * (2.0*2.0) * cell*cell * 100.0;
13     vec2 base_tex_coord, target_tex_coord, offset;
14     vec2 inv_base_extents   = 1.0 / (base_top_right - base_bottom_left);
15     vec2 inv_target_extents = 1.0 / (target_top_right - target_bottom_left);
16     vec2 rel_adj = vec2( rel.x-half_cell, rel.y-half_cell ); //offset output
17     mat2 rot_inv = mat2(cos(rel.z), -sin(rel.z), sin(rel.z), cos(rel.z)); //inverse
18     float correlation_sum = 0.0;
19     int overlap_count = 0;
20     for (float y=base_bottom_left.y+half_cell; y<base_top_right.y; y+=cell) {
21         base_tex_coord.y = (y - base_bottom_left.y) * inv_base_extents.y;
22         offset.y = y - rel_adj.y;
23         for (float x=base_bottom_left.x+half_cell; x<base_top_right.x; x+=cell) {
24             base_tex_coord.x = (x - base_bottom_left.x) * inv_base_extents.x;
25             offset.x = x - rel_adj.x;
26             target_tex_coord = ((rot_inv * offset) - target_bottom_left)
27                                                    * inv_target_extents;
28             vec4 base   = texture2D(base_submap_tex,   base_tex_coord);
29             vec4 target = texture2D(target_submap_tex, target_tex_coord);
30             correlation_sum += base.g * target.g;
31             if (   ( target.r < 0.25 && ( base.r   > 0.75 || base.g   > 0.0 )) ||
32                    ( base.r   < 0.25 && ( target.r > 0.75 || target.g > 0.0 ))   )
33                 overlap_count++;
34         }
35     }
36     if (overlap_count > 10)
37         gl_FragColor.a = exp(scale * correlation_sum / float(overlap_count));
38     else
39         gl_FragColor.a = 0.0;
40 }
```

Table 5.2. GPU-based Multimodal Constraint Generation: correlation algorithm implemented in OpenGL GLSL fragment shader code. This 2-D correlation algorithm executes on thousands of GPU cores in parallel. Each core evaluates one potential alignment of a pair of submaps at a time. An optimized (but less readable) version of this algorithm uses bounding boxes to increase the correlation speed by about 20 %.

bias the correlation results to prefer maximum overlap, while ignoring potentially correct alignments of finer geometry in the environment (e.g. the corridor cases described in Appendix B). Incurring minimal additional cost, the GPU counts how many free or occupied cells overlap between the two submaps, while ignoring unknown cells (lines 31-33). The overlapping cell count is used to normalize the correlation sum (line 37).

Using this algorithm and the GPU used throughout this research work, over 1,600 correlation samples can be calculated in parallel. For the real-world datasets in Chapter 7, this equates to about 1.2 million correlation samples per second, or calculating about 22 submap pair likelihood volumes per second.

5.4.3.2. Extracting Multimodal Gaussian Constraints

Both Olson's approach [282], and earlier versions of this research work [66], fit unimodal Gaussian distributions to likelihood volumes. In simple environments (such as Figure 5.4) this is adequate, however in more complex environments (such as Figure 5.3) it could be premature to reduce the likelihood volume to a single Gaussian by selecting a single peak. Other examples of complex likelihood volumes are shown in Figure 6.3 and Figure 6.4 on page 156.

To represent the likelihood volume's distribution more accurately, a Gaussian mixture is fit using the Expectation Maximization (EM) algorithm. EM is an iterative approach that is well-suited to extracting Gaussian modes, for a thorough description refer to Section 9.2.2 of [100].

Observing that many likelihood volumes have a non-zero bias around Gaussian-like peaks (for example Figure 6.1 on page 152), two modifications to the standard EM algorithm enable these peaks to be extracted from cluttered likelihood volumes. The first modification is to truncate the Gaussians to a 2-σ bounding ellipsoid when evaluating responsibilities (see [100]). The second is the addition of a uniform ellipsoidal volume to each Gaussian (coincident, and sized to 3-σ). It is difficult to visualize the resulting 3-D scalar volumes: if the same truncation was performed to a 1-D Gaussian, the result would look like a bowler hat.

The EM algorithm terminates after either convergence is achieved, or 50 iterations. The resulting Gaussian modes are sorted by weight and the dominant six modes are used to create a multimodal Gaussian constraint. This multimodal constraint generation approach is demonstrated in Chapter 7.

6 Robust Multimodal Pose Graph Optimization

Typical pose-graph SLAM techniques are designed to optimize constraints that have *unimodal* Gaussian distributions. Real-world environments, however, often exhibit more complex constraint distributions that are better represented by Gaussian mixtures. In this chapter I introduce Continuous Mode Blending Optimization (COMBO), a novel contribution that allows pose graphs with overlapping *multimodal* Gaussian constraints to be optimized using existing techniques. Section 6.1 motivates this research by describing how multimodal constraints can help prevent outliers and handle perceptual aliasing. Section 6.2 introduces the mathematics for multimodal constraints, while Section 6.3 reviews related work. In Section 6.4 I describe the COMBO technique in detail and provide an analysis of its convergence properties.

6.1. Introduction

In real-world environments, cluttered or repetitive geometry can cause *perceptual aliasing* ambiguities when viewed by sensors that are unable to perform data associations directly (Section 2.1.1). Lidar scanners, for example, in a room with parallel and self-similar structures such as Figure 6.1, produce complex multimodal data associations that are troublesome to integrate into standard SLAM algorithms.

The Mapbuilder multi-robot SLAM system, described in Chapter 4, builds large and complex pose graphs that represent the environment with overlapping submap gridmaps. When matching and aligning pairs of these submaps, perceptual aliasing ambiguities

create constraints with complex likelihood distributions. Chapter 5 describes how these likelihood distributions can be efficiently evaluated and approximate with multimodal Gaussians.

6.1.1. Research Contributions

Two contributions are described in this chapter that show how multimodal Gaussian constraints can be used in traditional pose-graph SLAM:

1. **Robust multimodal constraints:** enable loop closures to be preemptively added to the pose graph and outliers to rejected by consensus (Section 6.4.6). They also describe more complex spatial relationships between submaps, ideal for real-world environments with perceptual aliasing (Section 6.4.3).

2. **Multimodal pose graph optimization:** the continuous mode blending optimization (COMBO) technique enables complex multimodal Gaussian constraints to be optimized using traditional nonlinear least-squares approaches (Section 6.4.1).

To the best of my knowledge, the ability to optimize pose graphs with complex multimodal constraints has not been described previously in the literature. The COMBO technique exhibits convergence properties that are representative of the underlying multimodal constraint distributions. In Chapter 7, large-scale results in $\mathbb{SE}(2)$ are demonstrated with both the MAGIC challenge datasets and simulations (to provide ground truth).

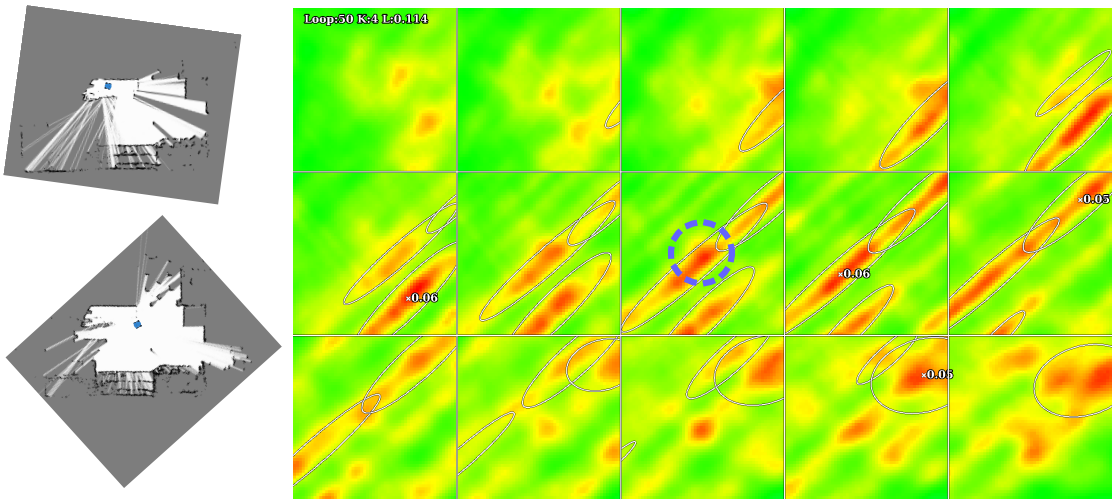


Figure 6.1. Multimodal constraint output: complex overlapping modes. Correlation results for an area in the Old Ram Shed Challenge. An overlapping pair of submaps is shown (left), along with 15 slices through their $(x, y, \phi) \in \mathfrak{se}(2)$ constraint likelihood volume (right). Each slice represents the 2-D correlation between the submaps varied over ± 3 m translation in x and y, at a fixed angular rotation, e.g. the middle row corresponds to -6, -3, 0, 3 and 6 degrees. A Gaussian mixture has been fit to the likelihood volume, modes are indicated by their 3-σ covariances (2-D ellipses). This example shows a complex likelihood volume with no dominant modes. The correct alignment is circled (blue), which is actually 0.55 m away from the highest peak.

6.1.2. Motivation

6.1.2.1. Avoiding Constraint Outliers

Outlier constraints typically occur in environments that are prone to perceptual aliasing, such as corridors or hallways with repetitive features. Figure 6.1 shows the output of an exhaustive 2-D correlation, or a *likelihood volume* (see Section 5.4), between a pair of submaps that exhibit perceptual aliasing. Likelihood volumes like these are difficult to simplify to unimodal distributions, since the correct alignment is not necessarily the highest peak in the likelihood volume. Figure 6.1 has multiple peaks, and the correct alignment is actually the third highest peak. Typical approaches would simplify likelihood volume into a unimodal constraint that is an outlier.

The unimodal constraints used in typical $\mathbb{SE}(2)$ pose-graph SLAM formulations produce residual errors that exhibit a *quadratic* influence during nonlinear least-squares maximum likelihood estimation (MLE). A single outlier constraint, therefore, will have a disproportionate effect and can prevent convergence to the correct solution. Pose graph implementations typically adopt one of two approaches: they either try to avoid outliers, or they to aim to make MLE robust against them.

In an attempt to avoid outlier constraints many SLAM researchers over the last decade have described robust SLAM front-ends[1] [144, 142, 248, 253, 53]. These approaches aim to validate loop-closure hypotheses to high confidence intervals (e.g. 3-σ) before adding them to the graph. This often result in robots driving many times further than their sensor range before constraint hypotheses can be accepted. These delays might be tolerable in single-robot SLAM, however in MR-SLAM they are very undesirable, particularly when multiple robots are operating in close proximity.

To avoid loop closure delays, a number of researchers have investigated shifting the requirement for robustness onto the SLAM back-end instead. Reviewed in Section 6.3, these approaches lower the SLAM front-end's validation thresholds and add constraints immediately— accepting the risk that some will be outliers [253, 250, 256, 262, 338, 301, 339, 117, 57, 277, 299]. These approaches typically use unimodal constraints, and unless additional information is available, complex constraints like the one in Figure 6.1 will be simplified into outliers.

In [117] Pfingsthorn and Birk explain how odometry-based priors allow this unimodal constraint simplification to work well for *single-robot* pose graphs. Odometry measurements will typically help to pick the correct mode from a complex likelihood volume. In *multi-robot* pose graphs, however, reliable priors are generally not available between pairs of robots, and choosing a single mode from a complex likelihood volume is more

[1] The separation into front-end and back-end roles is described in Section 3.1.

likely to produce unimodal constraints that are outliers. Figure 6.2, for example, shows two submaps from MR-SLAM that exhibit likelihood volumes with two equally likely alignments.

These problems motivate the use of multimodal Gaussian constraints to avoid outliers, particularly in the case of MR-SLAM. Growing interest by the research community is indicated by workshops such as the ICRA 2013 "Workshop on Robust and Multimodal Inference in Factor Graphs".

6.1.2.2. Handling Perceptual Aliasing

Perceptual aliasing is a recurring problem in the datasets used in this research work. It occurs when the UGVs are in sparse areas or long corridors: situations where parts of each lidar scan are not returned (some parts of the environment are too distant). In degenerate cases like these, the observable parts of the environment are unable to constrain uncertainty in the robot's pose. For example, Figure 6.3 shows two submaps from two robots driving together down a straight and relatively featureless alleyway that is 20 meters wide. Here parallel walls provide minimal information to constrain the robot's pose and odometry errors quickly accumulate in the direction of travel. In this example, the constraint likelihood volume does not exhibit any Gaussian-like peaks, rather it has a ridge of constant likelihood that runs the width of the matching volume. In MR-SLAM, without odometry or priors, every point along this ridge is equally likely.

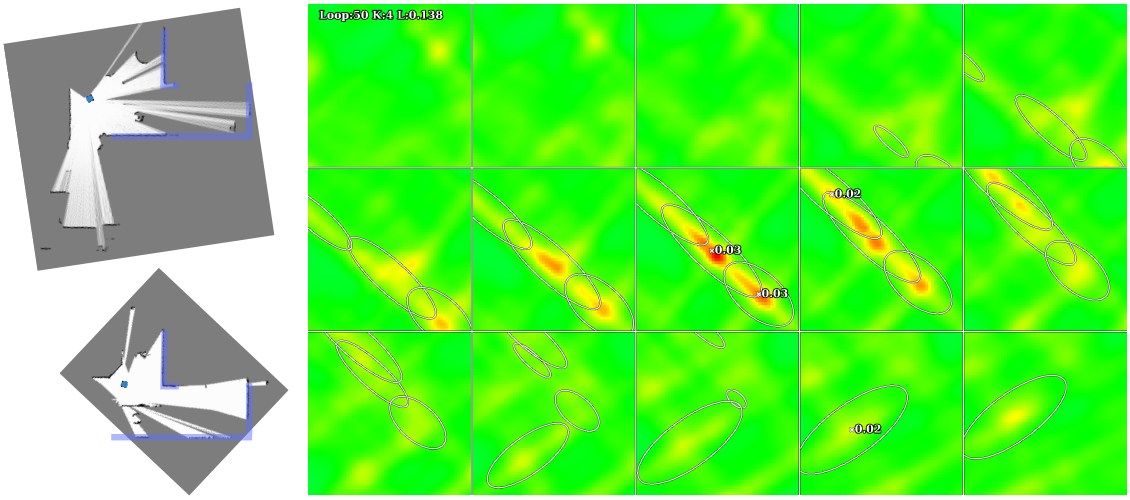


Figure 6.2. Multimodal constraint output: overlapping modes. Correlation results for an area in the Old Ram Shed Challenge. An overlapping pair of submaps is shown (left), along with 15 slices through their $(x, y, \phi) \in \mathfrak{se}(2)$ constraint likelihood volume (right). Each slice represents the 2-D correlation between the submaps varied over ± 3 m translation in x and y, at a fixed angular rotation, e.g. the middle row corresponds to -6, -3, 0, 3 and 6 degrees. A Gaussian mixture has been fit to the likelihood volume, modes are indicated by their 3-σ covariances (2-D ellipses). Occlusions in the environment have produced submaps with minimal overlapping geometry (highlighted in blue), which produces a complex likelihood volume with no dominant modes.

Featureless alleyways like this *do* provide valuable information, however only in the direction perpendicular to the walls. It is difficult to capture this information in the form of submap constraints, however. While a single Gaussian could fit the ridge from the alleyway in Figure 6.3, its eccentricity is likely to become very large and its inverse covariance would be almost rank deficient[2]. In nonlinear least squares, this rank deficiency creates a gauge freedom [234], and any optimization will either become numerically unstable or fail to converge. One way to avoid this rank deficiency, and ensure convergence, is to deliberately underestimate the covariance so that the eccentricity is reduced. This adds an arbitrary and erroneous bias to the optimization, however, since the correct location of the constraint mode is unknown. One research problem that arises here is how to incorporate constraint information from these degenerate configurations into a well-formed pose graph optimization, while not adding any biases to the constraints.

One solution is to use multimodal constraints with smaller modes that can *overlap*. Chapter 5 describes how to evaluate likelihood volumes like these and approximate them with Gaussian mixtures. For the example in Figure 6.3, four overlapping modes can accurately represent this flat ridge. When visualized as overlapping 3-D ellipsoids, these four modes sum together to form a smooth-sided cylindrical volume, similar to a French baguette. Figure 6.6 plots a 2-D slice through a similar volume, showing how the Gaus-

[2] Each $\mathbb{SE}(2)$ constraint has three degrees of freedom, in this situation only two of the three are constrained.

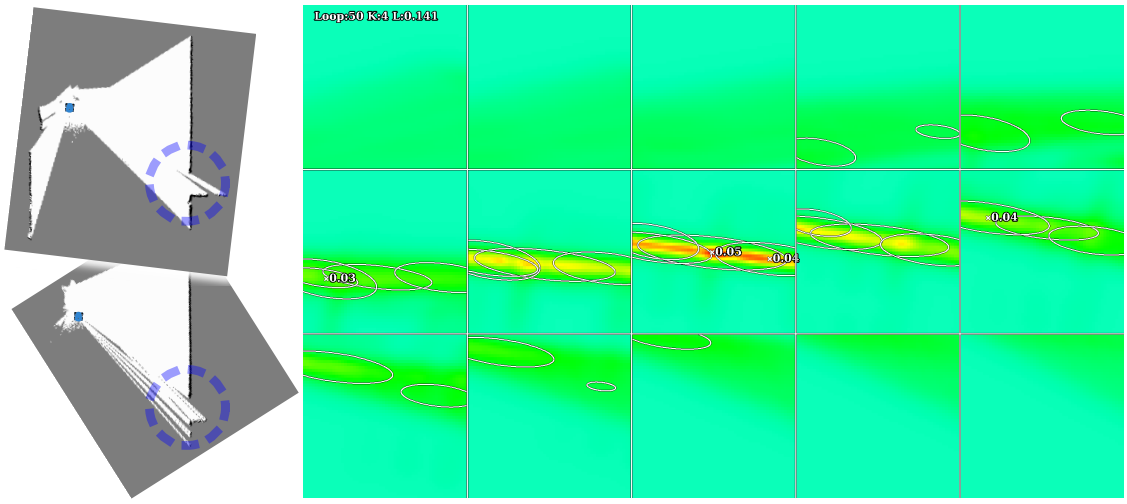


Figure 6.3. Multimodal constraint output: perceptual aliasing ambiguities. Correlation results for a 20 m wide alleyway in Phase 2. The overlapping pair of submaps have been rotated so both robots are driving down the page (left), while 15 slices through the $(x, y, \phi) \in \mathfrak{se}(2)$ constraint likelihood volume are shown (right). Each slice represents the 2-D correlation between the submaps varied over ± 3 m translation in x and y, at a fixed angular rotation, e.g. the middle row corresponds to -6, -3, 0, 3 and 6 degrees. A Gaussian mixture has been fit to the likelihood volume, modes are indicated by their 3-σ covariances (2-D ellipses). The only identifiable feature in the environment is circled (blue). The smoothness of the top of this ridge is a good example of the unique normalization approach described in Section 5.3. A typical "greedy" correlation approach would produces a peaked ridge that a unimodal Gaussian would fit well, however incorrectly.

sian components sum to produce a smooth surface. Each of these modes have minimal eccentricities, avoiding the risk of rank deficiencies. The cost surface for this multimodal constraint is both locally smooth, and approximately "flat" for the entire width of the matching volume. While the equivalent unimodal constraint would add an erroneous bias to the optimization, the multimodal constraint avoids this: it correctly models the fact that the lidar scanner has provided no information in the direction parallel to the walls.

Using the Mapbuilder MR-SLAM system (Chapter 4), the two robot's in Figure 6.3 would create a pose graph with about 80 constraints while driving down this alleyway. Each constraint provides minimal information in the direction parallel to the alleyway, however using multimodal constraints the dominant walls can still be aligned, while allowing groups of submaps to be freely extended or contracted like a telescope (the telescoping can occur with minimal changes to the constraint cost). Ones these robots reach the end of this alleyway, the orthogonal walls shown in Figure 6.4 result in a single dominant mode with a circular covariance: this constraint aligns the two submaps very precisely. Successive submap matches in this area further constrains the pose graph and eventually the optimization aligns the submaps to automatically correct the drift. For reference, the global gridmaps from this situation are shown in Figure 7.15 (b) on page 191.

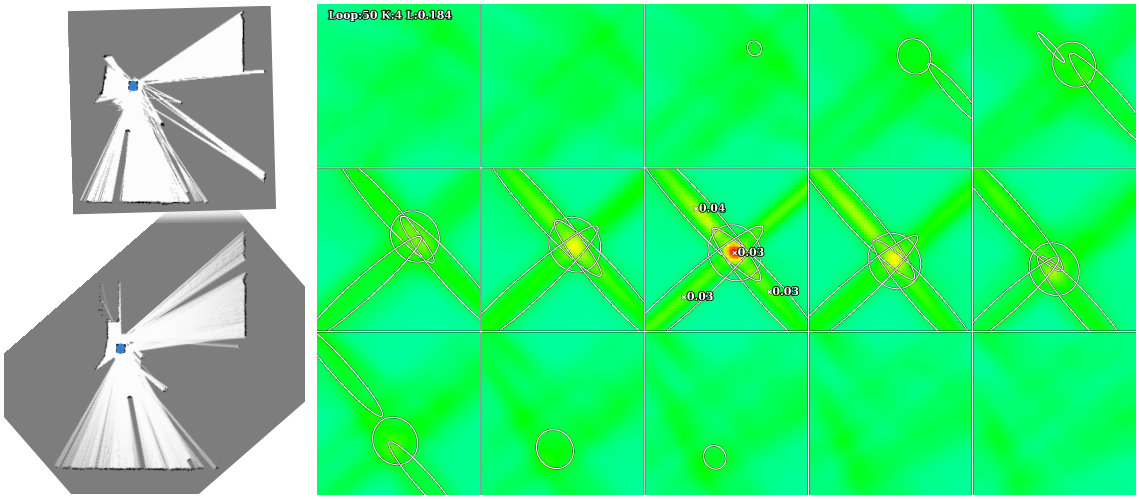


Figure 6.4. Multimodal constraint output: perceptual aliasing resolved. Correlation results for a 20 m wide alleyway in Phase 2. The overlapping pair of submaps have been rotated so both robots are driving down the page (left), while 15 slices through the $(x, y, \phi) \in \mathfrak{se}(2)$ constraint likelihood volume are shown (right). Each slice represents the 2-D correlation between the submaps varied over ± 3 m translation in x and y, at a fixed angular rotation, e.g. the middle row corresponds to -6, -3, 0, 3 and 6 degrees. A Gaussian mixture has been fit to the likelihood volume, modes are indicated by their 3-σ covariances (2-D ellipses). The aliasing in Figure 6.3 is resolved when the robots observe the end of the corridor. The circular mode indicates the correct alignment between the submaps, while the more eccentric modes identify incorrect alignments between the orthogonal walls.

6.1.3. Problem Statement

Pose graphs built with *overlapping* multimodal constraints are an expressive and accurate way to perform SLAM in complex real-world environments. Traditional MLE approaches, however, cannot be used to optimize them and the multimodal techniques described in the literature (reviewed in Section 6.3.3) can only optimize discrete modes. This problem motivated the development of the COMBO technique in this chapter.

6.2. Background

6.2.1. Unimodal Constraints

This chapter focuses on $\mathbb{SE}(2)$ constraints and their residual error function; these concepts were discussed in detail in Section 2.5.4. Here I adopt a slightly different representation, and derive each $\mathbb{SE}(2)$ constraint, $\mathbf{z}_i$, as a unimodal Gaussian probability density function, $p(\mathbf{z}_i|\mathbf{x})$. The state vector $\mathbf{x}$ for the pose graph has previously been defined as:

$$\mathbf{x} = [\mathbf{r}_0, \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3 \cdots \mathbf{r}_T]^T \tag{6.1}$$

The constraint $\mathbf{z}_i$ is a $\mathbb{SE}(2)$ transform describing the relative alignment between a pair of robot poses, $\mathbf{r}_j$ and $\mathbf{r}_k$. I define a "virtual" sensor model that gives a constraint measurement prediction, $\bar{\mathbf{z}}_i(\mathbf{x})$, from the current robot pose estimates:

$$\bar{\mathbf{z}}_i(\mathbf{x}) \underset{\text{def}}{=} z(\mathbf{r}_j, \mathbf{r}_k) = \ominus \mathbf{r}_j^{\mathcal{W}} \oplus \mathbf{r}_k^{\mathcal{W}} \tag{6.2}$$

For each constraint the residual error can be expressed in $\mathbb{R}^3$:

$$\mathbf{e}_i(\mathbf{x}) = \bar{\mathbf{z}}_i(\mathbf{x}) - \mathbf{z}_i = [x_i, y_i, \phi_i]^T \tag{6.3}$$

Figure 2.19 on page 71 shows this configuration. For unimodal constraints the expected measurement prediction $\bar{\mathbf{z}}_i(\mathbf{x})$ is the normal function with mean μ_i and covariance $\mathbf{\Sigma}_i$:

$$p(\bar{\mathbf{z}}_i|\mathbf{x}) = \mathcal{N}(\bar{\mathbf{z}}_i(\mathbf{x})|\mu_i, \mathbf{\Sigma}_i) = \frac{1}{(2\pi)^{3/2}|\mathbf{\Sigma}_i|^{1/2}} \exp\left(-\frac{1}{2}(\bar{\mathbf{z}}_i(\mathbf{x}) - \mu_i)^T \mathbf{\Sigma}_i^{-1}(\bar{\mathbf{z}}_i(\mathbf{x}) - \mu_i)\right) \tag{6.4}$$

Thus we expect the constraint measurement prediction, $\bar{\mathbf{z}}_i(\mathbf{x})$, based on the current pose estimates, to also have the distribution $\mathcal{N}(\bar{\mathbf{z}}_i(\mathbf{x})|\mu_i, \mathbf{\Sigma}_i)$. We can consider the cost, or Mahalanobis distance, of each unimodal constraint:

$$D_i^2 = \mathbf{e}_i(\mathbf{x})^T \mathbf{\Sigma}_i^{-1} \mathbf{e}_i(\mathbf{x}) = \|\mathbf{e}_i(\mathbf{x})\|_{\mathbf{\Sigma}_i}^2 \tag{6.5}$$

This is expected to exhibit a chi-squared, or $\chi^2(3)$, distribution. Section 2.5.4.7 describes in detail how the minimum cost, or the ML estimate, can be found for a pose graph of unimodal constraints by taking the negative log likelihood. Equation 2.59 shows how this estimation becomes a typical nonlinear least-squares optimization problem.

While the best alignment of two submaps may be apparent to a human, particularly when they are pre-rotated as shown in Figure 6.1, the correlation output often lacks a single dominant mode. Depending on the fitting method, a single Gaussian that is fit to Figure 6.1 would either result in a big "blob", or it would select one of the equally dominant modes (after first subtracting the background clutter). The "blob" constraint is certain to be an outlier, while the single most-dominant mode is likely to be an outlier. In this example the correct mode is the third highest peak and the dominant mode is an outlier.

6.2.2. Multimodal Constraints

The inability of unimodal Gaussians to accurately represent many of the complex constraint distributions encountered in lidar-based SLAM motivates this investigation into Gaussian mixtures. Gaussian mixtures are ideal for describing complex distributions, since in the limit, a mixture of infinite modes can describe *any* distribution [100]. The complex constraint distributions described here can often be simplified into small Gaussian mixtures. These multimodal constraints can be expressed as a weighted sum of M individual Gaussian components:

$$p\left(\mathbf{z}_i|\mathbf{x}\right) = \sum_{m=1}^{M} \omega_m \mathcal{N}(\mathbf{z}_i|\mu_m, \mathbf{\Sigma}_m) \tag{6.6}$$

Where $\sum_{m=1}^{M} \omega_m = 1$ and $\omega_m \geq 0 \, \forall \, m$. Multimodal constraints can also describe unimodal constraints when $M = 1$ and $\omega_1 = 1$.

While multimodal constraints are well-suited to describe real-world SLAM problems, they cannot be optimized using the standard nonlinear least-squares algorithms described in Section 2.5.4.7. Using Equation 2.53 on page 73 the joint distribution over the robot poses $\mathbf{x}$, given the constraint measurements $\mathbf{z}$, becomes:

$$p\left(\mathbf{x}|\mathbf{z}\right) \propto \prod_i p\left(\mathbf{z}_i|\mathbf{x}\right) \tag{6.7}$$

$$= \prod_i \sum_{m=1}^{M} \omega_m \mathcal{N}(\mathbf{z}_i|\mu_m, \mathbf{\Sigma}_m) \tag{6.8}$$

When taking the negative log likelihood of Equation 6.8, the summation prevents the logarithm from being combined with the exponential in $\mathcal{N}(\cdot)$, and thus prevents multimodal pose graphs from being reduced to a least-squares form.

6.3. Previous Work

6.3.1. Robust Loop Closures

With the goal of preventing constraint outliers from causing pose graph optimization from diverging, one line of defense is to evaluate potential constraints in the graph before accepting them permanently. Early approaches involved evaluating large numbers of unimodal constraint cycles by composing $\mathbb{SE}(2)$ transforms and covariances [247]. For valid cycles, the composed transforms should sum to almost zero, i.e. for a three constraint cycle: $\mathbf{z}_i \oplus \mathbf{z}_j \oplus \mathbf{z}_k \approx 0$. A cycle is generally considered valid if the Mahalanobis distance, $\|\mathbf{z}_i \oplus \mathbf{z}_j \oplus \mathbf{z}_k\|^2_{\boldsymbol{\Sigma}_{\mathbf{z}_i \oplus \mathbf{z}_j \oplus \mathbf{z}_k}}$ is below a chosen $\chi^2(3)$ threshold.

In other robust loop closure work, Olson's single-cluster spectral graph partitioning (SCGP) builds a consistency matrix between pairs of unimodal constraint by testing them in small cycles [256]. Latif et al. test unimodal constraint directly with their Realizing, Reversing, Recovering work [263, 340]. In their approach they evaluate potential loop closures in an optimization with χ^2 tests initially, and then in optimizations of smaller clusters of constraints.

6.3.2. Robust Unimodal Constraint Optimization

The effects of a single outlier in a nonlinear least-squares optimization are well known, and robust methods, such as M-estimators, have been described in the estimation literature for several decades. Zhang provides a tutorial in [341]. M-estimators replace the Mahalanobis cost function with a symmetric positive-definite function of the constraint residuals, $\rho(e_i(\mathbf{x}))$. The pose graph optimization problem becomes a minimization of:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \sum_i \rho(e_i(\mathbf{x})) \tag{6.9}$$

This is typically solved as an iteratively re-weighted least-squares (IRLS) problem [341].

Recently Sünderhauf and Protzel described Switchable Constraints (SC), where each constraint is augmented with a switch variable s_i and a switch function $\Psi(s_i)$ [342, 276, 244]:

$$\mathbf{x}^*, \mathbf{s}^* = \underset{\mathbf{x},s}{\operatorname{argmin}} \underbrace{\sum_i \|\Psi(s_i)\, e_i(\mathbf{x})\|^2_{\boldsymbol{\Sigma}_i}}_{\text{Switchable constraint}} + \underbrace{\sum_i \|1 - s_i\|^2_{\boldsymbol{\Xi}_i}}_{\text{Switch prior}} \tag{6.10}$$

The switch variables are initialized to $s_i = 1$, and the switch prior covariances $\boldsymbol{\Xi}_i$ are set to reflect the front-end's confidence in the constraint. During optimization, individual

constraints are slowly switched off ($s_i \to 0$) if the reduction in total cost is offset by the increased cost in the switch prior. They demonstrate SC providing robustness against large numbers of outlier constraints.

More recently Agarwal et al. described Dynamic Covariance Scaling (DCS) [301, 299], which avoids using extra switch variables, thus reducing system complexity and computation cost. Their approach exhibits similarly robust properties by approximating the switch variables with $s_i = \min\left(1, \frac{2\Phi}{\Phi+\chi_i^2}\right)$, where χ_i^2 is the Mahalanobis distance for each constraint, i.e. $\chi_i^2 = \|\mathbf{e}_i(\mathbf{x})\|^2_{\mathbf{\Sigma}_i}$. They demonstrate impressive robustness to constraint outliers.

In [299] Agarwal describes the Geman-McClure robust M-estimator [341], Sünderhauf and Protzel's switchable constraints [342, 276, 244] and their DCS approach [301, 299] as having similar properties. Each of these successful techniques suggest that the SLAM back-end can be made tolerant to front-end designs that output occasional outliers. To ensure convergence, these iterative approaches require the pose graph to be initialized in a reasonably good configuration.

6.3.3. Robust Multimodal Constraint Optimization

The "gold standard" approach to optimizing multimodal pose graphs with *non-overlapping*[3] modes would be to optimize a pose graph of unimodal constraints for *every* possible combination of modes: the lowest cost after all optimization corresponds to the optimal configuration. This is a combinatorial problem, however, that quickly becomes intractable: three poses with three constraints that have three modes each forms 27 unimodal graphs; while four poses with six constraints forms 351 possible graphs. While it is likely that most of the resulting graphs are incorrect, the cost explodes combinatorially. It is for this reason that earlier SLAM work in multi-hypothesis tracking (MHT) has found limited success [260]. While Olson's SCGP was not described as a multimodal method [256], it is a combinatorial approach that is easily adapted to multimodal constraints. I evaluated a variant of SCGP during this research, however observed no notable increase in convergence for cycles of three or four constraints.

Two groups have recently proposed techniques that can optimize multimodal pose graphs at large scale and in real-time. Pfingsthorn and Birk described their Prefilter method [117, 343], while Olson and Agarwal described Max-Mixtures [57, 299]. Both approaches use heuristics to identify and promote a *single* mode from each constraint: at each iteration of MLE they optimize the pose graph with only the promoted modes from each constraint.

[3] Here I consider two component modes of a multimodal constraint as "overlapping" when their 3-σ covariance ellipsoids overlap.

The inability to take the negative log likelihood of Equation 6.8 is the primary reason for this simplification.

In [117], Pfingsthorn and Birk describe an extensive set of experiments with several different approaches for optimizing non-overlapping multimodal constraints. They evaluate particle filters and find them unsuitable, even with 10,000 particles, because of the combinatorial nature of the problem. In another experiment they build a multi-edge graph by inserting separate constraints for each of the modes in the multimodal constraints. They use the Huber robust cost function [341], and report it converging incorrectly to a configuration between modes; this is unsurprising since the Huber cost function approaches the L_1 norm.

Pfingsthorn and Birk report the greatest success with their Prefilter approach, which extracts the least ambiguous spanning tree from the pose graph. They extract the tree by tracing over the pose graph while branching along constraints with the lowest mode count (least ambiguity). Using this initial prefiltered configuration, they optimize the resulting unimodal pose graph with standard MLE techniques. In situations with considerable perceptual aliasing, such as multiple robots in a featureless corridor, the Prefilter is unlikely to choose the correct spanning tree. The Prefilter is not robust, since a unimodal constraint that is an outlier is likely to be selected over a multimodal inlier. Further, the approach does not re-evaluate the promoted modes in each iteration, which means it is even less-likely to converge in the situations with perceptual aliasing described in Section 6.1.2.2.

Olson and Agarwal's Max-Mixtures [57, 299] is the most similar approach to COMBO. Their work was motivated by observations at the MAGIC challenge (Appendix B), specifically the "slip or grip" problem where wheel slippage produced non-Gaussian odometry errors. They augment odometry constraints with an additional mode that describes the probability that the robot slipped in place. To reduce the quadratic effect of outliers, Olson and Agarwal also add a large mode that represents the null hypothesis. They demonstrated Max-Mixtures with multiple modes, however their scan matcher only generated single modes [344]. Their Max-Mixture approach forms multimodal constraint distributions using the $\max(\cdot)$ operator:

$$p\left(\mathbf{z}_i|\mathbf{x}\right) = \max_m \omega_m \mathcal{N}(\mathbf{z}_i|\mu_m, \mathbf{\Sigma}_m) \tag{6.11}$$

In this form, the max operator acts as a selector, returning the single mode with the highest probability, given the current graph configuration. When taking the negative log likelihood, the max operator in Equation 6.11 allows the logarithm to be pushed into the expression so that a typical iterative least-squares problem is formed [57]. They optimize the resulting system with Cholesky decomposition and Gauss-Newton (Section 2.5.4.8). For each iteration, Max-Mixtures simplifies each multimodal constraint into its most likely unimodal constraint.

When using a max-mixture, non-overlapping multimodal distributions (i.e. multiple distinct peaks) will have the same component modes as a proper Gaussian mixture. When multimodal distributions have overlapping modes, however, Olson notes that different component modes will result. Section 7.4.7 consider the effects of these differences.

6.4. Robust Multimodal Pose Graph Optimization

The COMBO technique enables standard MLE techniques to be applied to pose graphs with multimodal constraints by converting each constraint into a unimodal distribution, once per iteration. Instead of selecting and promoting a single mode, as in Max-Mixtures, this approach selects multiple active modes and inexpensively *blends* them into a single mode. The blending is performed using a modified form of the covariance intersection (CI) algorithm described in Section 2.5.3.4 on page 62.

6.4.1. Continuous Mode Blending

The goal is to reduce each multimodal constraint distribution $p\left(\mathbf{z}_i|\mathbf{x}\right)$, into a single weighted Gaussian, i.e. the approximation:

$$\begin{aligned} p\left(\mathbf{z}_i|\mathbf{x}\right) &= \sum_{m=1}^{M} \omega_m \mathcal{N}(\mathbf{z}_i|\mu_m, \mathbf{\Sigma}_m) \\ &\approx \omega_i \mathcal{N}(\mathbf{z}_i|\mu_i, \mathbf{\Sigma}_i) \end{aligned} \tag{6.12}$$

The novel contribution here is to show how the parameters for the unimodal approximation ω_i, μ_i and $\mathbf{\Sigma}_i$ can be calculated using a modified version of CI (Section 2.5.3.4). These modifications extend CI so that it can fuse *multiple* Gaussians instead of pairs. They also addresses CI's efficiency, since mode blending occurs very frequently in the inner-most loops of MLE.

Recognizing that each Gaussian in CI contributes a single term to Equation 2.42, with coefficients β and $(1-\beta)$, the series in these expressions can be extended to fuse M Gaussians with coefficients β_m:

$$\begin{aligned} \mathbf{\Sigma}_i^{-1} &= \beta_1 \mathbf{\Sigma}_1^{-1} + \beta_2 \mathbf{\Sigma}_2^{-1} + \beta_3 \mathbf{\Sigma}_3^{-1} + \cdots + \beta_M \mathbf{\Sigma}_M^{-1} \\ \mu_i &= \mathbf{\Sigma}_i \left(\beta_1 \mathbf{\Sigma}_1^{-1} \mu_1 + \beta_2 \mathbf{\Sigma}_2^{-1} \mu_2 + \beta_3 \mathbf{\Sigma}_3^{-1} \mu_3 + \cdots + \beta_M \mathbf{\Sigma}_M^{-1} \mu_M \right) \end{aligned} \tag{6.13}$$

As with standard CI, the blending coefficients β_m are bounded by the interval $[0, 1]$ and sum to one, i.e $0 \leq \beta_m \leq 1 \forall\, m$ and $\sum_{m=1}^{M} \beta_m = 1$. These blending coefficients are discussed in Section 6.4.2. The same coefficients are used to blend the mode weightings:

$$\omega_i = \beta_1\omega_1 + \beta_2\omega_2 + \beta_3\omega_3 + \cdots + \beta_M\omega_M \tag{6.14}$$

Noting the multiple matrix inversions in Equation 6.13, the CI rules are modified to use the information form, $\mathbf{\Omega}_i = \mathbf{\Sigma}_i^{-1}$ (described in Section 2.5.3.5). Since many MLE implementations use information matrices to store covariances, this modification avoids repeated and expensive matrix inversions with no additional storage cost. The information matrix is related to the canonical form with $\mathbf{\Omega}_i = \mathbf{\Sigma}_i^{-1}$, while the information vector is $\eta_i = \mathbf{\Omega}_i\mu_i$. The M modes in each multimodal constraint are fused into a single weighted Gaussian, $\omega_i\mathcal{N}(\mathbf{z}_i|\mu_i, \mathbf{\Sigma}_i)$, with:

$$\begin{aligned}
\mathbf{\Omega}_i &= \sum_{m=1}^{M} \beta_m\mathbf{\Omega}_m \\
\eta_i &= \sum_{m=1}^{M} \beta_m\eta_m \\
\omega_i &= \sum_{m=1}^{M} \beta_m\omega_m
\end{aligned} \tag{6.15}$$

If necessary, the single weighted Gaussian can be recovered in the canonical form with $\mathbf{\Sigma}_i = \mathbf{\Omega}_i^{-1}$ and $\mu_i = \mathbf{\Omega}_i^{-1}\eta_i$. The mode blending technique in COMBO can be described most simply as the weighted sum of the component modes in their information form. The blended Gaussian maintains the same conservative properties as CI, described in [345], provided the coefficients, β_m, sum to one. Note that while the resulting Gaussian is weighted ($\omega_i > 0$), and not strictly a probability distribution, it is trivial to use in least-squares MLE.

6.4.2. Blending Coefficients

With this principled technique for blending modes, the remaining problem is to assign the blending coefficients, β_m. The blending vector $\beta = [\beta_1, \beta_2, \beta_3 \cdots \beta_M]$ acts as a selector that chooses which modes to combine. For example, a single mode can be selected with $\beta = [\cdots 0, 0, 1, 0, 0 \cdots]$, or two modes can be blended using $\beta = [\cdots 0, \beta_m, 0 \cdots 0, (\beta_m - 1), 0 \cdots]$ (which is standard CI from Equation 2.42).

The blending coefficients are chosen so that the blended mode exhibits MLE convergence properties that are as close to the full multimodal Gaussian distribution of $p(\mathbf{z}_i|\mathbf{x})$ as possible. That is, the cost function should be locally smooth, and the Jacobians should point towards the nearest peak or ridge in $p(\mathbf{z}_i|\mathbf{x})$.

In COMBO the coefficients $\beta = [\beta_1, \beta_2, \beta_3 \cdots \beta_M]$ are determined by sampling each mode at the current estimate using the virtual sensor model from Equation 6.2, i.e.:

$$\beta_m = \frac{\omega_m \mathcal{N}(\bar{\mathbf{z}}_i(\mathbf{x})|\mu_m, \mathbf{\Sigma}_m)}{p(\bar{\mathbf{z}}_i(\mathbf{x})|\mathbf{x})} = \frac{\omega_m \mathcal{N}(\bar{\mathbf{z}}_i(\mathbf{x})|\mu_m, \mathbf{\Sigma}_m)}{\sum_{m=1}^{M} \omega_m \mathcal{N}(\bar{\mathbf{z}}_i(\mathbf{x})|\mu_m, \mathbf{\Sigma}_m)} \tag{6.16}$$

Using Equation 6.16 the coefficients β_m necessarily sum to one, provided the weights $\sum \omega_m = 1$. The intuitive explanation for this assignment is that each mode contributes to the blended Gaussian the same relative amount it contributes to $p(\bar{\mathbf{z}}_i(\mathbf{x})|\mathbf{x})$.

6.4.3. Convergence Properties

This section demonstrates the ideal convergence properties of blended multimodal Gaussians when used in least-squares MLE. Non-overlapping and overlapping modes are the two cases considered here.

Non-overlapping modes: if the current estimate $\bar{\mathbf{z}}_i(\mathbf{x})$ is closest to a single isolated mode m in $p(\mathbf{z}_i|\mathbf{x})$, then this mode will dominate the blended result. The blended parameters $\omega_i \approx \omega_m$, $\mu_i \approx \mu_m$ and $\mathbf{\Sigma}_i \approx \mathbf{\Sigma}_m$ create a basin of attraction around the dominant mode's peak. MLE will converge to this peak as though no other modes were present. A 1-D example, where two isolated modes are separated by 6-σ, is shown in Figure 6.5. At the peak of either mode, the other mode has a negligible blending coefficient ($\beta_m = 10^{-8}$). Using COMBO on this multimodal constraint, the solution converges quadratically to the nearest mode, which is the desired property.

While it is straight-forward to demonstrate convergence near isolated modes, boundary conditions occur between modes. Using Figure 6.5 as an example, a *transitional region* exists between $\mathbf{z}_i = 2$ and $\mathbf{z}_i = 3$ where the blending coefficients are approximately equal, i.e. $\beta_1 \approx \beta_2$. Through this region the blended mode, and its basin of attraction around the mean μ_i, switches from the first mode, μ_1, to the second, μ_2. A single stationary point is created in this transitional region, where the two components are equal, i.e. $\beta_1 = \beta_2 = 0.5$. At this point, the constraint measurement prediction $\bar{\mathbf{z}}_i(\mathbf{x})$ equals the mean μ_i, so that the constraint's residual is zero; this can be seen in Figure 6.5 (b) where the blended mode's χ^2 cost dips to zero. This stationary point between the two modes is expected and desirable, given the shape of $p(\mathbf{z}_i|\mathbf{x})$. In the basin around the stationary point, the constraint exerts very little influence on MLE.

Overlapping modes: the mode blending technique was designed for more complex situations where multiple modes overlap significantly. Figure 6.6 (a) shows a 1-D example where six modes overlap in a way that the individual modes are not separately discernible; this example is modeled on the perceptual aliasing problem that is observed in alleyways, as described in Section 6.1.2.2. The Gaussian mixture $p(\mathbf{z}_i|\mathbf{x})$ exhibits a large flat top,

indicating a region where the constraint has no information and should not influence the convergence of $\mathbf{x}$. This desirable behavior is observed with COMBO in Figure 6.6 (c), where only the edges of the mixture $p(\mathbf{z}_i|\mathbf{x})$ exhibit quadratic convergence.

The flat part of $p(\mathbf{z}_i|\mathbf{x})$ in Figure 6.6 describes a region that is constantly transitioning between modes. Instead of having stationary points, however, Figure 6.6 (b) shows that the blended mode's χ^2 cost is almost zero throughout this region (at 10^{-5} it is not visible in the plot). This indicates that the residual error is approximately zero also, which occurs because the blended mode's mean is tracking the constraint measurement prediction, i.e. $\mu_i \approx \bar{\mathbf{z}}_i(\mathbf{x})$. Overlapping multimodal constraints like these can be chosen so that they exert vary little influence on the MLE. When considering $\mathbb{SE}(2)$ constraints, instead of this 1-D toy problem, however, overlapping modes may be arranged so that they lack influence in only one of the constraint's three degrees of freedom. This desirable trait was described in Section 6.1.2.2.

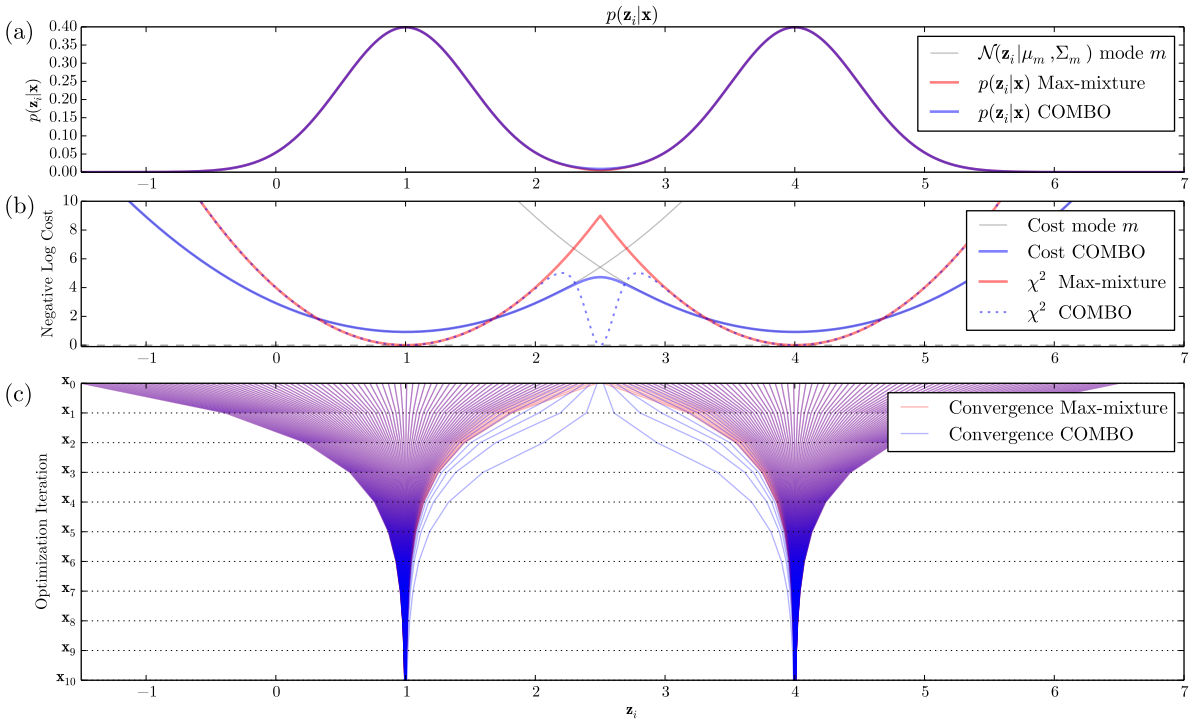


Figure 6.5. COMBO: Convergence Properties of Non-overlapping Modes. A simple 1-D example. (a) $p(\mathbf{z}_i|\mathbf{x})$ for two isolated modes separated by 6-σ. At the peak of the first mode, $\mathbf{z}_i = \mu_1 = 1$, the blending coefficients are $\beta = [0.99999999, 10^{-8}]$ such that that the second mode has negligible influence. (b) shows COMBO's negative-log cost function, which matching both modes' costs in the vicinity of their peaks. To emphasize the smooth convergence (c) shows 10 iterations of gradient descent starting from a wide range of initial values for $\mathbf{x}_0$. Without any external influences the solution converges quickly to the nearest mode. When Levenberg-Marquardt is used, this example converges quadratically. A stationary point exists between the two modes at $\mathbf{z}_i = 2.5$ where the coefficients $\beta = [0.5, 0.5]$ are equal. This is expected, and desirable, given the shape of $p(\mathbf{z}_i|\mathbf{x})$. For distributions with isolated modes, both COMBO and Max-Mixtures converge to the same solution.

6.4.4. Constraint Jacobians

To converge towards the MLE, the Jacobians of the error function, $\frac{\delta e(\mathbf{x},\mathbf{z_i})}{\delta \mathbf{x}}$, evaluated at the current constraint measurement prediction, $\bar{\mathbf{z}}_i(\mathbf{x})$, should point in the direction that is "up-hill" in the distribution $p(\mathbf{z}_i|\mathbf{x})$. There are three ways to calculate these Jacobians:

- **Fast approximation:** Jacobians are approximated using the weighted Gaussian $\omega_i \mathcal{N}(\mathbf{z}_i|\mu_i, \mathbf{\Sigma}_i)$ and the derivations in [243]. This approximation assumes the blended mode's parameters are relatively constant with respect to the blending coefficients β_m. It underestimates the Jacobians in transitional regions, which reduces their influence on MLE, an effect that is desirable in practice.

- **Numerical Jacobians:** the constraint cost function is evaluated 12 times to calculate numerical Jacobians. Each evaluation re-blends the modes to avoid underestimating the Jacobians.

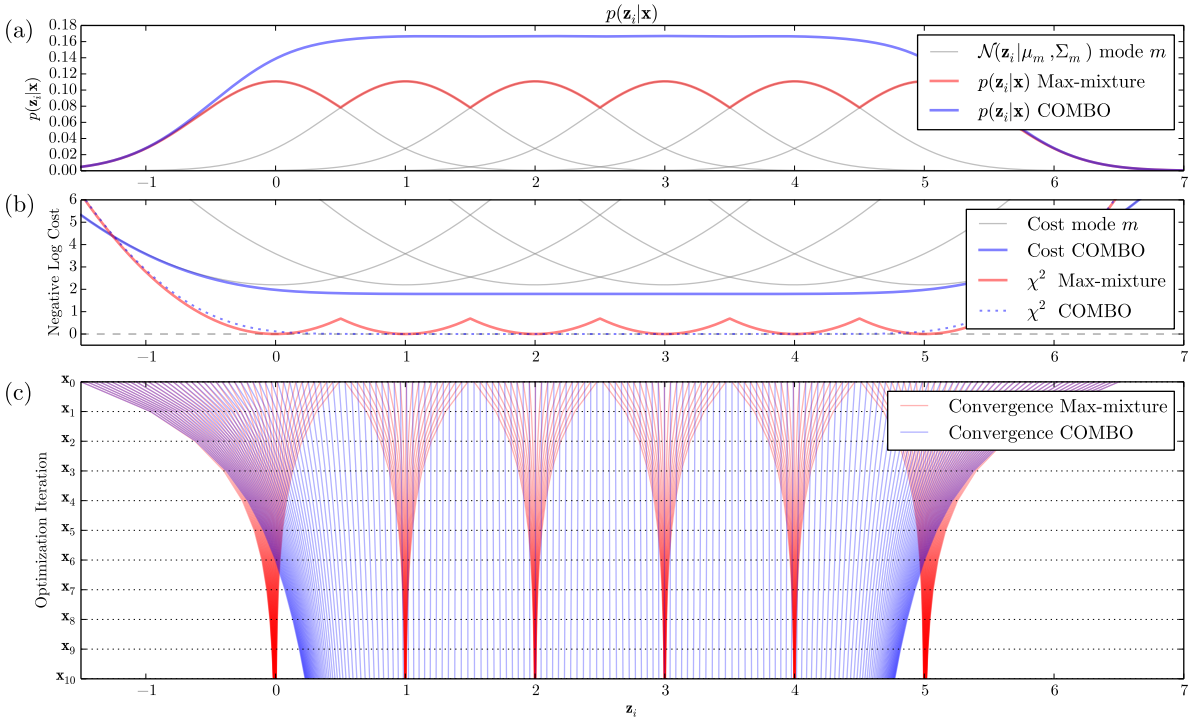


Figure 6.6. COMBO: Convergence Properties of Overlapping Modes. A simple 1-D example in an alleyway with perceptual aliasing. (a) $p(\mathbf{z}_i|\mathbf{x})$ has six overlapping modes separated by 1.67-σ. (b) In the middle of this flat-topped distribution COMBO produces a constant negative log cost indicating that the constraint has no information. (c) shows 10 iterations of gradient descent starting from a wide range of initial values for $\mathbf{x}_0$. For COMBO, the middle of the distribution exhibits no influence over $\mathbf{z}_i$, while the edges exhibit quadratic convergence. This is expected, and desirable, given the shape of $p(\mathbf{z}_i|\mathbf{x})$. COMBO produces a similar convergence when using Levenberg-Marquardt or Gauss-Newton. Max-Mixtures, however, converges quickly to the nearest peak, which is undesirable for overlapping multimodal distributions like this.

- **Hybrid Jacobians:** an optimized implementation that switches between the fast approximation and numerical Jacobians described above. The fast approximation is only used when a single mode m is dominating, i.e. $\beta_m > 0.8$.
- **Analytical Jacobians:** expansions of the derivatives are lengthy, however they can be performed with symbolic math software.

It is difficult to visualize Jacobians to evaluate their correctness. Instead small gradient descent steps are shown in Figure 6.5 (c) and Figure 6.6 (c), which indicate the Jacobian directions across a range of $\bar{\mathbf{z}}_i(\mathbf{x})$. Figure 7.20 shows a more complex and asymmetric 1-D example. In all three cases the Jacobians guide MLE towards the nearest peak in the multimodal distribution $p(\mathbf{z}_i|\mathbf{x})$.

6.4.5. Least Squares Optimization

The COMBO technique produces a single blended mode for each multimodal constraint that is valid for the current measurement prediction $\bar{\mathbf{z}}_i(\mathbf{x})$. The resulting unimodal pose graph can be optimized with any of the standard least-squares techniques described in Section 2.5.4.9.

For complex multimodal pose graphs, Levenberg Marquardt (LM) exhibits good convergence due to its ability to transition between Gauss-Newton and gradient descent. To control this transition, LM requires each constraint to have a *smooth* cost function, $\mathcal{C}_i$. LM typically uses the Mahalanobis distance from Equation 6.5, otherwise known as the χ^2 cost. The χ^2 cost, however, is not defined for multimodal distributions.

While it might seem consistent to directly use the χ^2 cost for each blended Gaussian, the χ^2 cost dips to zero around the stationary points that exist between modes. This causes a problem for LM, which expects the χ^2 cost to decrease when moving in the direction indicated by the Jacobians (i.e. away from the stationary points). This is shown clearly in Figure 7.20 (b), around the stationary point at $\mathbf{z}_i = 2$, where the χ^2 cost increases when moving toward either peaks. In LM, this cost increase would cause the step size λ to be decreased, inhibiting convergence.

Instead of the χ^2 cost, COMBO uses the actual negative log of the multimodal distribution, sampled at the current measurement prediction $\bar{\mathbf{z}}_i(\mathbf{x})$:

$$\begin{aligned} \mathcal{C}_i &= -\log\left(p\left(\bar{\mathbf{z}}_i|\mathbf{x}\right)\right) \\ &= -\log\left(\sum_{m=1}^{M} \omega_m \mathcal{N}(\bar{\mathbf{z}}_i|\mu_m, \mathbf{\Sigma}_m)\right) \end{aligned} \tag{6.17}$$

6.4.6. Robust Multimodal Constraints

Even with the exhaustive correlation approaches used in Mapbuilder (Section 5.4), the multimodal constraints generated will occasionally be outliers, or have outlier modes. A moving object, for example, can break the static-world assumption, adding noise to a submap and outlier modes. This is the case in Figure 7.26 on page 209, where three of four modes are outliers resulting from a person walking past the robot.

Like the robust approaches reviewed in Section 6.3, I adopt a *consensus-based* approach instead of explicitly trying to identify and suppress outliers. This approach requires the MLE to begin in the vicinity of the optimum, while consensus, or internal consistency, invites the inliers to overpower the outliers.

The cost function is made more robust (Section 6.3.2) to reduce the quadratic influence of outliers by clamping it to a fixed upper value, $\mathcal{C}_{max}$. This produces an effect similar to the Geman-McClure or Tukey M-estimators [341], however applied to multiple modes at the same time. Once the cost exceeds the threshold, $\mathcal{C}_i > \mathcal{C}_{max}$, the constraint is switched off by forcing the residual error in Equation 6.3 to zero and setting $\mathcal{C}_i = \mathcal{C}_{max}$.

Using an analogy similar to Section 2.5.4.4, each robust constraint acts like a magnet joining two submaps together. If external forces (other constraints) pull the submaps apart too hard (consensus), the constraint magnet will eventually break its hold ($\mathcal{C}_i > \mathcal{C}_{max}$). The constraint magnet does not exert any forces unless the submaps are brought into close proximity again ($\mathcal{C}_i \leq \mathcal{C}_{max}$).

The value for $\mathcal{C}_{max}$ is selected empirically, since it is not a true χ^2 cost. Each cost, $\mathcal{C}_i$, has a bias that depends on the coefficients in Equation 6.17, in particular the weights ω_m and covariances $\mathbf{\Sigma}_m$. For reference, $\mathcal{C}_{max}$ was set to $\chi^2_{0.99}(3) = 11.34$ for all of the MAGIC challenge datasets in Appendix B.

For implementation efficiency, only the *active set* of modes in each constraint are blended using Equation 6.16 and Equation 6.15. Modes are defined as active when the current measurement prediction $\bar{\mathbf{z}}_i(\mathbf{x})$ is inside their 3-σ ellipse[4]. The active set of modes is filtered using the inexpensive χ^2 test, $\|\bar{\mathbf{z}}_i - \mu_m\|^2_{\mathbf{\Sigma}_m} < \chi^2_{0.997}(3)$. Conveniently, this avoids numerical instabilities in Equation 6.16 when the current measurement prediction is far from any modes. Overlapping and non-overlapping multimodal constraints were described in Section 6.4.3. This definition is formalized here according to whether the individual modes' 3-σ ellipses overlap. By definition, the active set of modes will alway overlap.

[4]The 3-σ ellipses are shown on the likelihood volumes to indicate where modes would be active.

7 Results

This chapter presents results demonstrating my research contributions. Results in Section 7.1 demonstrate on-line hybrid-decentralized and distributed MR-SLAM with a team of WAMbot UGVs. Section 7.2 demonstrates large-scale MR-SLAM in real-time using three combined datasets recorded at the MAGIC challenge. In Section 7.3 I present additional results optimizing multimodal pose graphs using the COMBO technique. Section 7.4 concludes with an in-depth discussion of these results, including the accuracy, scalability and robustness, along with a comparison with other published systems.

7.1. Hybrid-Decentralized and Distributed MR-SLAM

The results presented in this section demonstrate decentralized and distributed MR-SLAM capabilities across multiple UGV and GCS computers using the Mapbuilder system. These results were recorded during *on-line distributed* deployments of the WAMbot UGVs and include typical communications latencies and losses. For reference, the WAMbot MRS architecture is described in Appendix A.

7.1.1. MAGIC Challenge

The MAGIC challenge was introduced in Section 1.3.4.2. The test environment, described in Appendix B, was a 500×500 meter urban environment that included a mixture of indoor and outdoor areas, that were both compact (e.g. small rooms and doorways) and sparse (e.g. car parks and large rooms). Figure 7.1 shows the judge's orthorectified map showing the GCS location and the four challenge phase areas [52]. A small amount of ground-truth survey data is shown also.

Occupancy gridmap results from the on-line deployment of five WAMbot UGVs are shown in Figure 7.2. A combination of hardware and software issues, described in [64], limited our UGVs ability to explore and they were only able to map 15 %, 8 % and 5 % of Phase 1, Phase 2, and Phase 3, respectively. The WAMbot MRS performed considerably better in the Old Ram Shed Challenge (ORSC), exploring 83 % of the area. This limited exploration during the challenge led to the acquisition and processing of datasets from TM and Penn— large-scale real-time results from Phase 1, Phase 2 and the ORSC are provided with comparisons to ground truth in Section 7.2.

The gridmaps in Figure 7.2 were generated by the Mapbuilder back-end and saved on the main GCS computer during deployment. These global gridmap outputs show that the submap gridmaps and pose graph data were being distributed around the UGV and GCS participants.

7.1.2. Distributed Occupancy Gridmap Comparison

This section demonstrates that Mapbuilder's global occupancy gridmaps converge towards the same solution by directly comparing the distributed copies built separately by UGV and GCS participants. When the hybrid-decentralized design is performing as expected, the distributed copies should appear very similar, but not identical. Section 4.4.2.3 explains how a threshold is used to determine when submap pose changes are broadcast. This threshold is tuned to balance pose graph convergence against wireless bandwidth usage. Cells far from each submaps origin (~20 meters) are expected to occasionally experience single cell (0.1 meter) misalignments.

Figure 7.3 (a) shows the global gridmap built by a GCS computer during a 180 meter drive through the University of Western Australia campus. During this on-line test the UGV was reconfigured to store copies of its locally generated gridmap outputs. In an off-line comparison, both UGV and GCS gridmaps appear visually identical. Minor differences are identified by comparing the gridmaps cell-by-cell. Figure 7.3 (c) shows a sample of these differences, where less than 0.1% of the cells are different. This minor noise does not affect the MRS's performance.

7.1.3. Distributed Pose Graph Comparison

This section demonstrates hybrid-decentralized MR-SLAM using priority filters and pinning (Section 4.3.9) by comparing copies of the distributed pose graphs[1]. The Mapbuilder back-end instances save snapshots of their local copy of the pose graph every five seconds. With a careful off-line analysis of these snapshots, the submap UUIDs can be cross

[1]It is difficult to compare large gridmap outputs since each UGV generates a 5000×5000 cell gridmap every second— around 4.4 GB/min of raw data.

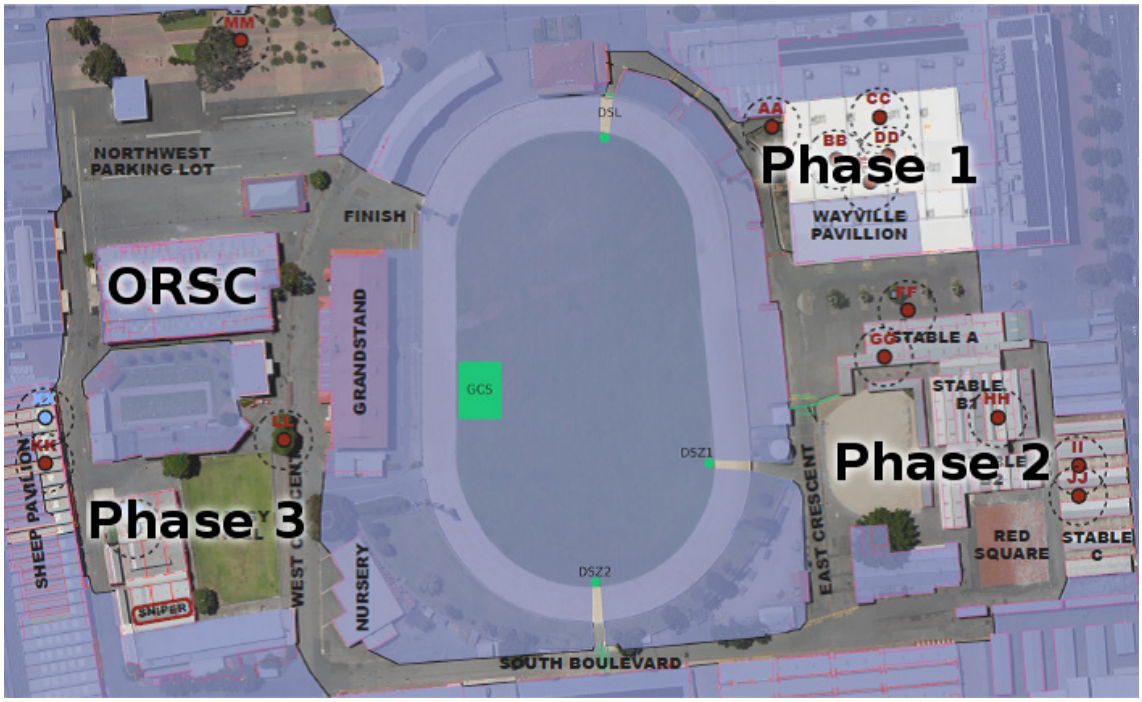


Figure 7.1. MAGIC challenge judge's map: Orthorectified map used to evaluate each team's performance in the four phases. This map is 500 meters wide, with partial ground-truth data overlaid in magenta. The GCS and phase starting locations are marked in green. Map courtesy Anthony Finn [52]. Survey and ground-truth data courtesy Adam Jacoff.

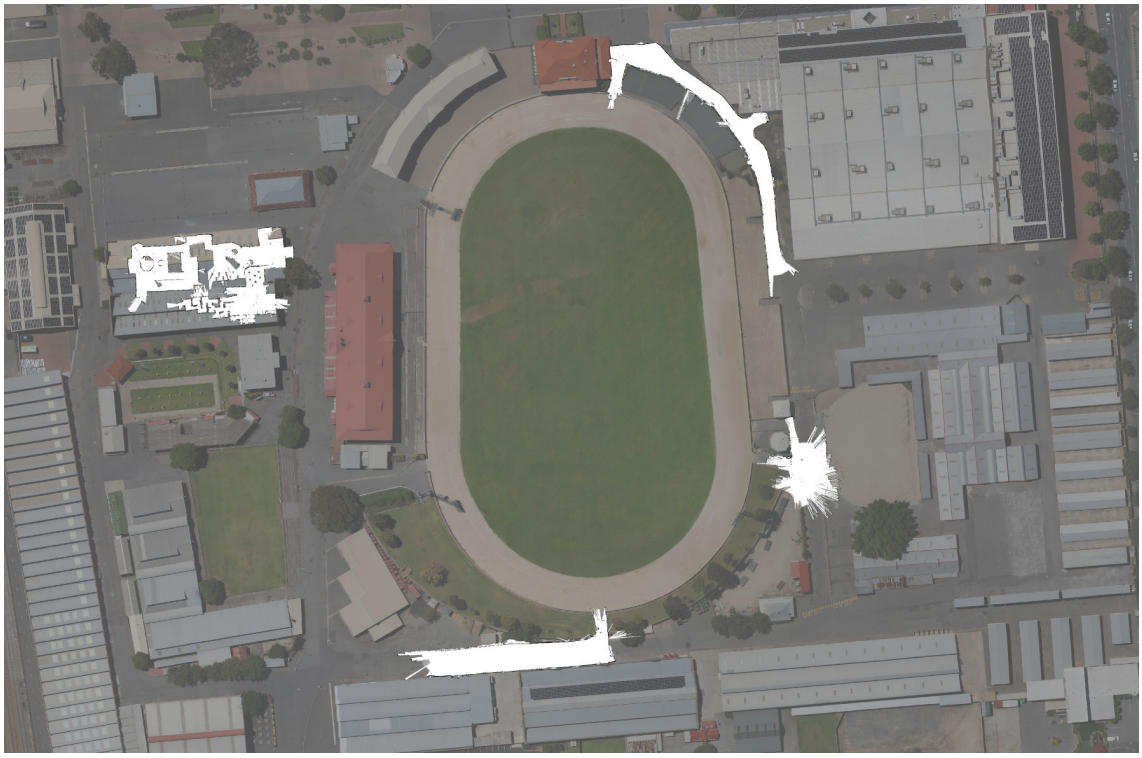


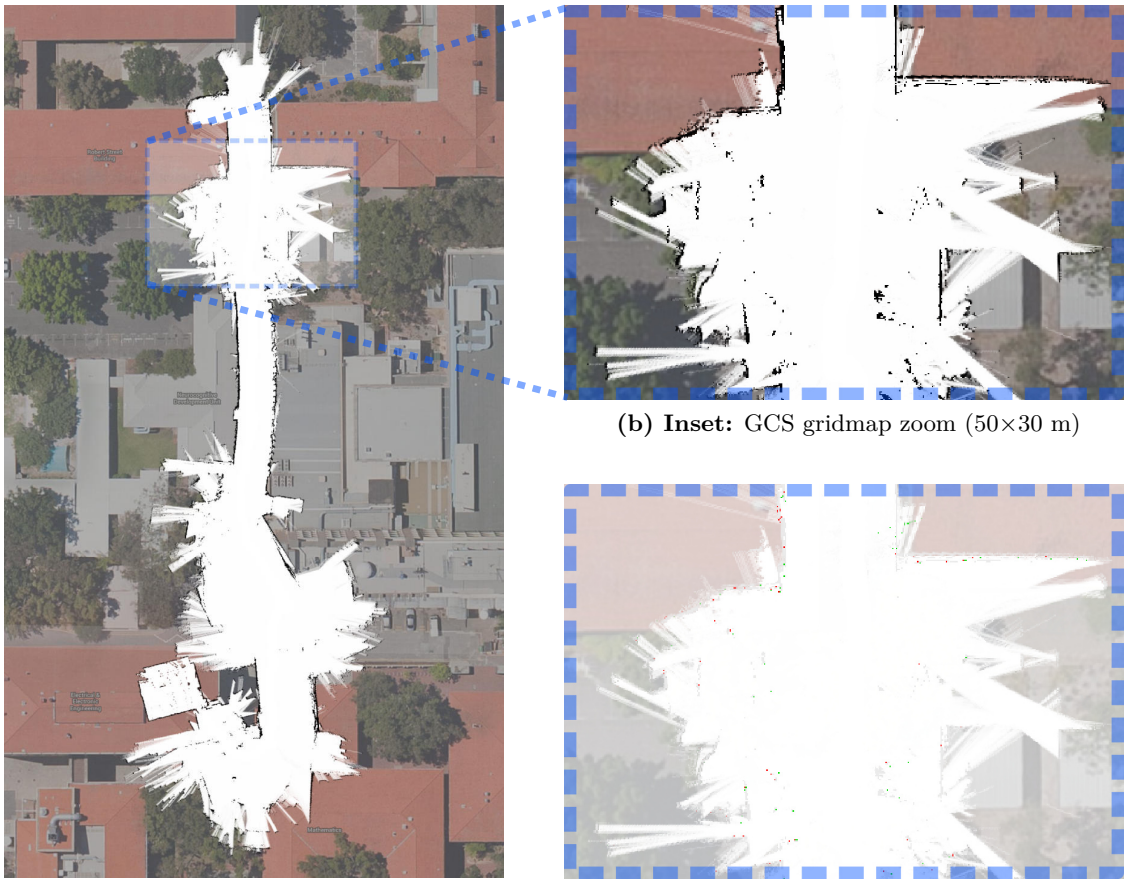
Figure 7.2. WAMbot challenge day results: Occupancy gridmaps superimposed on aerial imagery. Due to a combination of hardware and software issues, described in [64], our UGVs only explored 15 %, 8 % and 5 % of the three phases. WAMbot performed considerably better in the Old Ram Shed Challenge (ORSC) on the left, exploring 83 %. Aerial imagery courtesy DSTO.

referenced and the GCS and UGV participants' distributed pose graphs can be compared as they evolve over time.

Figure 7.4 plots a time-based comparison between the GCS and two WAMbots, recorded on-line during Phase 3 of the MAGIC challenge. The occupancy gridmap generated at the GCS is shown at the bottom of Figure 7.2 on the preceding page. The plots in Figure 7.4 show the steady growth in submaps and constraints as WAMbot 5 and 7 explored 100 meters into the phase. They demonstrate the distributed pose graph copies occasionally losing synchronization, as expected with lossy communications, and later resynchronizing as designed.

The sequence of submaps becoming "pinned" by the higher-priority GCS computers as described in Section 4.3.9, can be observed more clearly when enlarged in Figure 7.5. Here the flow of new submaps and constraints can be seen filtering out to the other computers; the GCS optimizes the pose graph and it takes 5-20 seconds for submap pose messages to pin the UGVs' pose graphs. The UGVs only have a few unpinned submaps at any moment in time; this leads to very minimal pose graph optimizations, as designed.

This was a particularly challenging area for wireless communications, since the UGVs were operating 150 m from the GCS and behind a large mound of dirt. All data had to be relayed, which meant bandwidth was restricted and intermittent. The network topology was not recorded during the challenge, however less than half of the normal 2-5 MB/s would have been available to the MRS. This bandwidth was shared with video streams and other subsystems. In this instance the DDS QoS design correctly discards minimal amounts of gridmap tile data (0.02% of occupied cells), and instead prioritizes pose graph data to correctly maintain synchronization.



(a) GCS gridmap (100×200 m)

(b) Inset: GCS gridmap zoom (50×30 m)

(c) Inset: WAMbot UGV gridmap differences

Figure 7.3. Mapbuilder Distributed MR-SLAM: Occupancy gridmaps generated by the WAMbot MRS over a 180 meter drive through the University of Western Australia. (a) is the gridmap built by Mapbuilder running on the GCS computer. (b) is an enlargement of the inset to show individual 10 cm grid cells. The gridmaps generated by Mapbuilder on the UGVs appear visually similar; to show the minor differences between the two distributed instances of Mapbuilder, (c) compares the cells between both gridmaps: white cells are identical, red cells are only occupied on the GCS, while green cells are only occupied on the UGV. The differences, which are less than 0.1% of cells have no affect on the performance of the MRS. This test was performed to evaluate decentralized MRS communications and not mapping accuracy— several artifacts are visible at the top of (b) that were caused by odometry and submap matching errors. These errors were not corrected, however they could have been using the Mapbuilder GUI. Aerial map courtesy Google.

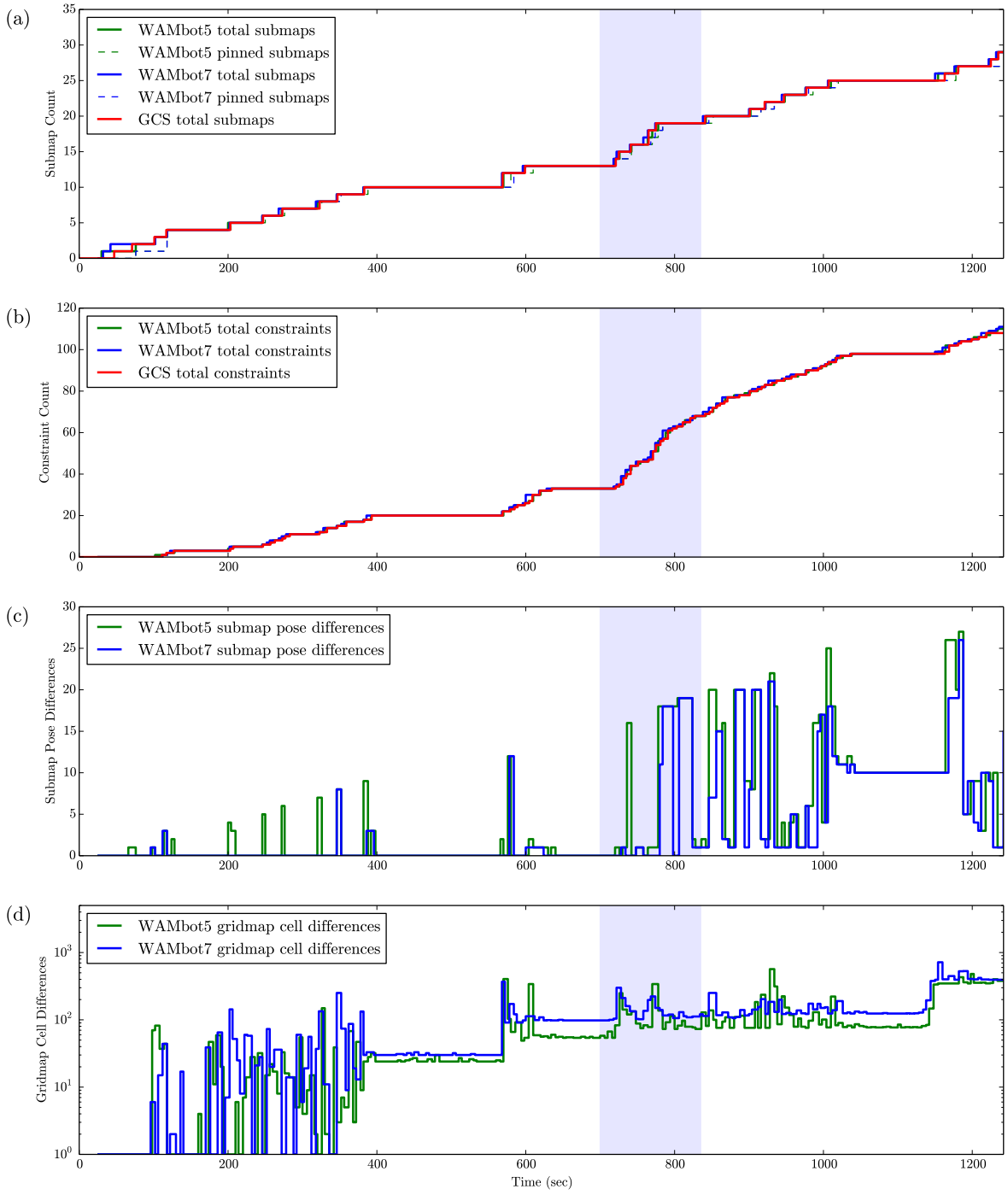


Figure 7.4. Mapbuilder Hybrid-Decentralized MR-SLAM: Time-based comparisons between three distributed copies of the pose graph recorded on-line during Phase 3 of the MAGIC challenge. The occupancy gridmap generated by the GCS is shown at the bottom of Figure 7.2 on page 171. While the WAMbot UGVs explored the first 100 meters of the phase, each Mapbuilder instance stored snapshots of the pose graph every 5 seconds. Plots (a) and (b) show the steady growth in submaps and constraints as WAMbot 5 and 7 explored. The area highlighted in blue is enlarged and explained in Figure 7.5. Plot (c) shows the number of submaps that had different pose estimates to the GCS; differences that are quickly corrected by the propagation of submap pose messages. Around t =1000 s the MRS experiences severe communications issues and the UGV pose graphs are only resynchronized after one of the UGVs moves into a better position to relay data. Plot (d) indicates a minor loss of gridmap data at the GCS, however over 99.98% of gridmap cells are successfully distributed at the end. This plot confirms that the QoS design correctly discards gridmap tile data, and instead prioritizes pose graph data to correctly maintain synchronization.

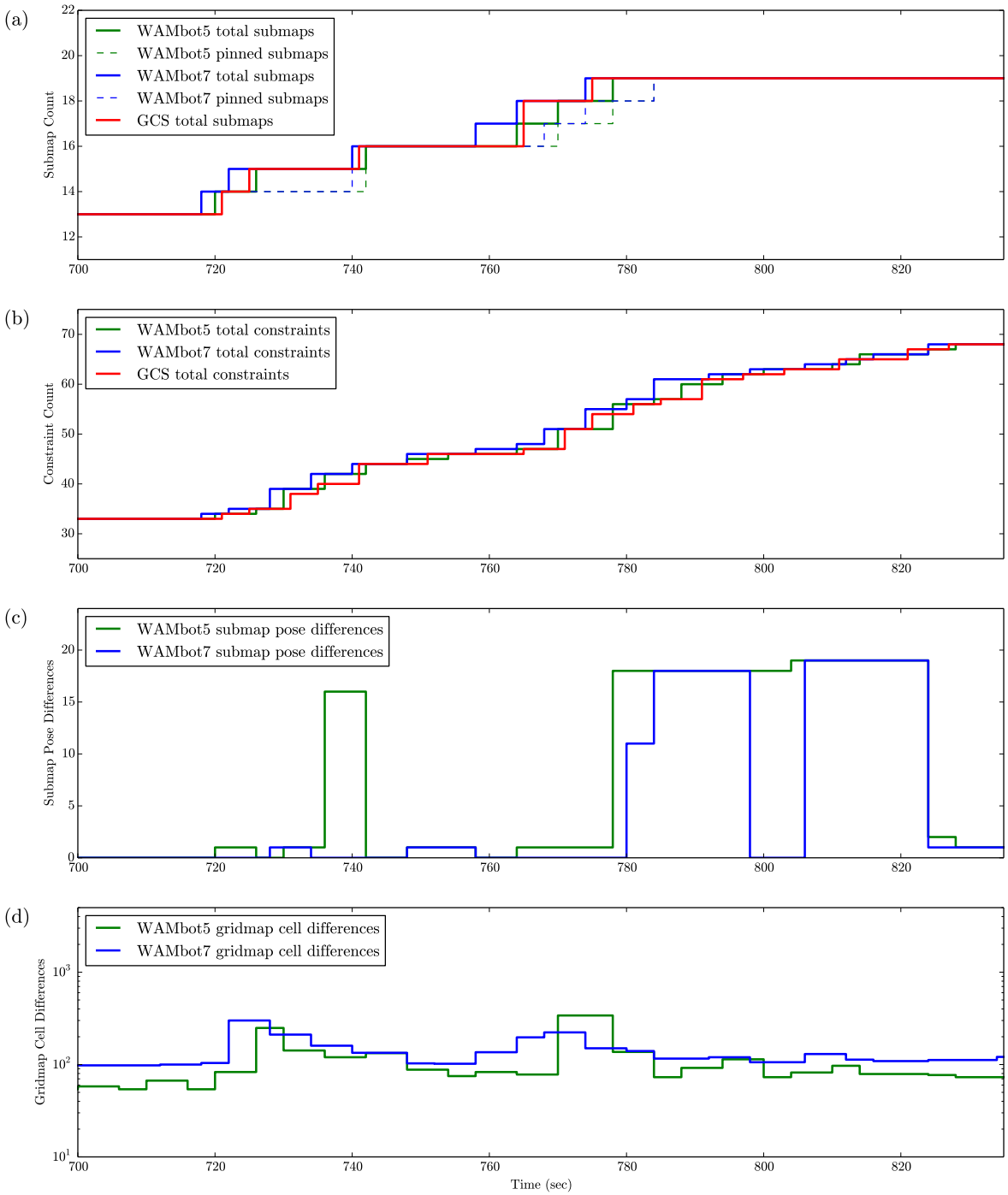


Figure 7.5. Mapbuilder Hybrid-Decentralized MR-SLAM: Enlarged section of the time-based pose graph comparison in Figure 7.4. Here WAMbot 7 drives 15 meters and creates six submaps, while the pose graphs were recorded every five seconds. Plot (a) indicates that new submaps in the WAMbots' pose graphs took up to five seconds to be shared to the GCS computer and then 5-20 seconds to be pinned. Plot (b) shows several constraints being generated for each new submap, the total count growing more smoothly as the Matcher takes time to generate and distribute new constraints. The UGVs remained stationary for a while around t =780, however they continue to match submaps and their new constraints fully connect the pose graph. In plot (c) the stream of newly created constraints causes several large changes to the submap pose estimates. The pose graphs resynchronize after some time. Plot (d) shows the number of gridmap cell differences changing quite dynamically, even when the UGVs are stationary. These changing cell differences indicate that objects were moving in the environment. This plot demonstrates the hybrid-decentralized approach pinning submaps to maintain pose graph synchronization.

7.2. Large-Scale Real-Time Multi-Robot SLAM

This section demonstrates real-time MR-SLAM using datasets recorded at the MAGIC challenge. The results presented here show how Mapbuilder and my research contributions can be deployed at large scales. The challenge and datasets were divided into the three phases that are listed in Table 7.1. These challenge phases, or test environments, are described in Appendix B.

Multiple datasets were recorded by the WAMbot, TM and Penn teams during the challenge, they are combined and replayed in real-time to prepare these results. Appendix A describes the MRS architectures and UGV front-end designs for the resulting team of 10-23 heterogeneous UGVs. Each UGV sensor package included at least wheel odometry, an IMU and one lidar scanner (either fixed in a horizontal plane, nodding up/down around the horizontal plane, or sweeping left/right with a vertical plane).

This section includes figures that show the evolution of global gridmaps over time, however these results are best viewed as video recordings of the GUI, captured while the datasets were processed in real-time. These videos are available on-line at http://reid.ai/thesis. The three datasets were all processed in the same way, notes common to each include:

- Each figure and video displays a time offset in the top right corner; these time offsets are used to refer to specific events in each dataset. These events can be viewed easily in the on-line videos by clicking on the gridmaps in the PDF version of this thesis. Similarly, links are embedded in text like this: (open video), which, when clicked, will open directly to a particular time in the corresponding video.

- Large non-Gaussian odometry errors occur frequently in these datasets. As discussed in Section 4.1.4 and Appendix B, the operator must intervene to correct for these errors. Corrections are performed by deleting corrupt submaps, and applying ground-truth constraints. The videos are overlaid with the captions *"Deleted submap"* and *"Adjusting ground truth"* to clearly indicate when these actions are performed.

- The total odometry, summed over all UGVs' tracks is shown in the lower right of each figure. This value is *not* monotonic; it decreases occasionally when the operator deletes submaps.

Phase	Phase Extents	UGV Count	Total Odometry	Time
Old Ram Shed Challenge	80×40 m	10	3,170 m	36 min
MAGIC Challenge Phase 1	210×140 m	14	6,153 m	80 min
MAGIC Challenge Phase 2	210×150 m	23	8,373 m	88 min

Table 7.1. Results: Large-scale real-time MR-SLAM dataset summary

- Mapbuilder fuses different versions of the output gridmap in parallel, each using a different algorithm for fusing gridmap cells (Section 5.2). The gridmaps displayed throughout these videos use an algorithm tuned to help the operator; other algorithms are designed for navigation.

- TM and Penn's UGVs and MRS were described in Appendix A. The development time required to integrate their datasets was on the order of tens of hours— testament to Mapbuilder's flexible back-end design and ability to inter-operate with different SLAM front-ends and heterogeneous UGVs.

- While Mapbuilder is designed to be distributed and decentralized, real-time large-scale MR-SLAM is demonstrated here in a centralized manner. This centralization is trivial and requires minor modifications; a single instance of Mapbuilder is executed, providing the MR-SLAM back-end and GUI, with additional threads processing the dataset log files.

- Processing was performed on a 2012 model Intel quad-core i7 CPU, with a NVIDIA GTX 970 GPU. For reference, this GPU was about 25 times more powerful than the WAMbot UGVs (Appendix A), and about half as powerful as the fastest COTS GPUs today. In a distributed deployment each UGV has enough GPU processing resources to perform its own submap matching. In this centralized configuration, the GPU performed centralized submap matching for all 23 UGVs.

- In the results presented here, about 25 % of the CPU's resources were used to capture the GUI screen at 30 Hz and encode the frames into h.264 format.

- Tunable parameters were not changed when moving between environments— the same parameters were used for all UGV types, and both indoor and outdoor environments.

- Communications losses occur in both teams' datasets. This can be observed many times in the accompanying videos, where the UGVs "jump" tens of meters and become badly localized. Appendix B describes these issues in more detail.

- Communications bandwidth estimates are calculated based on the size and volume of messages that would have been sent in a decentralized and distributed deployment. Estimates include the RLE compressed size of the submap gridmaps, and assume that all messages are sent using UDP broadcasts.

- To show Mapbuilder's computational cost scaling linearly with the size of the pose graph, some of the plots in this section place the total submap count on the abscissa, rather than time. This avoids the "flat spots" in the time-based plots where the UGVs are motionless for periods of time.

- To demonstrate the global gridmapping accuracy, surveyed maps were used to calculate the root-mean-squared error (RMSE). These surveyed ground-truth maps were only available for limited parts of the challenge. The RMSE was measured manually by placing a five meter grid over both the global gridmap and surveyed data. To avoid any sampling bias, a single representative association was made in each grid square, from which the error between the gridmap and survey was recorded.

7.2.1. Old Ram Shed Challenge

The Old Ram Shed Challenge (ORSC) datasets are described in Appendix B. A single dataset with 10 UGVs exploring an 80×40 meter indoor area was created by merging Penn and TM's data. This is the smallest dataset in these results, with 3,170 meters of odometry accumulated over 36 minutes. The dataset is unique because of the high average node degree and connectivity (Section 3.3.3). For reference, this dataset is about three times larger than the 45×25 meter indoor area used for the SDR program (Section 1.3.4).

I operated the Mapbuilder GUI to generate the ORSC results presented here. Prior to generating these results I had processed the dataset dozens of times and viewed the site in person. This prior knowledge is unlikely to have affected the mapping accuracy— only four minutes of cumulative time was spent adjusting ground-truth constraints. This four minutes was mostly spent initializing UGV poses and correcting repeated, systematic, odometry errors from UGV number 17. Figure 7.7 (e) indicates that the majority of this dataset was processed with minimal operator intervention.

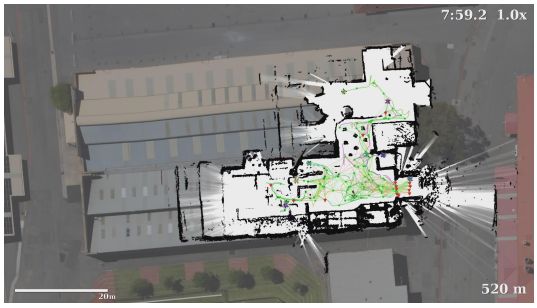
Figure 7.6 shows a sequence where the global gridmaps evolve over time as viewed in the Mapbuilder GUI. The largest of several loop closures is highlighted in part (c) and (d) of this figure. This loop closure forms a 230 meter closed path, as measured from the point in the first room where the UGVs separated. In the pose graph this corresponds to a minimum spanning tree of about 70 edges (constraints).

In the middle of the southern part of the Old Ram Shed a maze was built out of straw hay bales. Figure B.3 on page 236 shows two photos of this maze. The bales were stacked to form walls that were low, badly defined and difficult to map with lidar. At the start of the maze area, after eight minutes had elapsed, UGV number 17 becomes stuck and severe odometry errors occur (open video). The operator repeatedly deletes corrupted submaps and repositions the UGV in its correct location. Communications issues can be seen around 32 minutes, where the UGV numbered 13, 14 and 18 make several 5 meter jumps across the map (open video).

The surveyed ground-truth data is overlaid on the final gridmap in Figure 7.6 (e). Many of the internal obstacles in the Old Ram Shed were not surveyed, including the hay bale

maze. Based on the areas that were surveyed, the RMS error calculated from 88 samples was ±0.27 meters.

Time-based statistics are shown in Figure 7.7, while statistics plotted against the submap count are shown in Figure 7.8. Key measurements and statistics for the submaps, their constraints, and the optimization errors are shown using histograms in Figure 7.9. These results are summarized in Table 7.2 on page 196.



(a) 7m 59s: The UGVs start in the south-east corner. The first team explores west, the second team north.

(b) 15m 11s: Both teams explore on separate fronts. The first team enters a maze of straw hay bales.

(c) 22m 45s: The first team is about to exit the maze into the south-west corner, closing a loop with the second team.

(d) 29m 9s : After a 230 m loop closure. The map alignment continues to improve after more submaps are matched.

(e) 36m 5s (end): Global gridmap with surveyed ground truth overlaid (magenta). Note: many temporary barriers were not surveyed, including the straw hay bale "maze". The RMS error calculated from 88 samples was ±0.27 meters.

Figure 7.6. Results: Old Ram Shed Challenge. 10 heterogeneous UGVs explore a 80×40 meter indoor area in 36 minutes**.** Global gridmaps overlaid on aerial imagery. UGVs are shown colored by their identifier, their tracks overlaid as thin lines in matching colors. The pose graph is shown in green: dots represent submap poses, lines submap constraints and red triangles are ground-truth constraints. Click on the images to open the accompanying video at the corresponding time.

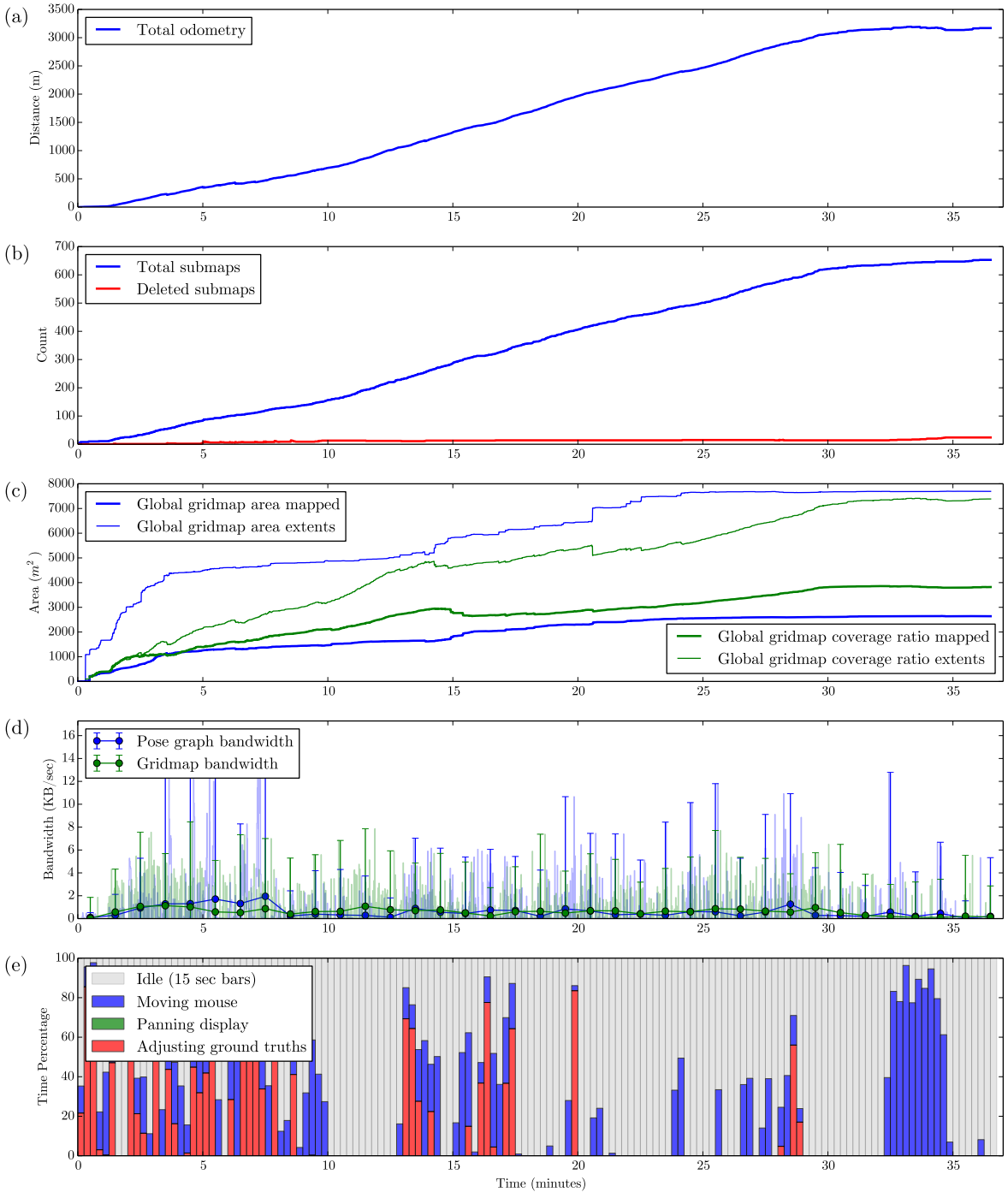


Figure 7.7. Results: Old Ram Shed Challenge. 10 heterogeneous UGVs explore a 80×40 meter indoor area in 36 minutes**.** In these time-based plots: (a) shows the steady accumulation of 3,170 m of odometry. (b) shows 653 submaps being created, 24 were deleted, giving an average of 5.04 m of odometry per submap. (c) shows the global gridmap growing to 2,640 m^2 total area, while an average of 38 submaps overlap each part of the map. (d) shows the variations in communications bandwidth, averaging 1.2 KB/s for a total of 2.5 MB of MR-SLAM data for the entire dataset. (e) shows a histogram of operator interactions in 15 s bars. The operator spent 10.6 % of the time, or just under 4 minutes, correcting for odometry errors.

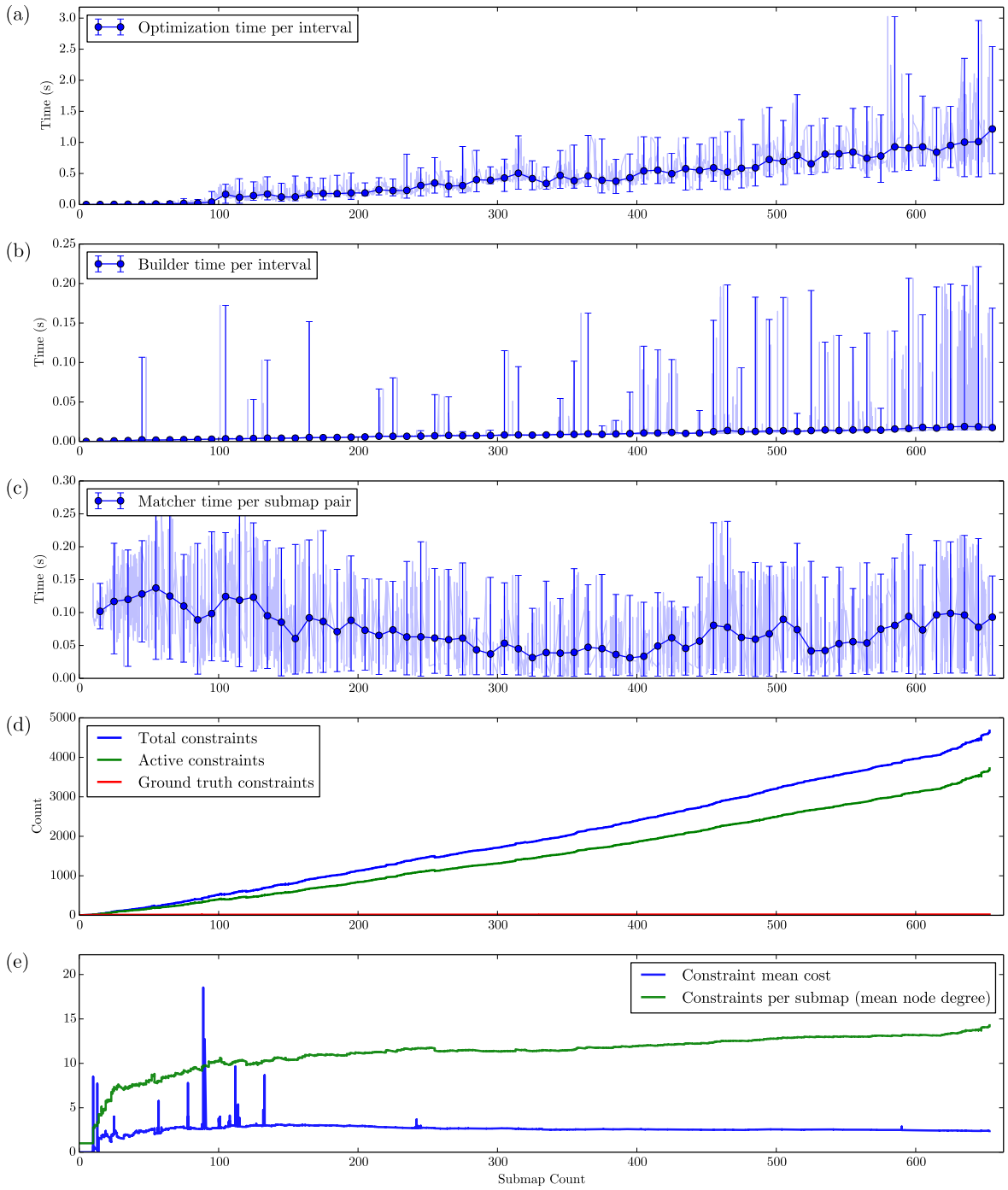


Figure 7.8. Results: Old Ram Shed Challenge. Plots (a), (b) and (c) shows the minimum, maximum and mean value per execution, with the total submap count on the abscissa. These statistics are noisy due to heavy contention for the CPU and GPU. (a) shows the optimization time per interval, which scales approximately linearly, at 1.1 seconds per 1,000 submaps. (b) shows the time the GPU spent building each 512×512 meter global gridmap; build times scale linearly at 0.028 seconds per 1,000 submaps. (c) shows the time the GPU spent matching submaps; an average of 0.078 seconds per match. (d) shows the growth in submap constraints; 80 % of the 4,674 constraints are active (not outliers). (e) indicates that the UGVs' exploration approach produced a relatively constant number of constraints per submap (responsible for the linear optimization times). The mean constraint cost in (e) decreases over time; spikes occur when the operator adjusts ground-truth constraints.

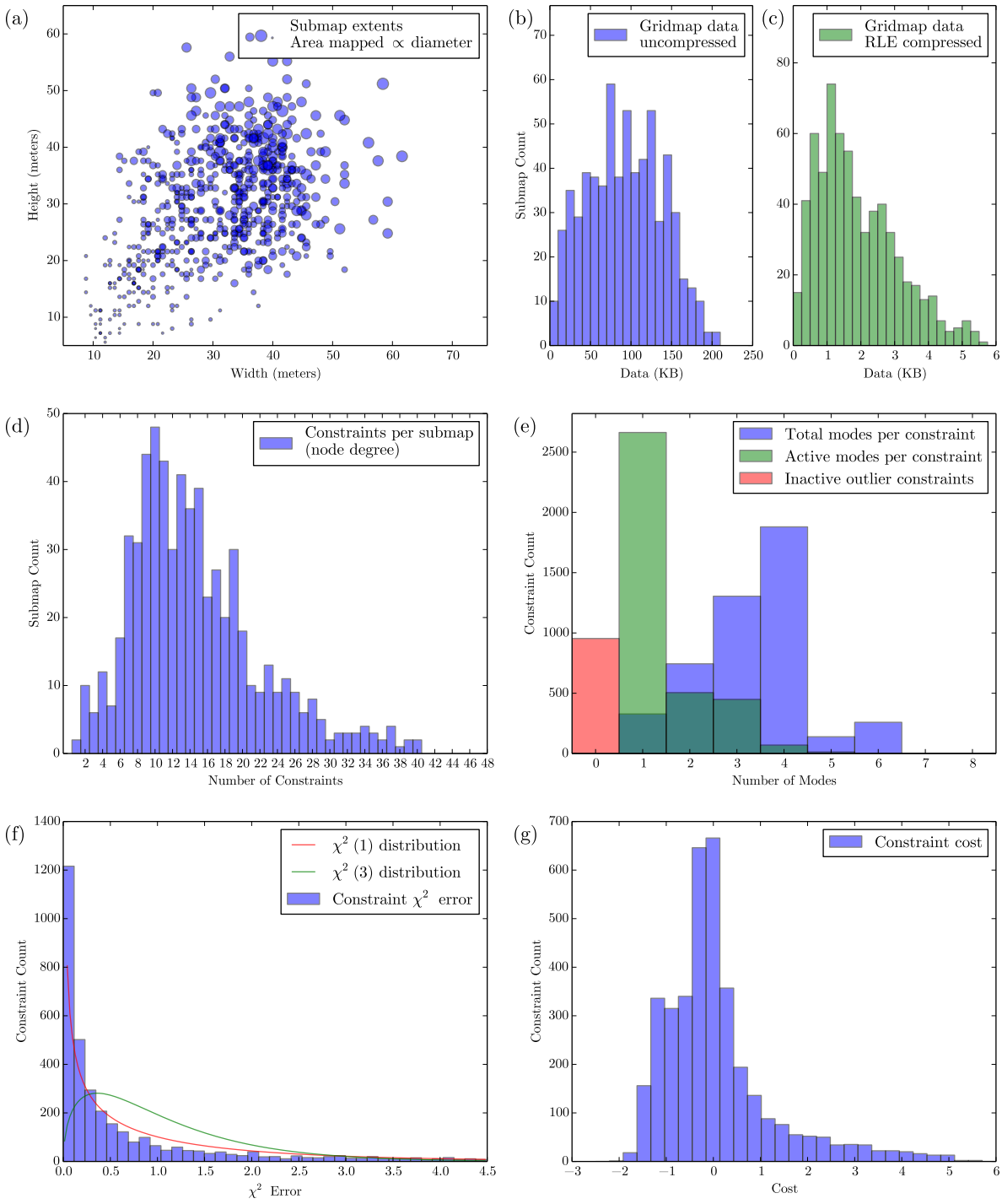


Figure 7.9. Results: Old Ram Shed Challenge. Plot (a) shows the distribution of submap extents; the average submap is 31×30 meters, while 170 m^2 of each gridmap area is mapped (occupied or free cells). (b) shows the average submap is 95 KB uncompressed, while (c) shows that the RLE compression reduces the average submap to 1.9 KB. (d) shows the number of constraints for each submap, or the node degree, which averages 14.3. (e) shows both the total number of modes per constraint, and the number of these modes that were active (refer to Chapter 6). (f) shows the expected χ^2 distribution, and the actual error distribution. (g) shows the distribution of constraint costs (refer to Chapter 6 also).

7.2.2. MAGIC Challenge Phase 1

Phase 1 was described in Appendix B. Only TM's data was available for this phase, creating the second largest dataset with 14 UGVs exploring over 9,000 m^2 of mixed indoor and outdoor environments in 80 minutes. I operated the Mapbuilder GUI to generate the results presented here. Having not viewed the Phase 1 area while at the challenge, and having not processed the dataset beyond the first seven minutes, I made decisions with the same logic that a trained operator would have used. Figure 7.10 and Figure 7.11 show a sequence of global gridmaps as they evolve over time as viewed in the Mapbuilder GUI.

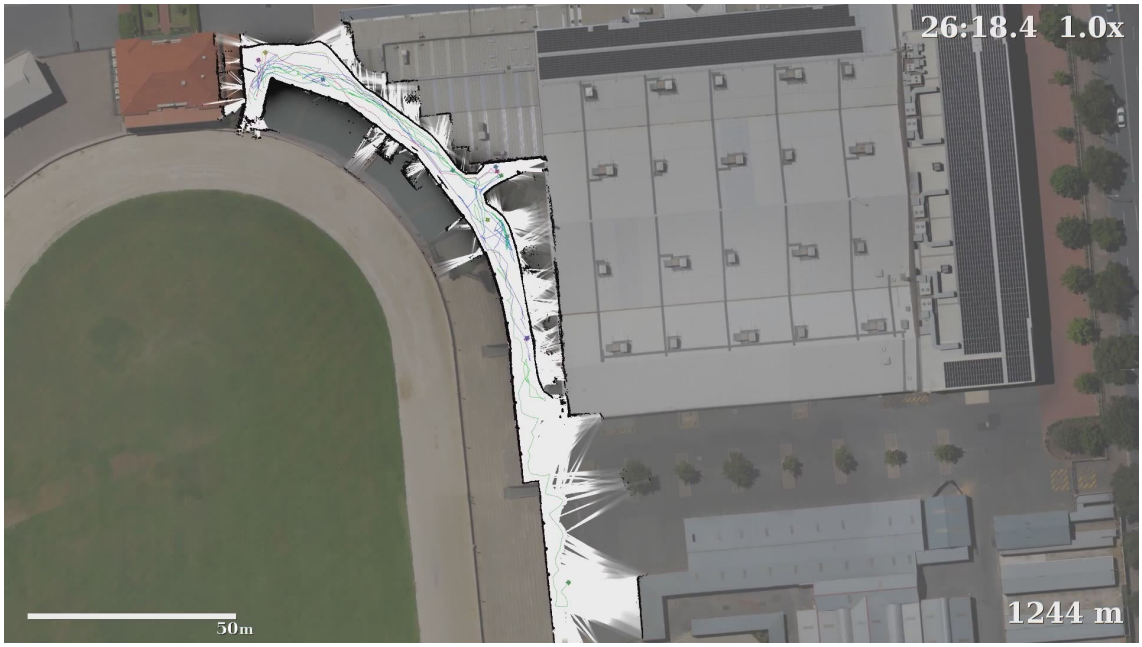
The UGV team takes 26 minutes to explore 170 meters down the west edge of the phase. Based on TM's challenge-day maps (Figure B.2 on page 235), their UGVs experienced large odometry errors in this long alley, which when combined with the degenerate scan matching configurations, took significant operator activity to correct (open video).

Transitioning through doorways can be particularly challenging, especially when there is a void on the other side. This was the case entering the pavilion, as shown in Figure 7.10 (b) (open video). Based on odometry, and with limited hints from submap matching, the operator quickly added ground-truth constraints. These constraints aligned the west-facing interior wall with the exterior wall, and the south-facing interior wall with the roof geometry visible in the aerial imagery.

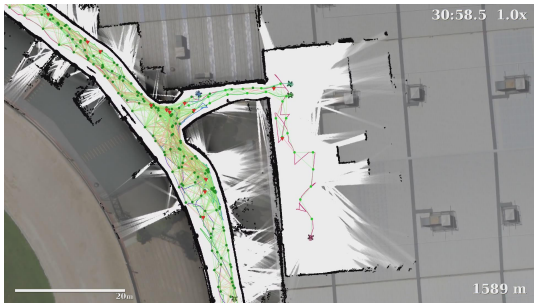
Figure 7.10 (d) shows the largest loop closure, around a 280 meter path. Odometry errors and perceptual aliasing resulted in more than two meters of accumulated drift before the closure. This drift was larger than the Mapbuilder Matcher's search range, which aligned the dominant north-facing interior wall. Two ground-truth constraints were added and the submaps snapped into alignment (open video). Inside the pavilion, the operator had limited information to correct for odometry errors. The roof beams, visible in the "x-ray" view in Figure 7.11 (d) could have been used to align to the roof geometry, however as the maps grew incrementally the operator could not be confident in this alignment.

The surveyed ground-truth data is overlaid on the final gridmap shown in Figure 7.11 (e). Based on the areas that were surveyed, the RMS error was $\pm 0.62\,m$. The north-facing wall inside the pavilion had an offset of about 1.6 meters. If I were to process these datasets again, with the benefit of hindsight, these walls would be correctly aligned to the roof geometry and the RMS error greatly reduced.

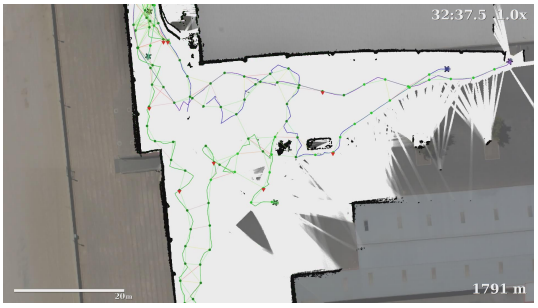
Time-based statistics are shown in Figure 7.12, while statistics plotted against the submap count are shown in Figure 7.13. Key measurements and statistics for the submaps, their constraints, and the optimization errors are shown using histograms in Figure 7.14. These results are summarized in Table 7.2 on page 196.



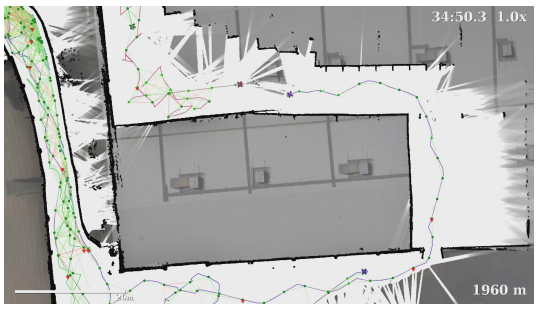
(a) 26m 18s: The 14 UGVs start in the north-west corner and explore down the west edge of the phase. Large odometry errors occur in this alleyway. One team of UGVs assembles near the west door to the pavilion, a second team moves south into the open car park.



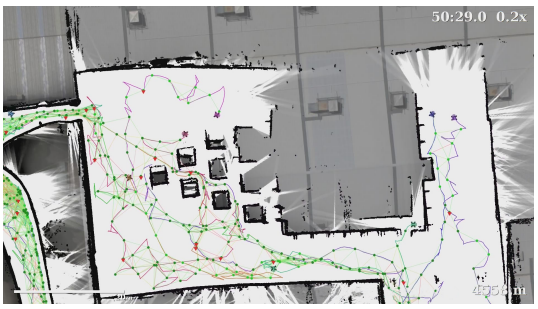
(b) 30m 58s: The first team enters the pavilion. With limited submap overlap ground-truth constraints are required to align the inside and outside walls.



(c) 32m 37s: The second UGV team explores the car park and finds the southern entrance. Odometry errors accumulate quickly in the sparse environment.

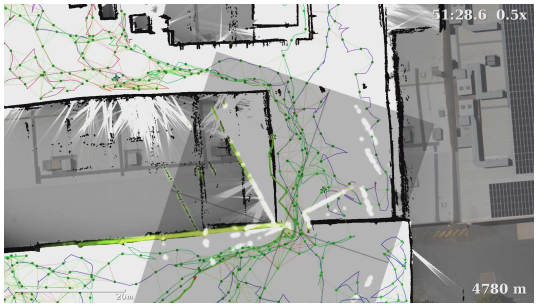


(d) 34m 50: Before loop closure between two UGVs at top center. Perceptual aliasing and odometry errors have shortened the walls, however it is unclear which ones.

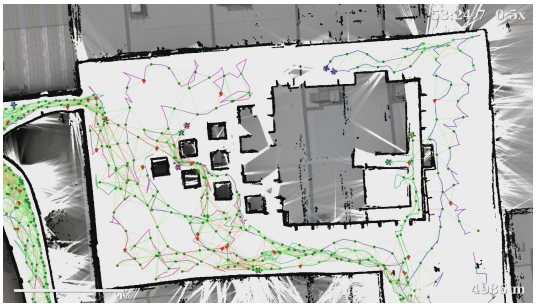


(e) 50m 29s: The inner pavilion is explored after the loop closure. One UGV team explores the east corridor and another enters the inner maze.

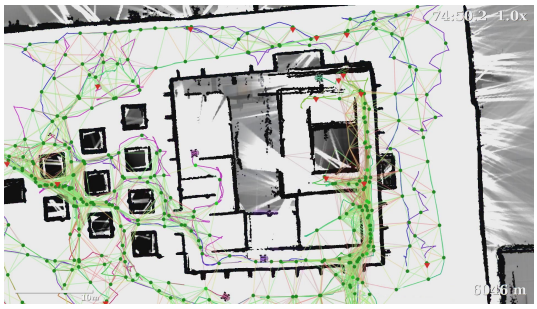
Figure 7.10. Results: MAGIC challenge Phase 1. TM's 14 UGVs explore a 210×140 m mixed indoor and outdoor environment in 80 minutes**.** The pose graph is shown in green: dots represent submap poses, lines submap constraints and red triangles are ground-truth constraints. Click on the images to open the accompanying video at the corresponding time.



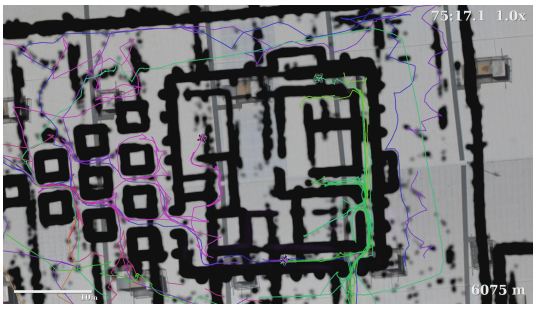
(a) 51m 28s: Odometry errors while entering the southern doorway. One of many instances where the submap gridmaps were corrupted (white ghosting).



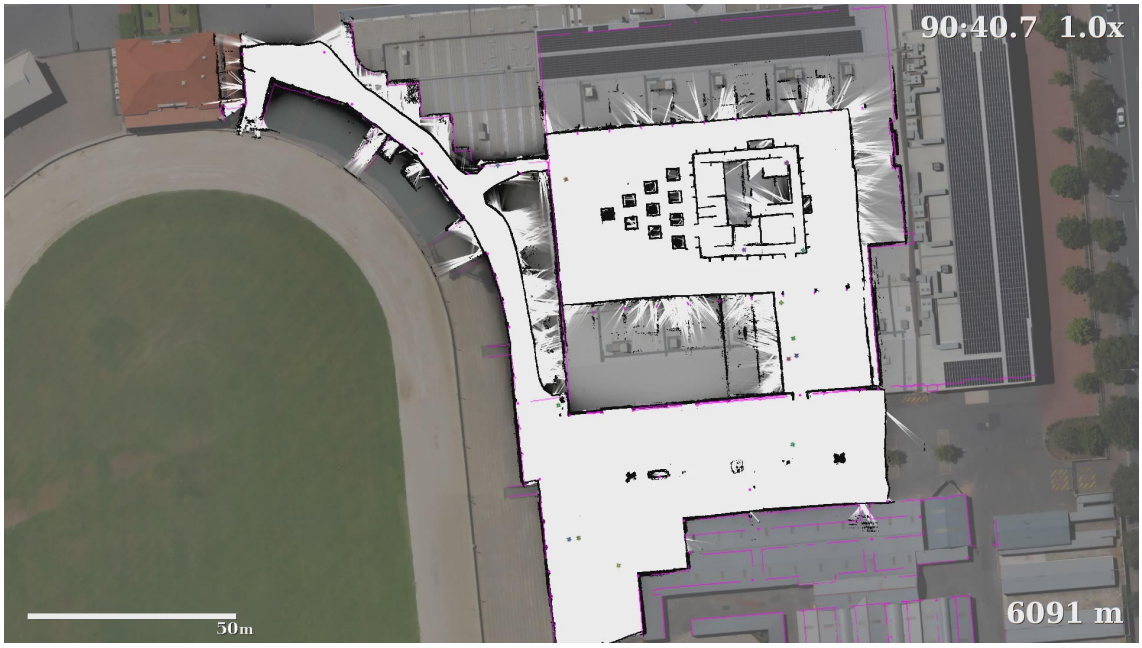
(b) 53m 24s: The second largest loop closure on the northern edge of the pavilion. Additional ground truths were required to resolve the perceptual aliasing.



(c) 74m 50s: The pavilion is 95 % explored. The inner maze contained many 15° ramps that tested the UGVs' mobility.



(d) 75m 17s: Alternate "x-ray" view showing roof beams and clutter that is successfully filtered in the regular gridmaps.



(e) 90m 40s: Global gridmap with surveyed ground truth overlaid (magenta). The temporary structures inside the pavilion were not surveyed. Note: the dataset ended after 80 minutes. The RMS error calculated from 76 samples was ±0.62 meters.

Figure 7.11. Results: MAGIC challenge Phase 1. TM's 14 UGVs explore a 210×140 m mixed indoor and outdoor environment in 80 minutes**.** The pose graph is shown in green: dots represent submap poses, lines submap constraints and red triangles are ground-truth constraints. Click on the images to open the accompanying video at the corresponding time.

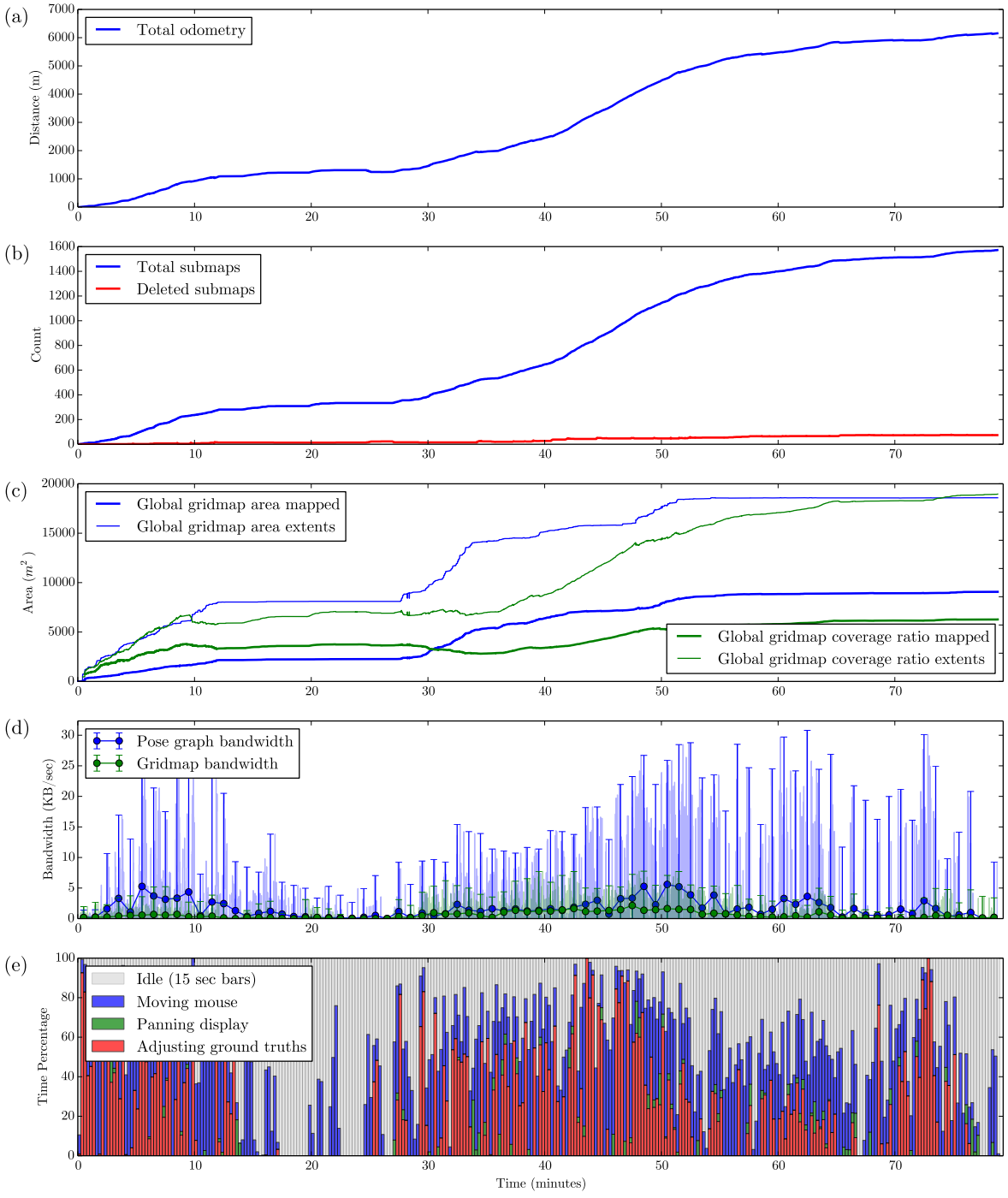


Figure 7.12. Results: MAGIC challenge Phase 1. TM's 14 UGVs explore a 210×140 m mixed indoor and outdoor environment in 80 minutes. In these time-based plots: (a) shows the steady accumulation of 6,153 m of odometry. (b) shows 1,572 submaps being created, 74 of them were deleted, giving an average of 4.1 m of odometry per submap. (c) shows the global gridmap growing to 9,067 m^2 total area, while an average of 22 submaps overlap each part of the map. (d) shows the variations in communications bandwidth, averaging 2.2 KB/s for a total of 10.2 MB of MR-SLAM data for the entire dataset. (e) shows a histogram of operator interactions in 15 s bars. The operator spent 22.4 % of the time, or about 18 minutes, correcting for odometry errors.

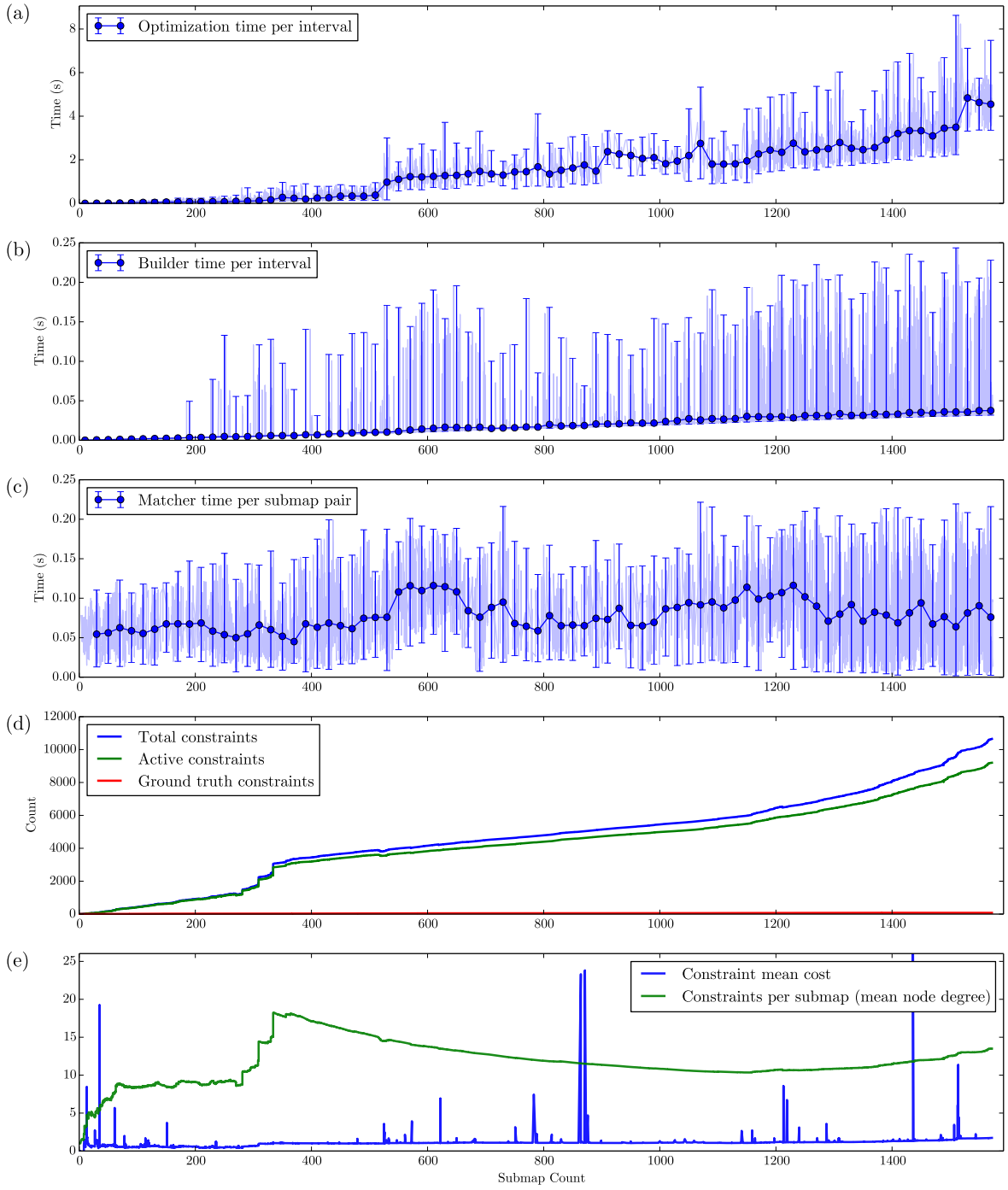


Figure 7.13. Results: MAGIC challenge Phase 1. Plots (a), (b) and (c) show the minimum, maximum and mean value per execution, with the total submap count on the abscissa. These statistics are noisy due to heavy contention for the CPU and GPU. (a) shows the optimization time per interval, which scales approximately linearly, at 1.53 seconds per 1,000 submaps. (b) shows the time the GPU spent building each 512×512 meter global gridmap; build times scale linearly at 0.021 seconds per 1,000 submaps. (c) shows the time the GPU spent matching submaps; an average of 0.075 seconds per match. (d) shows the growth in submap constraints; 86 % of the 10,649 constraints are active (not outliers). (e) shows the average number of constraints per submap peaking at 18 after all 14 UGVs drive down the first alley. This decreases as the UGVs explore in smaller groups.

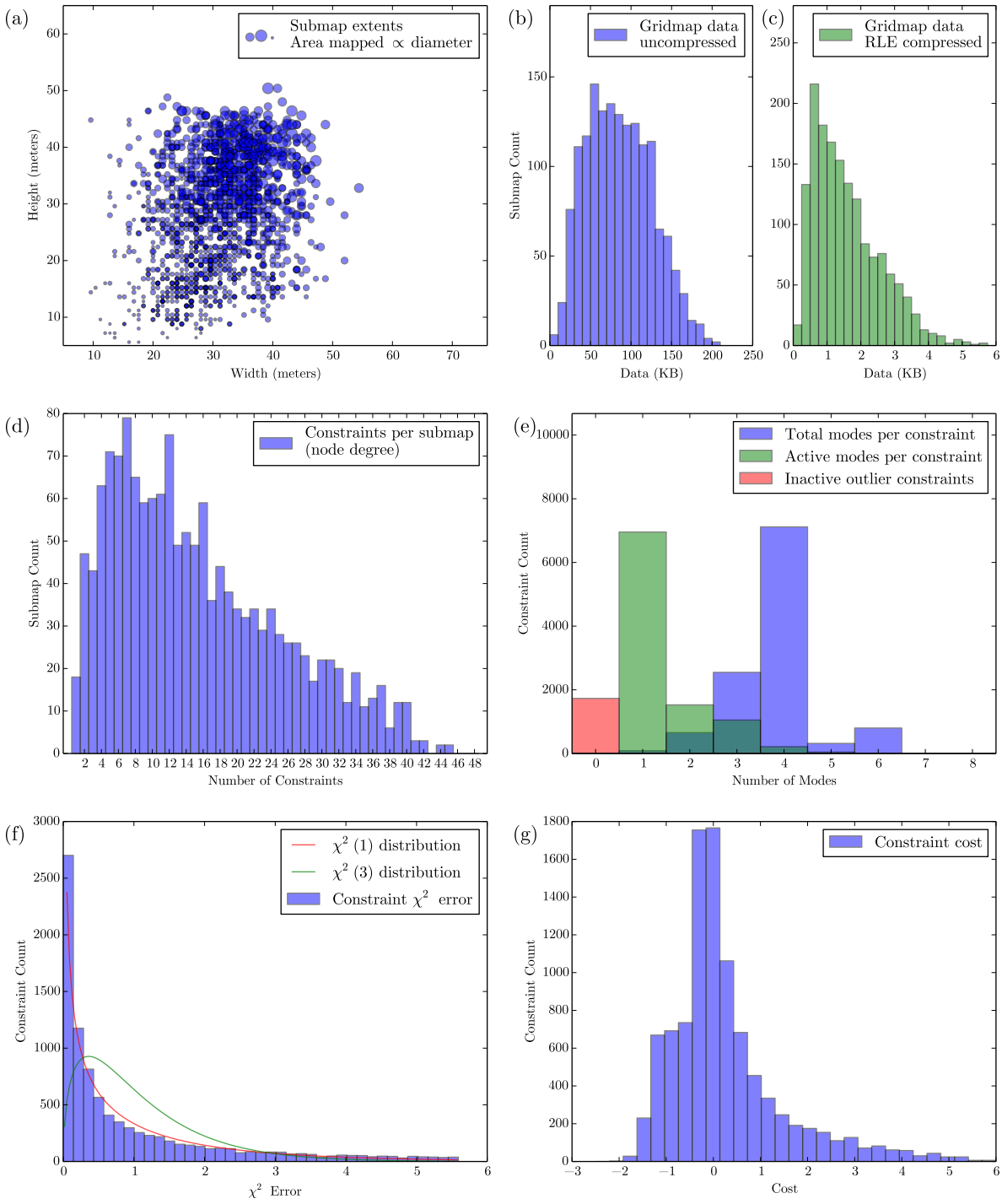


Figure 7.14. Results: MAGIC challenge Phase 1. (a) shows the distribution of submap extents; the average submap is 30×28 meters, while 144 m^2 of each gridmap area is mapped (occupied or free cells). (b) shows the average submap is 86 KB uncompressed, while (c) shows that the RLE compression reduces the average submap to 1.6 KB. (d) shows the number of constraints for each submap, or the node degree, which averages 14.7. (e) shows both the total number of modes per constraint, and the number of these modes that were active (refer to Chapter 6). (f) shows the expected χ^2 distribution, and the actual error distribution. (g) shows the distribution of constraint costs (refer to Chapter 6 also).

7.2.3. MAGIC Challenge Phase 2

The Phase 2 test environment is described in Appendix B. This is the largest combined dataset, with 23 heterogeneous UGVs (14 from TM and 9 from Penn) exploring over 11,400 m^2 of mixed indoor and outdoor environments across 88 minutes. I operated the Mapbuilder GUI to generate these results. Having processed Penn's Phase 2 dataset several years earlier [66], the environment was only vaguely familiar. The results presented here were the first time that TM's datasets, and all 23 UGVs together, had been processed beyond the first 20 minutes. These results are representative of what a trained operator might achieve on-line.

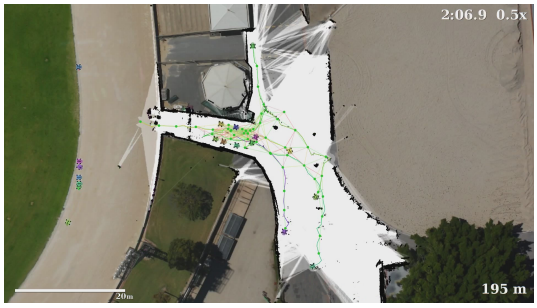
TM and Penn's UGVs were tested on separate days and their movements in the Phase 2 datasets are uncoordinated. This presents a challenge to the operator when multiple events occur on opposite sides of the phase simultaneously. The combined dataset was processed in real-time, however log playback, or effectively the UGVs' velocities were occasionally slowed to 0.5× real-time to avoid overloading the single operator.

Figure 7.15 and Figure 7.16 show a sequence of global gridmaps as they evolve over time in the Mapbuilder GUI. Perceptual aliasing problems are frequent in this phase due to large odometry errors, combined with relatively featureless alleys. Figure 7.15 (b) and (c) show submap matching and pose graph optimization correcting several meters of accumulated odometry errors when the UGVs reach the end of the first alley.

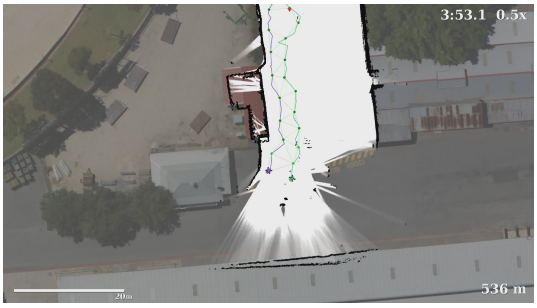
About 35 minutes into the dataset, 10 UGVs converge on the 45×45 meter square in the south-east corner of the phase (open video). A photo of this area is shown in Figure B.5 on page 237. Figure 7.16 (a) and (b) show a 210 meter loop closure occurring here. This area is challenging because the open space is larger than the lidar sensor's range, requiring repeated operator intervention.

The largest loop closure, about 330 meters, occurs in the horse stables. Figure 7.16 (c) shows the large odometry drift experienced by three UGVs prior to this loop closure. The pot-holes that caused these non-Gaussian errors can be seen in the photo in Figure B.4 (b) on page 237. The operator corrected the gross odometry errors, and the submap matching and pose graph optimization corrected the remaining 1-2 meters.

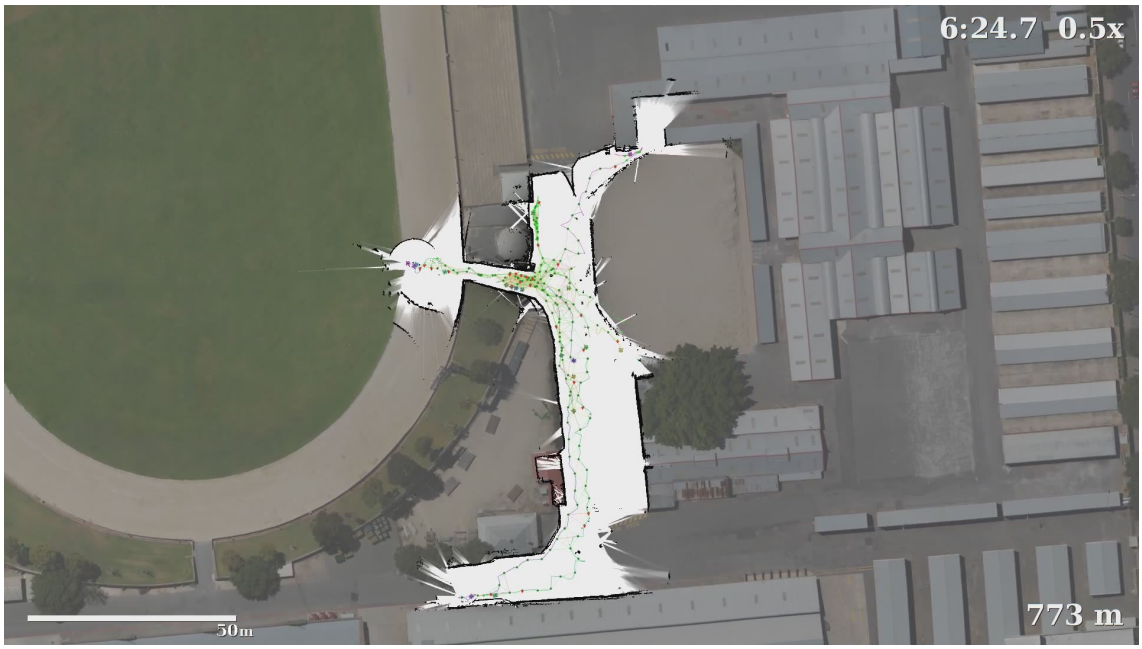
The surveyed ground-truth data is overlaid on the final gridmap shown in Figure 7.16 (e). By sampling 100 evenly spaced points across the phase, the RMS error was calculated as ±0.35 m. Time-based statistics are shown in Figure 7.17, while statistics plotted against the submap count are shown in Figure 7.18. Key measurements and statistics for the submaps, their constraints, and the optimization errors are shown using histograms in Figure 7.19. These results are summarized in Table 7.2 on page 196.



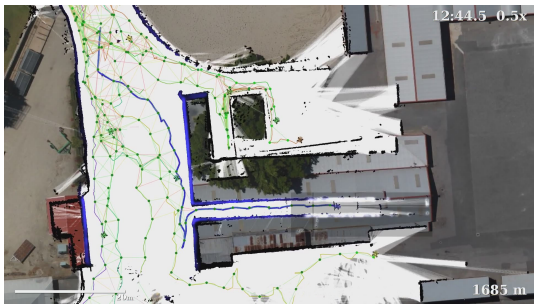
(a) 2m 6s: Initial global localization: 12 of 23 UGVs are connected in the pose graph. Odometry errors are already visible in the 20 m wide alley to the south.



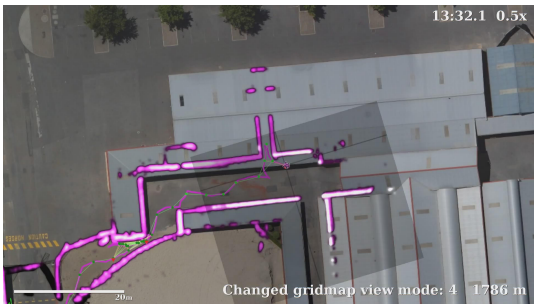
(b) 3m 53s: Two UGVs reach the end of the wide alley. Significant perceptual aliasing prevents submap matching from correcting the odometry drift.



(c) 6m 24s: Mapbuilder aligns the submaps at the end of the alley as additional submap matches are made. The UGVs explore the phase on five separate frontiers.

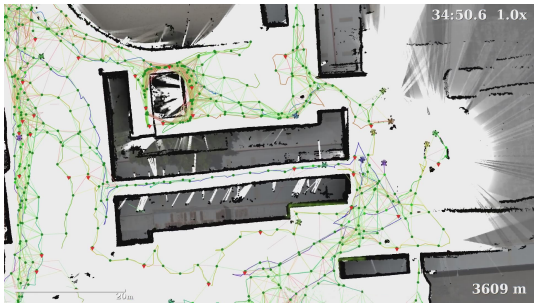


(d) 12m 44s: Three teams explore frontiers towards the east. UGV number 16 (blue) accumulates odometry errors in a narrow corridor and its path is adjusted.

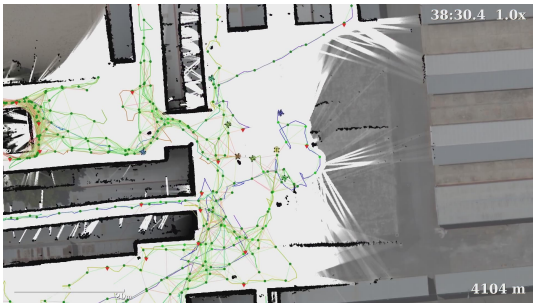


(e) 13m 32s: UGV number 18 (magenta) explores through the horse stables. The "x-ray" view is used to align its gridmap to the building eaves in the aerial imagery.

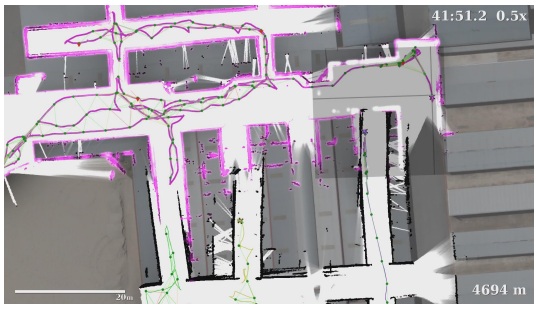
Figure 7.15. Results: MAGIC challenge Phase 2. 23 heterogeneous UGVs explore a 210×150 m mixed indoor and outdoor environment in 88 minutes. The pose graph is shown in green: dots represent submap poses, lines submap constraints and red triangles are ground-truth constraints. Click on the images to open the accompanying video at the corresponding time.



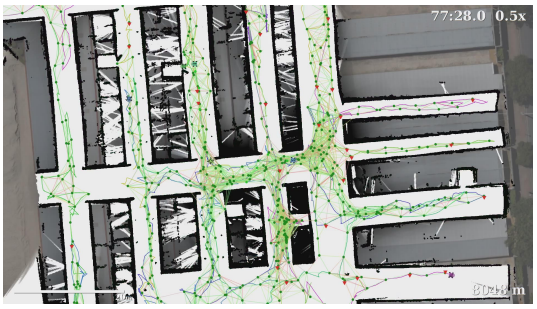
(a) 34m 50s: 10 UGVs converge in the Red Square (right) before closing a 210 m loop. The map distortions indicate that about 1 m of drift is corrected by the closure.



(b) 38m 30s: After the loop closure the UGV's explore north into the stables. Red Square is larger than the lidar sensors range and uncorrected odometry errors result.



(c) 41m 51s: One UGV (magenta) has explored the north horse stables. Several meters of drift is corrected to enable five loop closures, the first and largest is 330 m.



(d) 77m 28s: The density of nodes in the pose graph indicates the stables have been mapped thoroughly. The UGVs explore the corridors to the east.

(e) 92m 42s: Global gridmap with surveyed ground truth overlaid (magenta). Small colored marks in the white (free) areas are UGVs, while many black marks are round barrels used to support infrastructure. The RMS error calculated from 100 samples was ±0.35 meters.

Figure 7.16. Results: MAGIC challenge Phase 2. 23 heterogeneous UGVs explore a 210×150 m mixed indoor and outdoor environment in 88 minutes. Click on the images to open the accompanying video at the corresponding time.

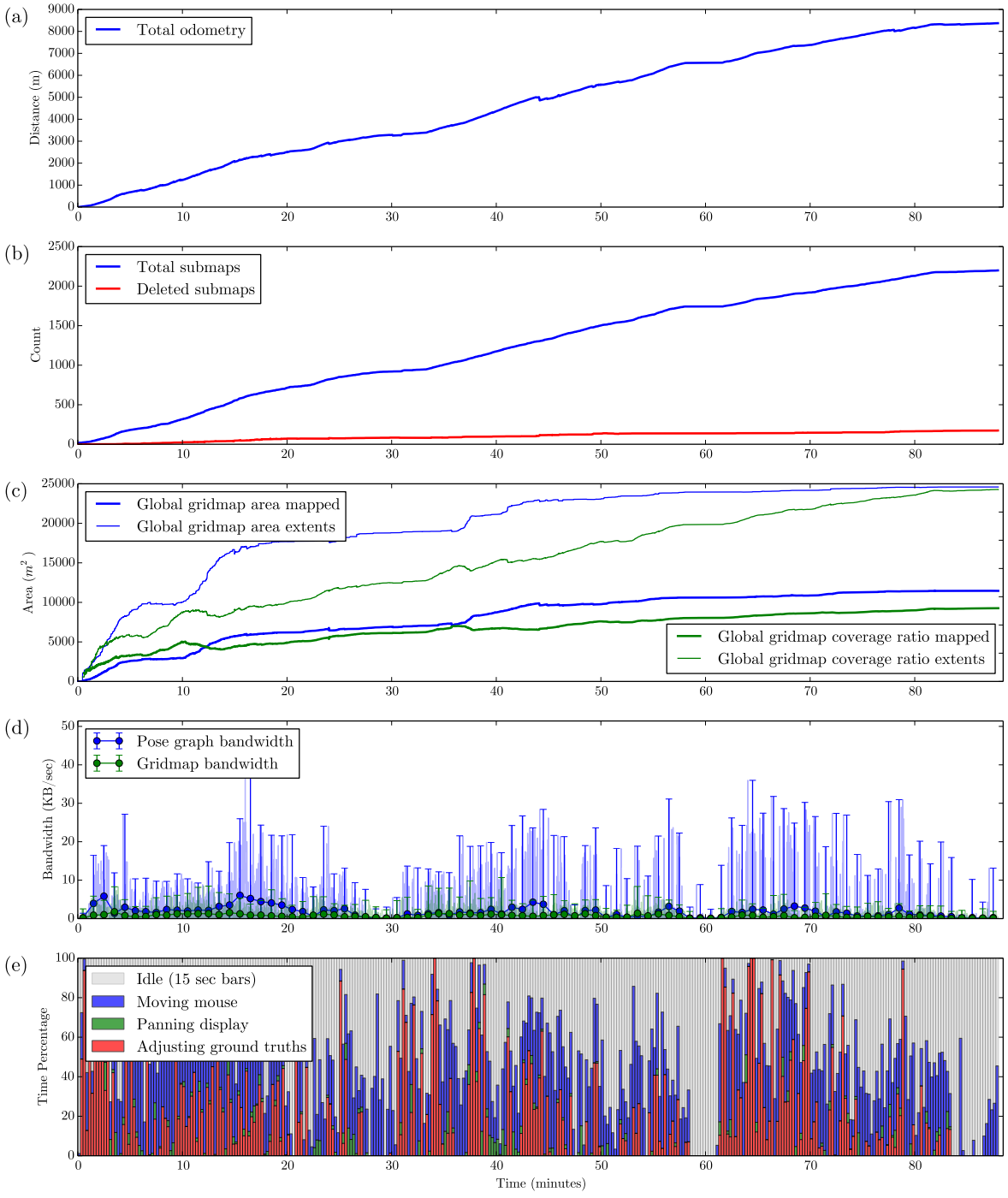


Figure 7.17. Results: MAGIC challenge Phase 2. 23 heterogeneous UGVs explore a 210×150 m mixed indoor and outdoor environment in 88 minutes. In these time-based plots: (a) shows the steady accumulation of 8,373 m of odometry. (b) shows 2,200 submaps being created, 174 of them were deleted, giving an average of 3.81 m of odometry per submap. (c) shows the global gridmap growing to 11,456 m^2 total area, while an average of 26 submaps overlap each part of the map. (d) shows the variations in communications bandwidth, averaging 2.4 KB/s for a total of 12.5 MB of MR-SLAM data for the entire dataset. (e) shows a histogram of operator interactions in 15 s bars. The operator spent 19.3 % of the time, or about 17 minutes, correcting for odometry errors.

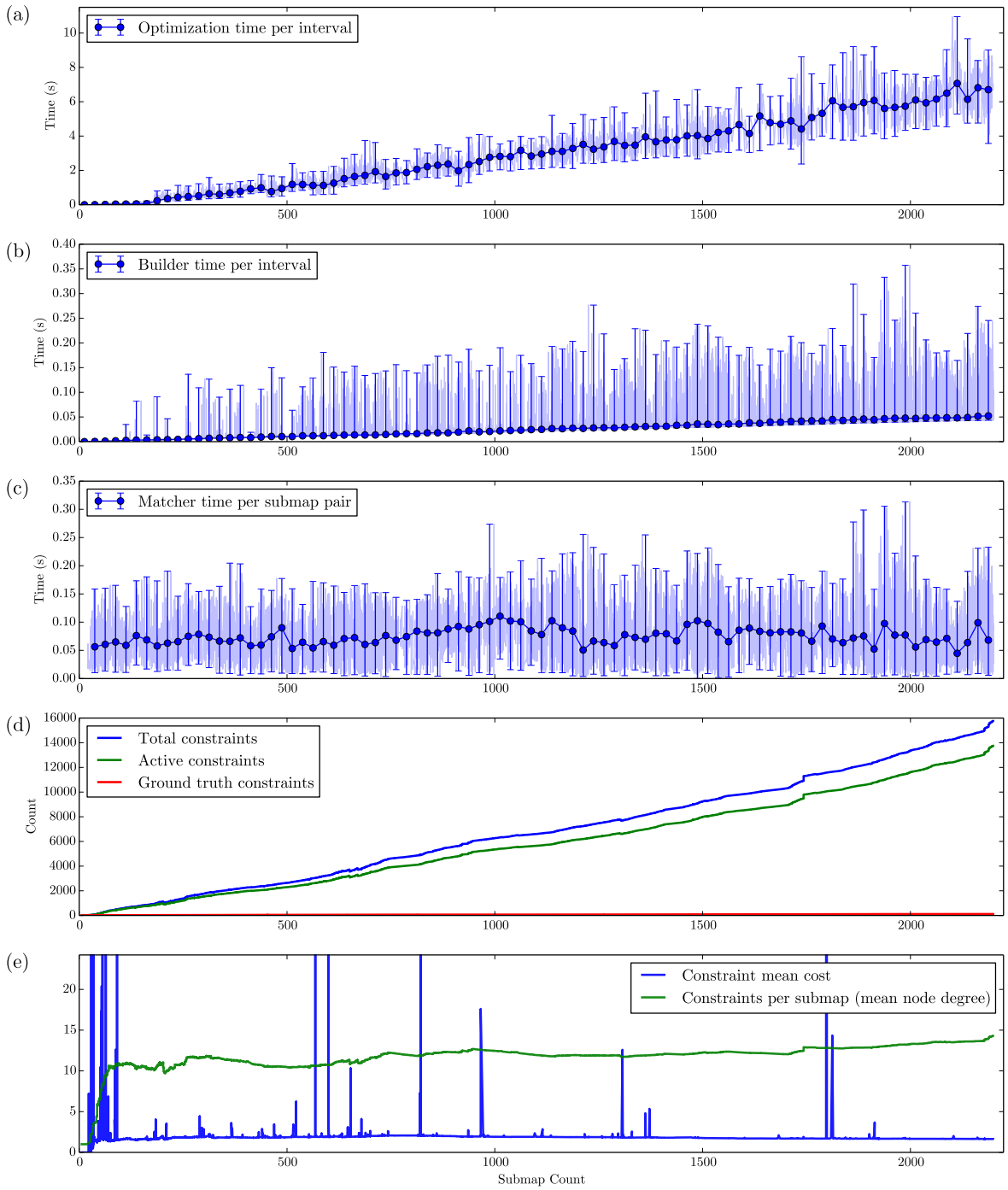


Figure 7.18. Results: MAGIC challenge Phase 2. Plots (a), (b) and (c) show the minimum, maximum and mean value per execution, with the total submap count on the abscissa. These statistics are noisy due to heavy contention for the CPU and GPU. (a) shows the optimization time per interval, which scales approximately linearly, at 2.4 seconds per 1,000 submaps. (b) shows the time the GPU spent building each 512×512 meter global gridmap; build times scale linearly at 0.022 seconds per 1,000 submaps. (c) shows the time the GPU spent matching submaps; an average of 0.074 seconds per match. (d) shows the growth in submap constraints; 86 % of the 15,757 constraints are active (not outliers). (e) shows the average number of constraints per submap, or the mean node degree, averaging about 14.8.

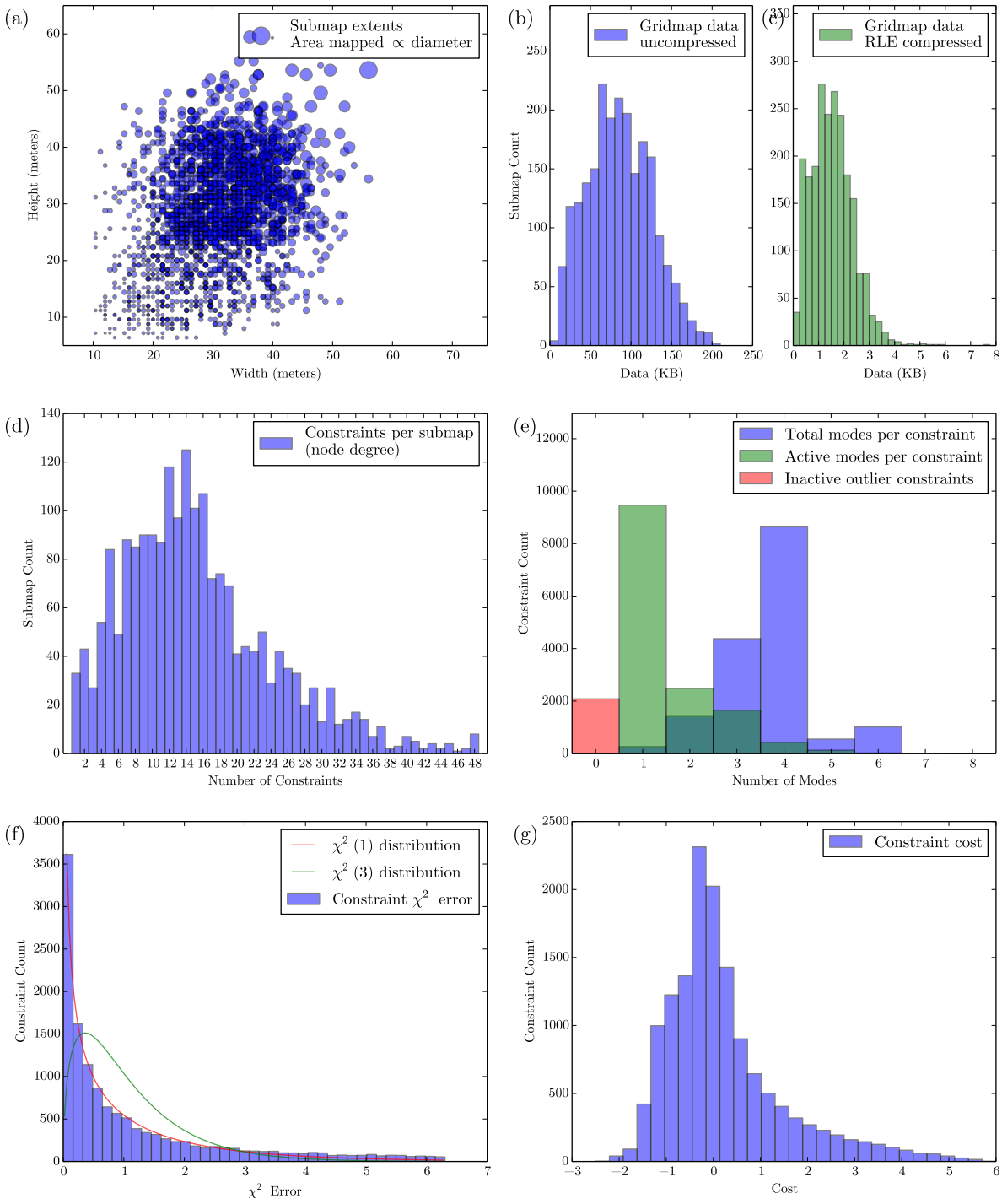


Figure 7.19. Results: MAGIC challenge Phase 2. Plot (a) shows the distribution of submap extents; the average submap is 30.6×28 meters, while 160 m^2 of each gridmap area is mapped (occupied or free cells). (b) shows the average submap is 87 KB uncompressed, while (c) shows that the RLE compression reduces the average submap to 1.5 KB. (d) shows the number of constraints for each submap, or the node degree, which averages 14.8. (e) shows both the total number of modes per constraint, and the number of these modes that were active (refer to Chapter 6). (f) shows the expected χ^2 distribution, and the actual error distribution. (g) shows the distribution of constraint costs (refer to Chapter 6 also).

	ORSC	Phase 1	Phase 2
UGV count	10	14	23
Phase extents (m)	80×40	210×140	210×150
Phase extents area (m^2)	3,200	29,400	31,500
Mapped area (free or occupied cells) (m^2)	2,640	9,067	11,456
Time total (minutes)	36	80	88
Odometry total (m)	3,170	6,153	8,373
Largest loop closure path length (m)	230	280	330
Gridmap accuracy RMS error to survey (m)	0.27	0.62	0.35
Submap count total	653	1,572	2,200
Submaps deleted	24	74	174
Submap extents mean width×height (m)	31×29.7	30.8×28	30.6×28
Submap extents area mean (m^2)	917.7	858.2	863.8
Mapped area per submap (free or occupied)	18.6 %	16.7 %	18.6 %
Odometry per submap mean (m)	5.04	4.10	3.81
Mapped area (free or occupied) coverage ratio	38	21.9	25.9
Constraint count total (pose graph edges)	4,674	10,649	15,757
- 1 mode active	56.7 %	59.5 %	57.4 %
- 2 modes active	10.8 %	13.1 %	15.0 %
- 3 modes active	9.6 %	9.0 %	10.0 %
- 4 modes active	1.5 %	1.8 %	2.6 %
- 5 modes active	0.3 %	0.4 %	0.8 %
- Inactive (outlier constraint)	20.3 %	14.8 %	12.6 %
- Ground-truth constraints	0.4 %	0.6 %	0.7 %
Constraints per submap (mean node degree)	14.3	14.7	14.8
Optimization mean time per interval per 1000 submap (s)	1.100	1.534	2.402
Builder mean time per interval per 1000 submap (s)	0.028	0.021	0.022
Matcher mean GPU time per submap pair (s)	0.078	0.075	0.074
Submap gridmap data uncompressed mean (KB)	94.6	85.9	86.8
Submap gridmap data RLE compressed mean (KB)	1.9	1.6	1.5
- compression ratio	50:1	54:1	58:1
Submap gridmap bandwidth average over phase (KB/s)	0.586	0.553	0.679
Pose graph bandwidth average over phase (KB/s)	0.589	1.667	1.741
Total bandwidth average over phase (KB/s)	1.18	2.22	2.42
Total bandwidth peak over phase (avg over 1 s) (KB/s)	22.63	36.2	52.8
Total data transmitted over phase (MB)	2.5	10.2	12.5
GUI operator time moving constraints	10.6 %	22.4 %	19.3 %
GUI operator time moving mouse	17.7 %	28.8 %	28.7 %
GUI operator time panning display	0.0 %	1.6 %	1.9 %
GUI operator idle time	71.7 %	47.2 %	50.1 %
GUI operator active time (minutes)	10.2	42.2	43.9

Table 7.2. Results: Mapbuilder performance on MAGIC datasets

7.3. Robust Multimodal Pose Graph Optimization

The GPU-based multimodal constraint generation algorithm (Section 5.4) and the COMBO technique (Section 6.4) were both used to process the combined MAGIC datasets in the previous section. While both of these contributions demonstrate global gridmaps converging accurately (discussed in Section 7.4.1), ground-truth data was not available to evaluate pose graph convergence. This section presents additional experimental results using COMBO to optimize multimodal distributions in $\mathbb{R}^1$, $\mathbb{R}^2$ and $\mathbb{SE}(2)$.

7.3.1. Multimodal Gaussians in $\mathbb{R}^1$

Figure 7.20 shows a simple 1-D problem that is modeled on the "slip or grip" problem described in Section 6.3.3. In this problem COMBO converges correctly to the nearest peak in the multimodal distribution $p(\mathbf{z}_i|\mathbf{x})$.

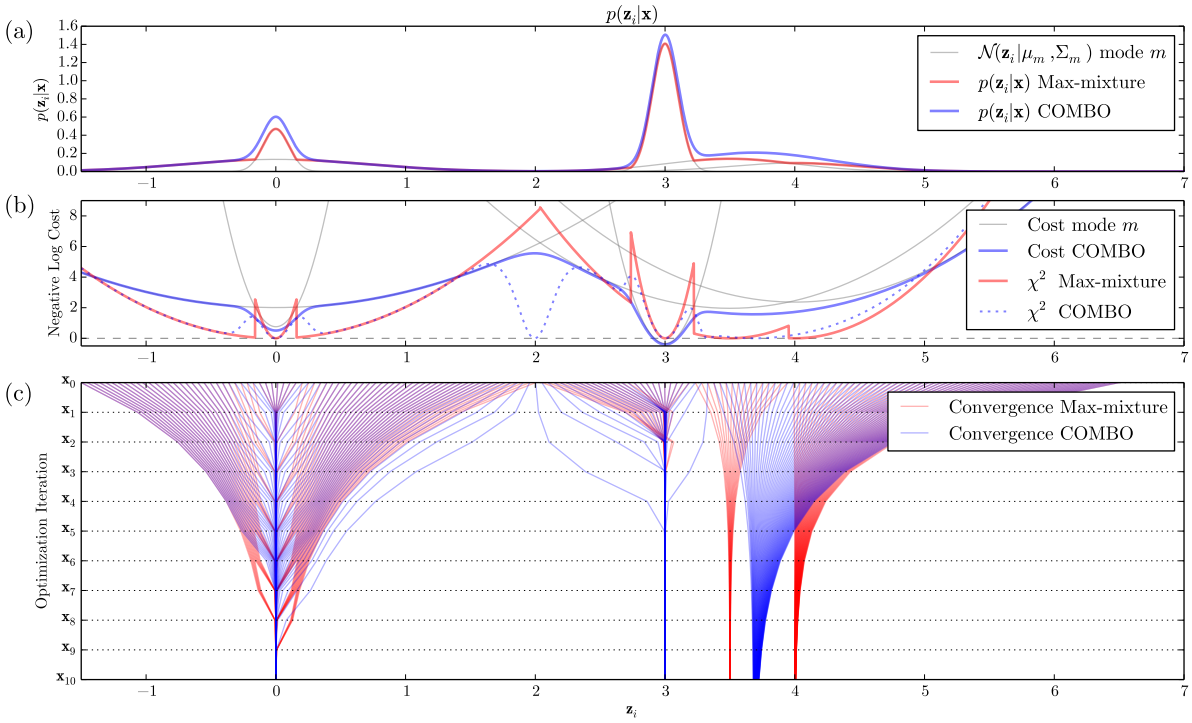


Figure 7.20. Results: COMBO Convergence in 1-D. A simple 1-D example modeled on the "slip or grip" odometry problem. (a) $p(\mathbf{z}_i|\mathbf{x})$ has five overlapping modes: the two tallest modes capture the possibility that either the robot's wheels slip ($\mathbf{z}_i = 0$) or the odometry is correct ($\mathbf{z}_i = 3$). Two additional modes overlap at $\mathbf{z}_i = 3.5$ and $\mathbf{z}_i = 4$, emulating weak likelihoods from scan matching. A wide null mode is added at $\mathbf{z}_i = 0$. (b) COMBO's negative log cost function is continuous and smooth, well suited for Levenberg-Marquardt. While the Max-Mixture cost function is discontinuous, this does not affect convergence with Gauss-Newton. (c) shows 10 iterations of gradient descent starting from a wide range of initial values for $\mathbf{x}_0$. COMBO converges to the nearest peak in $p(\mathbf{z}_i|\mathbf{x})$, as expected. Max-Mixtures converges to the nearest mode, which is undesirable for the overlapping modes around $\mathbf{z}_i = 3.7$.

Figure 7.21 shows another simple 1-D problem that validates COMBO's performance with widely varying mode weights. The tall, but low-weight, mode at $\mathbf{z}_i = 1$ was positioned to give $p(\mathbf{z}_i|\mathbf{x})$ a small additional peak. This type of distribution is observed in the MAGIC challenge datasets. COMBO converges correctly to the nearest peak.

7.3.2. Multimodal Gaussians in $\mathbb{R}^2$

It is difficult to confirm COMBO's convergence properties visually in dimensions higher than $\mathbb{R}^1$. In particular, multimodal $\mathbb{SE}(2)$ constraints are difficult to visualize because they define a space in $\mathbb{R}^3$ (in their minimal $[x, y, \phi]^T \in \mathfrak{se}(2)$ form). Figure 7.22 instead visualizes a single 2-D slice through a multimodal constraint. The slice corresponds to a fixed angle ϕ, and allows us to confirm COMBO's convergence properties in $\mathbb{R}^2$.

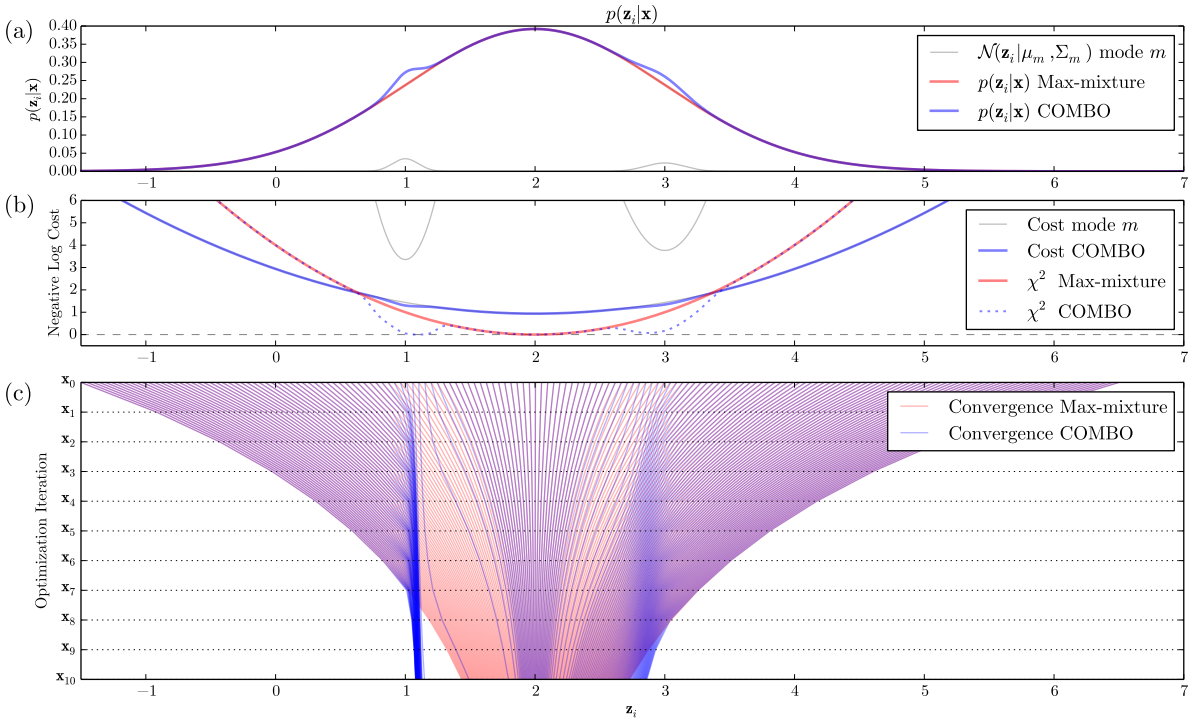
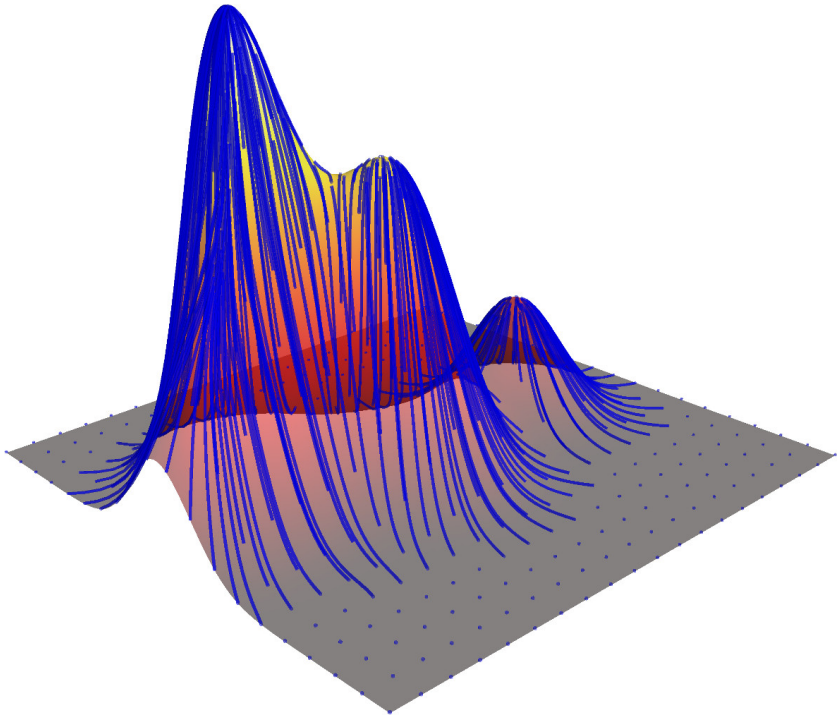
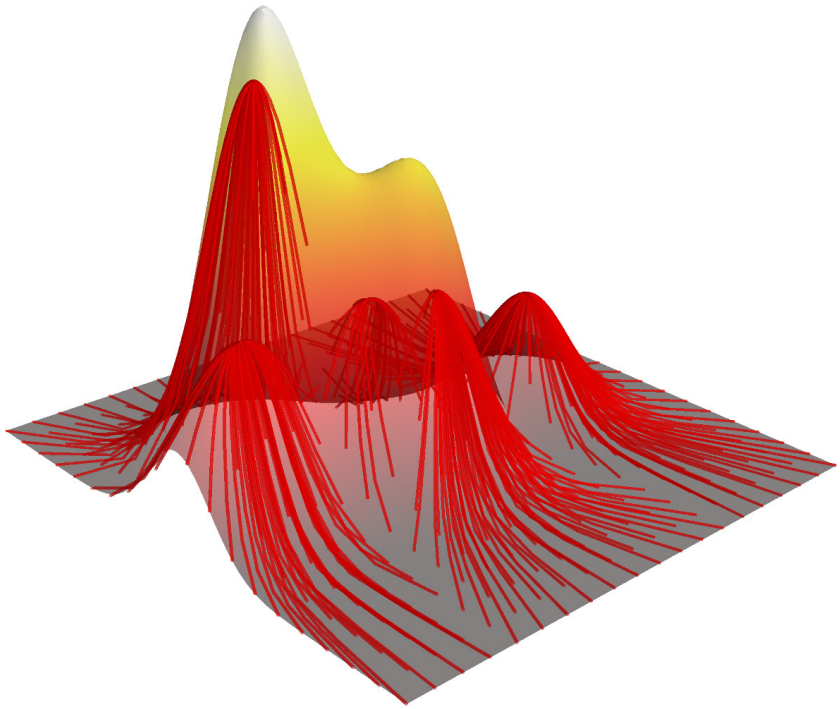


Figure 7.21. Results: COMBO Convergence in 1-D. A simple 1-D example that shows desirable convergence with a variety of mode weights. (a) $p(\mathbf{z}_i|\mathbf{x})$ has three overlapping modes: a dominant mode at $\mathbf{z}_i = 2$, flanked by two smaller low-weight modes at $\mathbf{z}_i = 1$ and $\mathbf{z}_i = 3$. The mode at $\mathbf{z}_i = 1$ is tall enough to give $p(\mathbf{z}_i|\mathbf{x})$ a small additional peak at $\mathbf{z}_i = 1.1$. (b) COMBO's negative log cost function is continuous and smooth, while the Max-Mixture cost function indicates that only the dominant mode is active. (c) shows 10 iterations of gradient descent starting from a wide range of initial values for $\mathbf{x}_0$. COMBO converges to the nearest peak in $p(\mathbf{z}_i|\mathbf{x})$, as expected. Max-Mixtures converges to the dominant mode only, which is the expected behavior.



(a) COMBO



(b) Max-Mixture

Figure 7.22. Results: COMBO Convergence in 2-D. The convergence properties of COMBO and Max-Mixtures can be visualized in $\mathbb{R}^2$ by taking a 2-D slice through a multimodal $\mathbb{SE}(2)$ constraint. For a constraint distribution $p(\mathbf{z}_i|\mathbf{x})$ with five modes, the height of the gray surface indicates the value of $p(\mathbf{z}_i|\mathbf{x})$ across an x, y slice for fixed angle ϕ. In (a) the COMBO constraint is active when its cost $\mathcal{C}_i \leq \mathcal{C}_{max}$. In this region COMBO converges quickly to nearest peak in $p(\mathbf{z}_i|\mathbf{x})$. For robustness, outside this region, $\mathcal{C}_i > \mathcal{C}_{max}$, the constraint remains inactive. (b) Max-Mixtures converges to the nearest mode's peak instead.

7.3.3. Multimodal Gaussians in $\mathbb{SE}(2)$

To evaluate COMBO's performance with multimodal $\mathbb{SE}(2)$ constraints I generated several synthetic pose graphs. This was necessary because there were no widely available multimodal datasets with both *ground truth* and *overlapping modes*. Other researchers have generated synthetic unimodal datasets modeled on Manhattan, New York [233, 236, 244, 302], where the simulated world is a fixed grid pattern modeled on city blocks. To simulate the perceptual aliasing problem discussed in Section 6.1.2, however, it was necessary to generate datasets with varying geometry, including parallel walls and corridors.

Inspired by the Manhattan datasets, I used real street map data from middle and lower Manhattan to provide spatially varying geometry. A team of robots are simulated driving repeatedly across the city to random destinations. The resulting pose graph becomes very dense in the center with over ten constraints per pose. Simulated submap matching is used to generate loop closures, such that around intersections and more complex geometry unimodal constraints are generated. Modeled on the corridor case in Figure 6.3, perceptual aliasing is simulated in areas where the streets are straight. This results in multimodal constraints with 2-7 overlapping modes, much like in Figure 6.6 on page 166. COMBO is designed to be an *incremental* optimization technique, as such the pose graph is initialized by perturbing the submap poses with σ=20 m of Gaussian nose; this could simulate noisy GPS readings between buildings. A handful of x, y priors are included to simulate operator interaction (ground-truth constraints) and to keep the pose graph aligned.

COMBO was evaluated against unimodal constraints and Olson et al.'s Max-Mixtures [57]. The three techniques were implemented using the g^{2}o library [238]. They all used Cholesky decomposition, while the unimodal and Max-Mixtures techniques used Gauss-Newton and COMBO used Levenberg-Marquardt for optimization (Section 2.5.4.9). Two datasets were generated: Manhattan Small and Manhattan Large, they are summarized in Table 7.3. Three optimization tests were performed for each of these datasets: unimodal, Max-Mixtures and COMBO. The unimodal constraints were generated to fit the type of distribution seen in Figure 6.3, resulting in highly-eccentric, but not rank deficient, modes.

The Manhattan Small dataset is a three block (315 meters) wide subset of the Manhattan dataset with over 7,400 meters of odometry. An average of five overlapping modes were created for each constraint. Figure 7.23 shows the configuration of the pose graph before and after optimization. Convergence and timing statistics are presented in Table 7.4.

The Manhattan Large dataset is a 4×4 km area centered on Washington Square Park, with over 177 km of accumulated odometry. An average of ten overlapping modes were created for each constraint. Figure 7.24 shows the configuration of the pose graph before and after optimization. Convergence and timing statistics are presented in Table 7.5.

	Manhattan Small	**Manhattan Large**
Submaps	370	8,889
Constraints	2,071	39,327
Ground truths	8	78
Mean node degree	11.2	8.8
Total odometry (m)	7,400	177,780

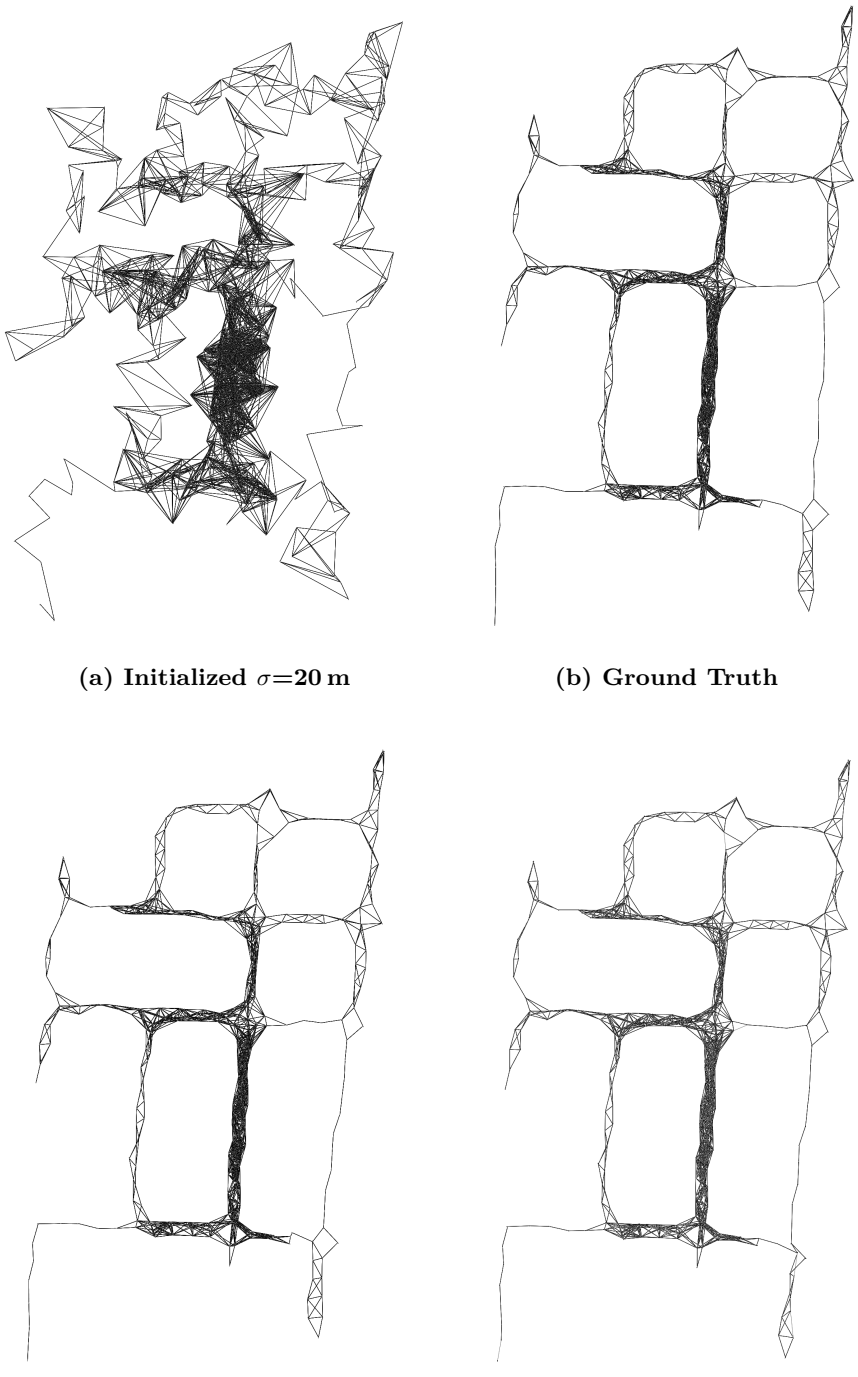
Table 7.3. Results: Simulated multimodal $\mathbb{SE}(2)$ pose graph dataset summary

	Initial	**Uni-modal**	**Max-Mixture**	**COMBO**
Pose error RMSE (m)	24.77	2.02	2.10	**1.17**
Pose error stdev (m)	12.78	2.71	4.02	**1.24**
Optimization time (s)	-	**0.029**	2.090	34.0
Time per iteration (s)	-	**0.003**	0.083	0.557
Optimization iterations	-	10	25	61

Table 7.4. Results: Convergence and timing for Manhattan Small dataset

	Initial	**Uni-modal**	**Max-Mixture**	**COMBO**
Pose error RMSE (m)	25.30	3.20	2.23	**2.20**
Pose error stdev (m)	13.17	5.08	3.93	**3.53**
Optimization time (s)	-	**1.762**	40.854	617.3
Time per iteration (s)	-	**0.057**	1.409	8.818
Optimization iterations	-	31	29	70

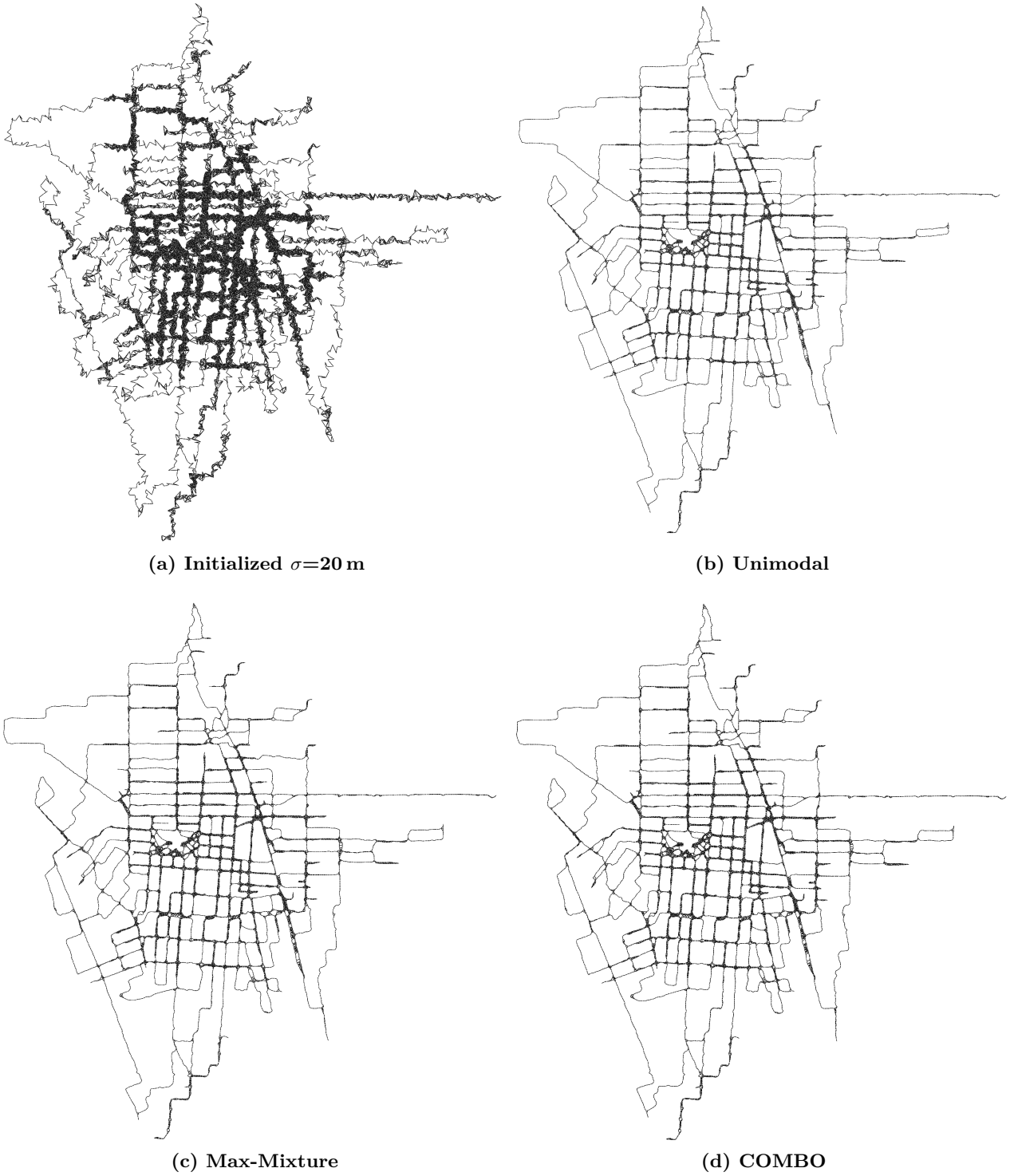
Table 7.5. Results: Convergence and timing for Manhattan Large dataset



(a) Initialized σ=20 m **(b) Ground Truth**

(c) Max-Mixture **(d) COMBO**

Figure 7.23. Results: Pose graphs for Manhattan Small dataset. A three block (315 meters) wide subset of the Manhattan dataset with 7,400 meters of odometry. The pose graph constraints are shown in black. (a) The $\mathbb{SE}(2)$ pose graph is initialized by perturbing with Gaussian noise (σ=20 m). (b) Ground truth used in RMSE calculations. The pose graphs (c) and (d) both converge to almost identical solutions.



(a) Initialized σ=20 m **(b) Unimodal**

(c) Max-Mixture **(d) COMBO**

Figure 7.24. Results: Pose graphs for Manhattan Large dataset. Centered on Washington Square Park, this 4×4 km area from the Manhattan dataset represents 177 km of accumulated odometry. The pose graph constraints are shown in black. (a) The $\mathbb{SE}(2)$ pose graph is initialized by perturbing with Gaussian noise (σ=20 m). The pose graphs (b) (c) and (d) converge to almost identical solutions after optimization with three different techniques.

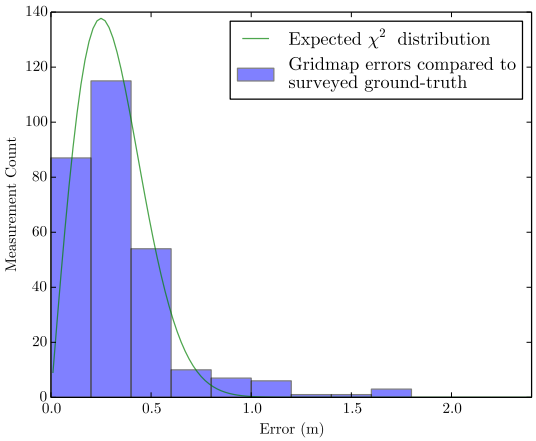
7.4. Discussion

The results presented in this chapter demonstrate my research work and contributions. The Mapbuilder system successfully performs real-time MR-SLAM at large scales. In the largest of the datasets, 23 UGVs accumulate over 8,300 meters of odometry while exploring 11,400 square meters of mixed indoor and outdoor environments. To the best of my knowledge, this is the largest large-scale MR-SLAM demonstration described in the literature (Section 3.3). This section highlights several key results and statistics from Table 7.2 that demonstrate Mapbuilder's accuracy, scalability and robustness. It also compares my research work to other recent large-scale MR-SLAM publications.

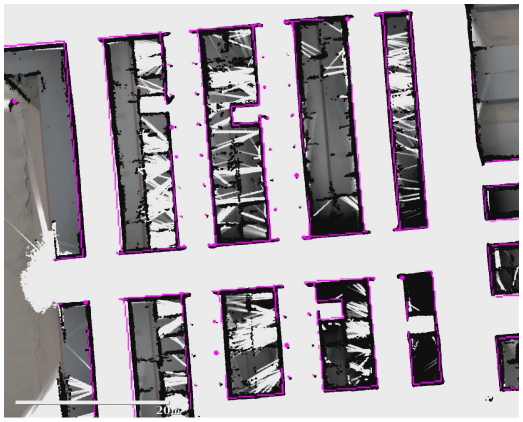
7.4.1. Accuracy

The root-mean-squared (RMS) error between the global gridmap output and the surveyed ground-truth data, is a concise way to capture the global-referenced mapping accuracy. The RMS errors were ±0.27 m, ±0.62 m and 0.35 m for the three datasets, which is 0.34 %, 0.30 % and 0.17 % of the extents of each dataset, respectively. To validate this error measurement, Figure 7.25 (a) plots the distribution of the errors measured. This is closer to the corresponding χ^2 distribution than anticipated.

More important than the global accuracy, however, is the local mapping accuracy. In real terms: whether the gridmap accurately represents obstacles in the environment so that the UGV's global navigation algorithms can avoid obstacles and navigate through doorways.



(a) Histogram of map errors between Mapbuilder's global gridmaps and the surveyed ground-truth. Errors measurements were spatially distributed using a grid. The $\chi^2(2)$ distribution is shown for σ=0.25 m.



(b) The stables from Phase 2 overlaid with ground-truth survey data (magenta). Four small rooms with open doors have been mapped, gaps allowed many other rooms to be partially mapped.

Figure 7.25. Global gridmap accuracy: Small obstacles such as posts and open doorways are clearly visible in Mapbuilder's global gridmaps.

Figure 7.25 (b) shows an enlarged section from Phase 2. The gridmap accurately represents the narrow corridors, roof-support poles and entrances into four small rooms. The pose graph has converged well, and there is no duplication or ghosting of obstacles in the environment.

An evaluation of the accuracy of a SLAM algorithm should include the exactness of the UGV localization also, however ground-truth localization data was not made available after the challenge. The natural coupling of mapping and localization uncertainty in SLAM, however implies that the UGVs' localization accuracy would have been close to the ±0.27 m, ±0.62 m and 0.35 m RMS mapping errors.

7.4.2. Scalability

The main factors limiting scalability are computation, memory and communications resources. The ideal "full" SLAM solution is combinatorial, and the problem is NP-complete (refer to Section 2.5). Thus the best *real-time* solution will use all of the available computational resources. While the UGVs are actively exploring, Mapbuilder utilizes nearly 100 % of the CPU's four cores and 100 % of the GPU's 1,664 cores. Contention for the CPU's time is complicated by the GUI screen capture and video encoding.

7.4.2.1. Computational Resources

Optimizer Task: the computational cost of on-line, incremental, pose-graph SLAM can vary considerably. The requirement for Mapbuilder's Optimizer was for it to scale linearly; for example, deployment in a 20 % larger environment should require 20 % more computational resources. Using the total submap count as a proxy for both total UGV odometry and area mapped, the results in Figure 7.8, Figure 7.13, Figure 7.18 and Table 7.2 suggest that it is approximately linear. A linear fit to the three datasets indicates that 1,000 submaps require on average 1.1, 1.5 and 2.4 seconds per optimization cycle.

Two factors affect this normalized computation time: the mean node degree (average number of constraints per submap) m, and the average number of Levenberg Marquardt (LM) iterations required for each cycle to converge. The mean node degree was discussed in Section 3.3.3; while Cholesky factorization is typically $\mathcal{O}(n^3)$ for n nodes (submaps), the Hessian is very sparse, and after optimum variable reordering this is reduced towards $\mathcal{O}(nm)$. Kaess discusses this complexity in Chapter 3 of his thesis [236]. For a constant mean node degree, the optimization time should scale in proportion to the submap count; this is observed in the plots.

Convergence in an LM-based pose graph optimization is a complex topic, especially when the cost surface is highly-concave. In an ideal world, the submap constraints would

perfectly describe the relative submap poses and the pose graph would easily converge to a solution where all the constraints agree. In reality, the constraint measurements are noisy and as more constraints are added they "fight" against each other resulting in highly complex cost surfaces. In this situation LM is likely to make more iterations of gradient descent ($\lambda \rightarrow \infty$) while it achieves very incremental cost improvements. The normalized average optimization cycle time for each dataset reflects the noise, ambiguities and perceptual aliasing experienced: the ORSC environment was well constrained (1.1 seconds/cycle), Phase 1 had minimal perceptual aliasing (1.5 seconds/cycle), while Phase 2 had large aliasing issues along the many alleyways (2.4 seconds/cycle).

The Optimizer scales linearly with the number of *unpinned* submaps. With the hybrid-decentralized design, most of the UGVs' pose graphs are pinned and their Optimizers estimate only a handful of free submaps in constant time. This effectively avoids redundant optimization, at the minimal bandwidth cost to transmit the pinned submaps' pose updates (described in next section).

To scale to 20,000 submaps (an order of magnitude larger than Phase 2) the optimization cycle time at the main GCS computer would becomes a bottleneck. MRS operations could tolerate a 24 second cycle time, however GUI interactions with the pose graph would be unfeasible with the slow the update rate. This could be mitigated by performing local, windowed, optimizations to maintain responsiveness for the operator while performing larger batch optimizations in parallel.

The design described in Section 4.4.2 could be improved in several areas: The already multi-threaded CPU code could be parallelized even further by unrolling loops with, for example, OpenMP. The Cholesky decomposition of the Hessian could be reused between invocations of the Optimizer task, since most of the time the pose graph structure remains constant. The handling of the λ step size in the LM algorithm could be improved; instead of restarting each cycle with λ close to zero (Gauss-Newton) it could save a few LM iterations by restarting with the previous value, reflecting the fact that the cost surface is only incrementally changing.

Builder Task: uses the GPU-based gridmap fusion algorithm described in Section 5.3. It uses minimal GPU resources, averaging 0.024 s to fuse 1,000 submaps. The algorithm scales linearly, as shown in Figure 7.8, Figure 7.13 and Figure 7.18. It took 0.048 s to render the whole of Phase 2, utilizing about 5 % of the GPU's processing for a 1 second cycle time. This solution would easily scale to 20,000 submaps, or an order of magnitude larger. By treating submaps as 2-D textures, the GPU-based fusion algorithm is likely near-optimum for modern GPUs.

Mapbuilder's design results in a large number of overlapping submaps— for the ORSC an average of 38 submaps covered each global gridmap cell. For the three datasets the

coverage ratios were 38, 22, and 26. While this requires the gridmap fusion algorithm to process large volumes of redundant data, the GPU-based algorithm is highly efficient. This efficiency is key to enabling distributed mapping, where redundant computation is performed by each participant to build local copies of the global gridmap.

Matcher Task: uses all of the spare GPU processing time to perform exhaustive correlations between pairs of submaps. An average of 0.075 s per submap pair was more than sufficient to generate matches for 23 UGVs in real-time. In a decentralized deployment the distributed instances of the Mapbuilder back-end would probably run on less powerful GPUs, however there would be more GPUs available. The distributed solution scales if each UGV includes enough processing power to match its own submaps. Only a few matches per submap are required to maintain a well-connected pose graph, and for UGV speeds under $3\,\mathrm{ms}^{-1}$ as few as one match per second would be sufficient.

7.4.2.2. Memory Resources

In the implementation of the Mapbuilder back-end used in these results, the bulk of the MR-SLAM data is stored and processed directly in the GPU's memory. The average gridmap size across the datasets is $870\,\mathrm{m}^2$, or 87,000 cells. Each submap is stored as a texture using two bytes per gridmap cell, thus gridmaps require on average 170 KB of GPU memory. All three MAGIC datasets combined require only ~735 MB of texture memory. The commodity GPU used to produce these results could easily store an order of magnitude more submaps in memory. With GPU-based texture compression enabled, over 80,000 submaps could easily be handled with minimal loss in output accuracy.

For the GPU-based gridmap fusion algorithm, the FBOs used to store the 500×500 meter global gridmaps used 100 MB of GPU memory. The commodity GPU used in these results has been tested fusing gridmaps up to 1640×1640 meters (1024 MB of GPU memory).

7.4.2.3. Communications Bandwidth

Central to Mapbuilder's efficient use of bandwidth is the fusion of hundreds of lidar scans into compact submap gridmaps. The subsequent quantization and compression of the gridmap data reduces the required bandwidth by more than 50 times. Despite the heavy compression, the finer gridmap details are preserved in the fused global gridmap output. At least every five seconds the UGV front-ends transmit compressed copies of their current submap gridmaps. With efficient DDS middleware and a broadcast-capable wireless network, at most 0.34 KB/s of bandwidth was required per UGV for submap gridmap data. Across the three datasets the total average gridmap data rates were 0.59 KB/s,

0.55 KB/s and 0.68 KB/s for all UGVs. These averages are considerably less because the UGVs spend different amounts of time sitting idle in each dataset.

It is difficult to model the bandwidth required to maintain the distributed pose graphs. Bandwidth requirements have been estimated here by counting the quantity of pose graph related messages that would have been sent: the bulk of which were `SubmapPose` messages. Events such as loop closures and cliques splitting and rejoining (large batches of `SubmapPose` messages) are unpredictable in real-world environments and depend on the UGV exploration strategy. Assuming ideal conditions, the three datasets require an average bandwidth of 0.59 KB/s, 1.67 KB/s and 1.74 KB/s for all pose graph related messages. Despite the pose graph growing over time, the volume of traffic appears relatively constant throughout each dataset, suggesting that loop closures are only updating submap poses in more localized areas.

The total bandwidth requirements for MR-SLAM appear to correlate most strongly with the rate of submap creation, hence proportional to the number of UGVs and their average velocity. This traffic is often irregular and occurs in bursts, thus it is important to consider the peak bandwidth required over any small period. For the three datasets the peak total bandwidth over any one second period was 22.6 KB/s, 36.2 KB/s and 52.8 KB/s— a small fraction of the 54 MB/s bandwidth that 802.11g Wi-Fi theoretically provides.

7.4.3. Robustness

The Mapbuilder back-end design is robust in several ways. The low RMS errors in Section 7.4.1 indicates that the map converged in a manner that is tolerating both Gaussian odometry noise and small non-Gaussian errors. This is most evident when comparing the results presented here with earlier implementations of Mapbuilder [66]. Earlier implementations would frequently fail to converge, with errors varying from minor visual distortions to completely unusable maps. The contributions described in Chapter 6 are mostly responsible for this improved convergence.

The Local SLAM front-end (Section 4.4.1) uses RANSAC to identify outliers caused by moving objects in the environment. This is particularly important in sparse environments where the moving object may return more lidar measurements than the environment itself. Slow or objects that move intermittently are still likely to become "baked" into occupied cells in submaps. These artifacts are robustly handled by the MR-SLAM back-end instead.

Penn's front-end produced several artifacts, including trails behind walking human OOIs. These are shown in Figure 7.26, where two submaps have large clusters of occupied cells that are outliers. Mapbuilder's multimodal constraints capture several large erroneous modes, along with the correct one. In this instance the pose graph converges to the correct configuration and correctly ignores the outlier modes.

The binary Bayes filtering algorithm used to fuse global gridmaps (Section 5.3) is effective at filtering outliers out of the global gridmaps. The corrupted submaps from Figure 7.26 can be seen in the south-east corner of the Phase 2 gridmap in Figure 7.16 (b) on page 192; the black outliers are filtered in the final gridmap in Figure 7.16 (e) demonstrating robustness to slow and intermittently moving objects.

7.4.4. Perceptual Aliasing

In the datasets recorded by all three UGV designs (WAMbot, TM and Penn's), the most significant SLAM errors were caused by the perceptual aliasing ambiguities described in Section 6.1.2. These ambiguities occur frequently in the challenge datasets, particularly in large sparse areas, such as the 100×25 meter car park in Phase 1 (Figure 7.11 (e)), and very frequently in straight featureless corridors, such as the 20 meter wide alleyways in Phase 2 (Figure 7.15 (b)).

In sparse open areas, there is very little that can be done, other than to dead-reckon with odometry and hope that the wheels slippage is both minimal and mostly Gaussian. In the case of featureless corridors, the multimodal pose graph constraints described in Section 6.4 allow submaps with spatial information perpendicular to the walls to be integrated.

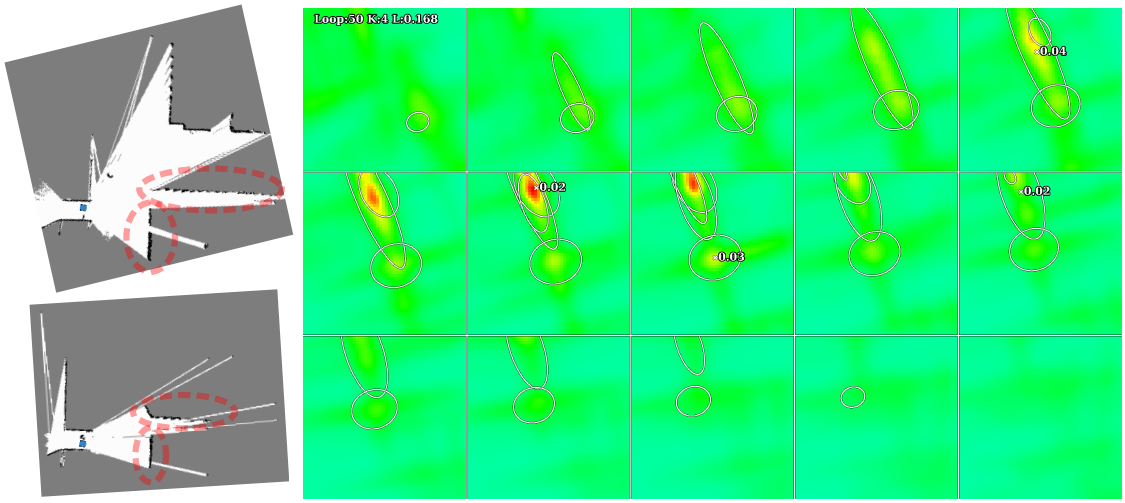


Figure 7.26. Multimodal constraint output: robustness to moving objects. Correlation results for an area in Phase 1. An overlapping pair of submaps is shown (left), along with 15 slices through their $(x, y, \phi) \in \mathfrak{se}(2)$ constraint likelihood volume (right). Each slice represents the 2-D correlation between the submaps varied over ±3 m translation in x and y, at a fixed angular rotation, e.g. the middle row corresponds to -6, -3, 0, 3 and 6 degrees. A Gaussian mixture has been fit to the likelihood volume, modes are indicated by their 3-σ covariances (2-D ellipses). Due to occlusions, only a small fraction of the occupied cells (black) in this submap pair actually overlap. An array of columns in the environment dominate the matchers output. The two submaps are corrupted by a person walking past (outlier cells circled in red). This example demonstrates robustness to moving objects: the correct mode (labeled 0.03) is found, even though it is overwhelmed by a mixture of three other modes that occur due to the moving object's trails.

7.4.5. Usability and Cognitive Load

The Mapbuilder GUI, described in Section 4.4.3, was designed to enable "human-in-the-loop" graph-based MR-SLAM. It was essential for processing these datasets because of the frequent non-Gaussian errors. The time and cognitive loading required for these operator interactions was of particular interest to the MAGIC challenge organizers, particularly for multi-hour ISR deployments [52].

Similar to Olson et al. [53], I use the GUI interaction times as a proxy for cognitive load. Histograms for each of the datasets, shown in Figure 7.7, Figure 7.12 and Figure 7.17, provide a breakdown of how the operator's time was spent. It is interesting to note that there are many periods of several minutes in all of the datasets where Mapbuilder ran without any operator interaction.

The types of operations that corrected non-Gaussian odometry errors, essential to ensuring the pose graph convergences, were adjusting ground-truth constraints, and deleting corrupted submaps. The time spend adjusting ground-truth constraints is clearly indicated in the plots, however the delete operations requires only a brief key press. Much of the operator's cognitive attention is applied to observing and identifying badly localized UGVs and corrupt submaps; this time is captured in the "moving mouse" statistic in the plots. These operations are better demonstrated in the videos available on-line at http://reid.ai/thesis.

While processing the three datasets, the operator spent 28.3 %, 52.8 %, and 49.9 % of the time actively looking for, or correcting, issues. It is difficult to make direct comparisons with Olson et al.'s work [53], since the results presented here include Penn's UGVs. The complete Phase 2 dataset, with 23 UGVs, took 44 minutes of operator time to process with Mapbuilder, while Olson et al. report using 70 minutes of operator time [53] with 14 UGVs. Olson et al. report manually adding a new loop-closure constraint on average every 24 seconds; in Mapbuilder all loop closures are generated automatically. The equivalent corrective operation in Mapbuilder is the addition of ground-truth constraints, of which one was added on average every 80 seconds.

One of the high-level goals of the MAGIC challenge was to encourage advances in MRS autonomy that would allow operators to control increasingly large teams of UGVs. It is unclear whether Mapbuilder's 2-D "Google Maps" styled GUI provides an optimum interface or whether 3-D interfaces, such as TM's SAGE display with its "fog of war" [53], increase the operators' situational awareness and efficiency. It is also unclear how the operator's cognitive load scales with the number of UGVs. Humphrey et al. suggest that this scaling is sub-linear [346]. There is considerable scope in this area for large-scale MR-SLAM usability studies; my research work has barely scratched the surface.

7.4.6. Comparison to Recent Work

This section compares my research work and the Mapbuilder system against the recent MR-SLAM systems were reviewed in Section 3.2.3. Table 3.2 on page 93 summarizes the most comparable systems and highlights and compares their capabilities. The most comparable MR-SLAM systems are TM's, described by Olson et al. [53], and Penn's, described by Lee et al. [311]; this section focuses on key differences between their work and mine.

Architecture: both TM and Penn describe pose graph architectures with SLAM front-ends on each UGV, and centralized MR-SLAM back-ends [53, 311]. They both use DCA (Section 3.2.2) for limited UGV autonomy. Loop closures and UGV pose updates are not shared by the centralized back-end, however, and as such each UGV's ability to navigate independently is limited by the rate at which odometry errors accumulate. In contrast, Mapbuilder's distributed and hybrid-decentralized design allows small cliques of UGVs to navigate in their own local copies of the global map and perform the entire MR-SLAM back-end role independent of the GCS. Table 3.1 on page 88 summarizes how the proposed architecture compares against other architectures.

SLAM front-end: TM's front-end produced a 2-D "maplet" for every 1.25 second sweep of its UGV's lidar scanner, while Penn's front-end built a single gridmap with a sliding window (they are both described in Appendix A). The Local SLAM front-end design (Section 4.4.1) worked well during testing, however was not demonstrated extensively in the same environments due to hardware issues [64]. The most significant differences between the three front-ends can be attributed to the arrangement of lidar scanners, such as fixed, sweeping or nodding. Figure 7.27 shows example submaps for the three front-ends after the reprocessed datasets had been turned into submaps. The WAMbot and Penn submaps have similar depth-of-fields due to the fixed horizontal lidar; Penn's lidar was mounted 0.3 meters higher, explaining some variation. TM's nodding horizontal lidar frequently produced more distant returns and larger submaps.

Data reduction: before transmitting gridmaps, TM's front-end uses a custom predictive compression method that achieved a 21:1 compression ratio; they merge free cells with unknown cells and transmit binary gridmaps [53]. Mapbuilder's submaps are considerably larger and they include all three cell states (free, occupied, unknown), which results in larger gridmaps. For comparison, the compression ratio in this work was about 14:1, which means the design transmits about 50 % more data per square meter of gridmap, however, it transmits gridmaps less often and the back-end is able to differentiate between free and unknown cells. Penn's front-end transmitted compressed Matlab objects that contained lists of cell changes; it is difficult to make any quantitative comparisons to their design.

MR-SLAM back-end: TM's pose graph optimization applies exact minimum degree variable reordering to the Gauss-Newton normal equation (Section 2.5.4.7) before using Cholesky decomposition to solve for each optimization step [53]. Olson et al. implement a graph simplification algorithm that identifies redundant constraints and removes them; they report using this several times in each of their MAGIC datasets. Table 7.6 compares TM and Mapbuilder's mean node degrees and optimization times. Mapbuilder's mean node degree for Phase 1 is about three times higher, suggesting that TM's graph simplification approach is effective. While Olson et al. did not report the total number of constraints, they can be estimated from the mean node degree; interestingly, for Phase 1 Mapbuilder's submapping approach and TM's graph simplification both resulted in about 11,000 constraints. In Phase 1, TM's graph optimization is about six times faster than Mapbuilder, however their times reported in [53] used unimodal constraints while Mapbuilder implements the multimodal constraints described in Chapter 6. When moving to Phase 2, TM's mean node degree was halved, however Mapbuilder's mean node degree remained similar. This can be explained by the addition of Penn's dataset with nine UGVs, and the resulting increase in submap density. Despite the interval between graph update cycles being higher in Mapbuilder, the operator's interaction times were still considerably less (Section 7.4.5). To the best of my knowledge, Penn's back-end was only designed and not implemented.

Constraint generation: TM's front-end reduces each sweep from their nodding lidar scanners into small binary gridmap scans. Olson describes a CPU-based scan correlation approach similar to [282] that searches for alignments between pairs of gridmaps and creates unimodal Gaussian constraints. In [344] they describe a newer implementation that uses a multi-resolution approach to speed the search; they report unimodal matches taking 0.002 s. This rate of matching is essential for highly-centralized MR-SLAM where a single centralized CPU performs all back-end computation, and when each UGV generates a new gridmap every 1.25 seconds. The GPU-based constraint generation described in

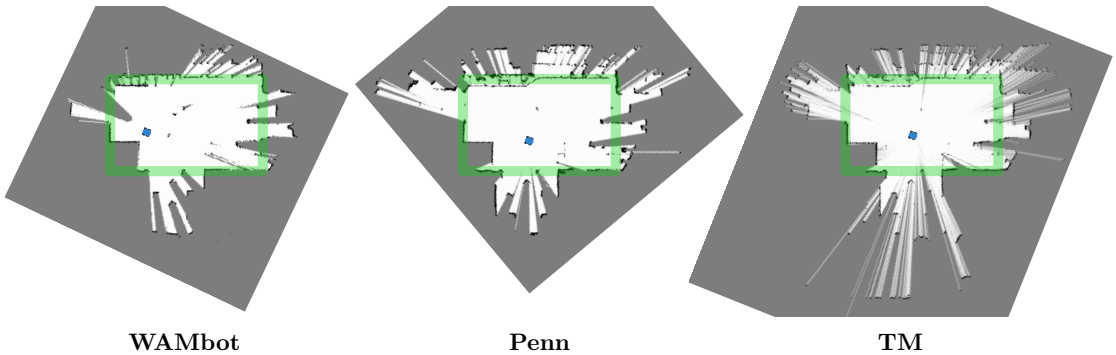


Figure 7.27. Sample submaps: Captured from the three different UGVs around the same location in the Old Ram Shed Challenge (ORSC). For reference, the 20×12 meter boundary of first room is highlighted in green.

this work (Section 5.4) is an order of magnitude slower, however it is generating more-rich multimodal constraints, and its computation is typically distributed across all computers, including UGVs. Furthermore, the submapping technique described here produces less submaps, thus requiring less matches also. Olson's multi-resolution approach [344] could be modified to generate multimodal constraints, however it is unclear how much slower it would be.

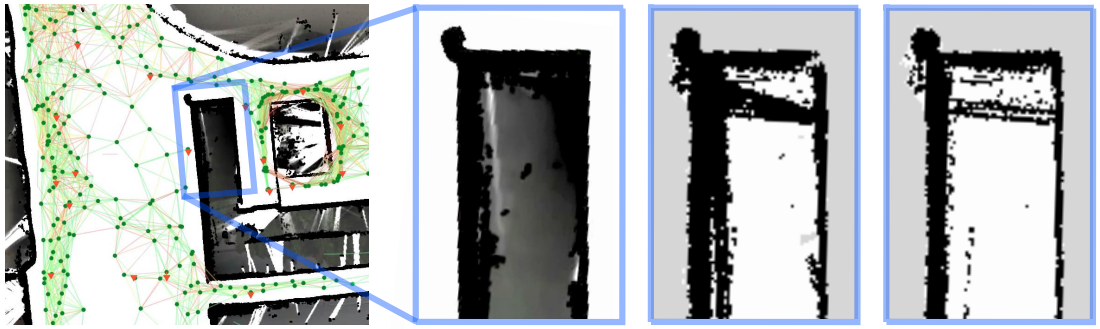
Global map building: TM have described improvements to their occupancy gridmap building algorithm since the MAGIC challenge. In [56] Strom and Olson described how their naive rasterization approaches were not fast enough for large-scale MR-SLAM. Their new approach uses a caching scheme that exploits the fact that most of the time only small parts of the global gridmap are changing. A separate node covering analysis allows them to avoid rasterizing redundant gridmaps that are obscured by others while removing moving objects with a voting scheme. Combined, their two techniques reduce the average time it takes to build the complete Phase 2 dataset from 5.62 s for their naive solution, down to 1.24 s (with maximums of 6.94 s and 3.6 s, respectively).

In comparison, Mapbuilder takes 0.048 seconds to build Phase 2, including the additional submaps from Penn's nine UGVs and while implementing a full binary Bayes filter. TM's CPU-based results were generated on a laptop— while their algorithms are efficient, it is difficult to compete with a massively-parallelized algorithm running on thousands of GPU cores. To compare more directly, a laptop with the specifications they report might have a GPU that is 16 times slower; extrapolating Mapbuilder's performance, it would take 0.47 seconds per build for their 14 UGVs (the approach described here is about 2.6 times faster). The limited availability of ground-truth data makes it difficult to compare the fine-scale accuracy of the global gridmaps. Figure 7.28 compares a small sample of Mapbuilder's global gridmap with TM's from [56].

TM's MR-SLAM system is the most comparable to Mapbuilder and my research work [53, 54], and as such it has been referred to many times in this thesis. The first description of their MR-SLAM system [55], was published concurrently with earlier versions of Mapbuilder in 2010 [68]. Both systems shares several design aspects including pose graphs

		TM	Mapbuilder
Phase 1	**Nodes/submaps**	3,876	1,572
	Mean node degree	5.7	14.7
	Optimization time	0.756	4.55
Phase 2	**Nodes/submaps**	4,265	2,200
	Mean node degree	2.96	14.8
	Optimization time	0.249	6.72

Table 7.6. Discussion: Pose graph optimization timing comparison



(a) Mapbuilder (b) TM: Naive (c) TM: Cover

Figure 7.28. Global occupancy gridmap comparison: Comparison between outputs from Mapbuilder and Strom and Olson's approach [56]. Mapbuilder's output from the start of Phase 2 is shown for context (left). The three insets show enlargements of a barrel at the corner of a 5 meter wide building. (a) shows Mapbuilder's output, note the aerial map is visible through the inside of the building. (b) is TM's naive rasterization approach, and (c) is their node covering approach.

and occupancy gridmaps. The most significant difference is in the high-level architecture: Mapbuilder implements a hybrid-decentralized and distributed approach, while TM uses DCA with a centralized server. Mapbuilder provides each UGV with a copy of the global gridmap, allowing considerably more autonomy and distributed processing. The performance of both systems has improved considerably since 2010, the most significant improvements resulting from the addition of robust multimodal constraints, discussed in the next section.

7.4.7. Multimodal Pose Graph Optimization

Chapter 6 described the COMBO technique for optimizing multimodal pose graphs. Convergence properties have been demonstrated with real-world datasets (Section 7.2), with minimal problems in $\mathbb{R}^1$ and $\mathbb{R}^2$, and with simulated datasets in $\mathbb{SE}(2)$ that provide ground-truth information (Section 7.3). The most comparable work to COMBO is Olson and Agarwal's Max-Mixtures [57, 299], which is discussed and compared in this section.

For non-overlapping modes, similar to the multimodal pose graphs considered in [117, 343], Figure 6.5 on page 165 shows COMBO correctly converging to the nearest peak in the likelihood distribution. It converges quadratically at the same rate as Max-Mixtures for most of the distribution. For the alleyway example in Figure 6.6 on page 166, which exhibits a flat-topped likelihood distribution, Max-Mixtures quickly converges to the nearest peak. This exerts an undesirable quantizing effect on the optimization, while COMBO correctly does not exert any influence in the flat part of this distribution. The "slip or grip" problem, shown in Figure 7.20 on page 197, shows COMBO having more desirable convergence also.

COMBO and Max-Mixtures use different approaches to represent constraint probability distributions: COMBO is a true Gaussian mixture, while Max-Mixtures approximates

a mixture using the max function as a selector. By design, Max-Mixtures converges quickly and correctly when modes are either non-overlapping, or they are exactly overlapping (Olson et al. describe robust constraints that overlap in [57]). Many complex constraint probability distributions are better approximated by Gaussian mixtures, however, where overlapping modes produce smooth distributions such as the alleyway example in Figure 6.6 on page 166, or the 2-D distribution in Figure 7.22 on page 199. If a max-mixture is created from the modes of a Gaussian mixture, i.e. the same overlapping Gaussian modes, it is certain to have an undesirable effect on convergence.

Section 6.2.2, discusses how *any* distribution can be approximated with a sufficient number Gaussian modes; the same is true for max-mixtures. A complex "lumpy" distribution like Figure 6.1 on page 152 might be well-approximated by a max-mixture having a few dozen modes, however a distribution like the "flat top" in Figure 6.3 on page 155 would need many more. A key observation in Section 6.1.2.2 is that constraint distributions will often have flat areas like Figure 6.6 on page 166, that are difficult to model with unimodal constraints because they are highly-eccentric. COMBO's blending technique is ideal for these cases, while Max-Mixtures is likely to require many more modes.

The experiments in Section 7.3 were performed to evaluate COMBO against Max-Mixtures and unimodal Gaussians in exactly this case. For the Manhattan Small dataset, Table 7.4 on page 201 shows COMBO producing much lower pose errors than both unimodal constraints and Max-Mixtures. The unimodal constraints had a RMSE 73 % higher than COMBO, while Max-Mixture's RMSE was 79 % higher. For this dataset, however, the unimodal constraints are optimized almost 200 times faster than COMBO, while Max-Mixtures 6.7 times faster. For the Manhattan Large dataset, in Figure 7.24 on page 203, COMBO is shown producing lower pose errors also, however its lead over Max-Mixtures and unimodal constraints was less— the unimodal constraints' RMSE was 45 % higher, while Max-Mixtures was only 1.5 % higher. The unimodal constraints optimized almost 155 times faster than COMBO, while Max-Mixtures execute 6.3 times faster.

The implementation of COMBO used in these tests went through a similar level of code optimization as the Max-Mixtures implementation. At about six times slower, the computational cost of COMBO's mode blending is not negligible. The test cases presented here, however, are almost pathological, since almost every constraint is blending multiple modes continuously, and every constraint is near a stationary point. Section 6.4.4 describes a fast Jacobian approximation that is rarely used in these simulated datasets.

In a small, real-world, test using the Mapbuilder back-end for incremental SLAM (with 62 submaps, 549 constraints and an average of 5.5 modes per constraint) each optimization cycle took 21.2 ms for COMBO (an average of 2.3 modes were active per constraint), and 4.4 ms for Max-Mixtures. While Max-Mixtures is 4.8 times faster per iteration, in this

case it was taking five iterations to converge. Here the Gauss-Newton optimization used in Max-Mixtures was oscillating between solutions, while COMBO's Levenberg-Marquardt optimization (with its variable step size) was converging in a single iteration.

In practice, COMBO is designed to be used on-line in an incremental pose graph optimization. In on-line SLAM, the batch optimization times shown in Table 7.3 and Table 7.4 on page 201 would be amortized over the entire deployment time. Furthermore, in normal operations the pose graph would remain close to the optimum, which would require less optimizer cycles total.

For pose graphs with complex constraint distributions, COMBO assists convergence by providing better approximations to these distributions. Using overlapping multimodal Gaussian modes, it produces smoother paths through the cost function (Section 6.4.3). In the case of non-overlapping multimodal constraints with discrete modes, such as those considered by [57, 117], COMBO switches between modes and executes as efficiently as Max-Mixtures.

The source code and scripts to reproduce the experiments in Section 7.3 are available on-line at http://reid.ai/thesis. This includes a C++ implementation of COMBO that integrates into the g^{2}o library [238], and Python scripts to generate large-scale pose graph datasets for cities using data from OpenStreetMap.

8 Conclusion

To conclude this thesis, Section 8.1 summarizes the multi-robot SLAM problem and the architectural aspects of the problem that my research work addressed. Section 8.2 lists the research contributions described in this thesis while highlighting their significance to the fields of multi-robot SLAM and multi-robot systems. Section 8.3 outlines limitations in my research approach and identifies interesting directions for future research work. Finally, Section 8.4 considers the broader implications of my research work and its applications for the next generation of multi-robot systems.

8.1. Summary

On-line localization and mapping are fundamental requirements for teams of mobile robots to cooperate in everyday situations, such as inside our homes or along urban streets, particularly when these environments have either not been mapped or they change over time. From emergency search and rescue, to precision agriculture and mining, there are many applications where we would like to deploy teams of robots quickly, without *a priori* maps, and only using noisy on-board sensors.

When localization and mapping are intrinsically coupled— i.e. when a robot needs a map of its environment to localize itself using on-board sensors, however to build the map incrementally it needs to estimate its localization— we describe this as the SLAM problem. In Chapter 2 I reviewed several broad classes of algorithms that solve the *single-robot* SLAM problem. While each of these algorithms have been adapted to solve the *multi-robot* SLAM problem, most adaptations have ignored the limitations of real-world

communications and proposed *highly-centralized* solutions. For teams of robots that rely on centralized servers, wireless communications become a single point of failure— it is not hard to imagine the potential dangers and frustrations of a robot team that freezes each time its wireless network drops out.

The state-of-the-art MR-SLAM systems demonstrated at the MAGIC challenge used centralized servers, and while their decoupled centralized architectures allowed individual robots to operate for brief periods, their robots did not share SLAM data or perform loop closures, which limited their autonomy. Several *decentralized* MR-SLAM architectures have been described in the literature, however none have been demonstrated at large scales while maintaining the map fidelity required for autonomous deployments both indoors and outdoors.

For truly *robust* deployments of autonomous robot teams, a decentralized— or at least partly decentralized— solution to the MR-SLAM problem is required; one that allows teams of robots to operate independently for extended periods by sharing SLAM data and performing loop closures on-board. Decentralized SLAM, by design, also requires a *distributed* architecture; one where every robot stores, processes and shares its own local copy of MR-SLAM data.

8.2. Research Contributions

Chapter 4. Hybrid-Decentralized and Distributed Multi-Robot SLAM:

The research contributions I describe in this thesis enable teams of robots to operate autonomously for extended periods, independent of any centralized server. The proposed *hybrid-decentralized* architecture runs distributed instances of the same MR-SLAM back-end on each robot and server. These distributed instances allow teams of robots to continue MR-SLAM after they separate into smaller cliques, and include the ability to perform local loop closures. With the robots in each clique able to build their own copies of the global gridmap, global navigation and high-level autonomy can continue despite intermittent and lossy communications. Contributions in Chapter 4 include:

1. **Distributed MR-SLAM back-end:** allows computationally expensive MR-SLAM algorithms to be distributed over all robots and servers. The back-end design adapts to the available computational resources (Section 4.4.2).

2. **Distributed global map building:** allows each participant to build its own local copy of the global occupancy gridmap. This distributed spatial awareness enables decentralized global MRS operations (Section 4.3.1).

3. **Hybrid-decentralized pose graphs:** enables cliques of participants to operate independent of the GCS. Submap pinning and priority-based filters ensure that local pose graph copies converge to the global solution (Section 4.3.9).

4. **Immutable submapping:** minimizes overall system complexity, in particular, making the distributed and decentralized parts of the design both simple and robust. Immutable, or "read only," submaps, retain the ability to re-evaluate the most uncertain data associations (loop closures), while minimizing communications bandwidth (Section 4.3.3).

5. **Human-in-the-loop MR-SLAM:** increases mapping accuracy and convergence by allowing a human operator to interact with the pose graph and submaps in real-time. This is essential in real-world environments where Gaussian assumptions do not always hold (Section 4.4.3).

Chapter 5. Efficient Occupancy Gridmap Fusion and Matching:

Two algorithmic contributions in Chapter 5 are essential to enabling real-time MR-SLAM using the proposed submapping technique. Manipulating thousands of submaps is computationally expensive and these contributions show how the massively parallel capabilities of modern GPUs can be leveraged. The proposed approach is capable of creating large-scale global occupancy gridmaps with sufficient detail for indoor and outdoor navigation at large scales (over 500×500 meters). Contributions include:

6. **GPU-based occupancy gridmap fusion:** merges tens of thousands of submaps per second using a efficient highly-parallelized adaptation of the log odds binary Bayes filtering algorithm (Section 5.3).

7. **GPU-based multimodal constraint generation:** extracts descriptive, yet minimalistic, representations of the spatial relationships between submaps. A highly-parallelized gridmap correlation algorithm calculates likelihood volumes and extracts multimodal Gaussian distributions (Section 5.4).

Chapter 6. Robust Multimodal Pose Graph Optimization:

When perceptual aliasing ambiguities and cluttered environments produce submap alignments with complex multimodal Gaussian likelihoods, traditional unimodal techniques are forced to choose either a single mode (risking an outlier), or delay deciding until the hypothesis is confirmed (non-ideal for MR-SLAM). The COMBO technique enables complex likelihoods to be represented with multimodal constraints and avoids both delaying loop closures and creating outliers. Contributions in Chapter 6 include:

8. **Robust multimodal constraints:** enable loop closures to be preemptively added to the pose graph and outliers to rejected by consensus (Section 6.4.6). They also describe more complex spatial relationships between submaps, ideal for real-world environments with perceptual aliasing (Section 6.4.3).

9. **Multimodal pose graph optimization:** the continuous mode blending optimization (COMBO) technique enables complex multimodal Gaussian constraints to be optimized using traditional nonlinear least-squares approaches (Section 6.4.1).

8.3. Future Work

Chapter 4. Hybrid-Decentralized and Distributed Multi-Robot SLAM:

Robust ego-motion: The most significant issues in real-world deployments of the Mapbuilder MR-SLAM system can be traced back to odometry errors. More specifically, the moments when wheel slippages cause non-Gaussian odometry errors while the environment is either too sparse, or geometrically similar, for the errors to be corrected using lidar data alone. While the Mapbuilder GUI allows an operator to correct these errors, this interaction may create a bottleneck when scaling this MR-SLAM approach up to larger environments and larger team of robots.

If external localization such as RTK-GPS is not available, the best way to make ego-motion estimation more robust is with multi-modal sensing. Cameras are the most complementary sensors to wheel odometry and lidar, since their failure modes are somewhat orthogonal (Section 2.4.2). In the five years since the various robots were designed for the MAGIC challenge, it has become more practical to add *visual odometry* (VO) as an additional source of ego-motion estimates for wheeled robots [319, 220, 221]. It has been shown that fusing VO estimates with wheel odometry, IMU and lidar scan matching produces more robust ego-motion estimates [217, 125]. More recently, tightly-coupled visual inertial odometry (VIO) systems have been demonstrated that further increase accuracy and robustness [160].

Tightly-coupled VIO could dramatically improve the robustness and accuracy of the Local SLAM front-end, and would certainly avoid the majority of odometry problems experienced. A potential area of research is to integrate range-bearing measurements into a tightly-coupled filter along with visual feature tracking. While similar concepts have been demonstrated indoors using the Kinect [129] and outdoors using cameras and lidar [347], there are potential contributions that could be made here, particularly if extended to the multi-robot SLAM problem.

Loop closures: While the MR-SLAM approach described in this thesis creates thousands of *local* loop closures between nearby submaps, it was designed with the assumption that *global* loop closures would occasionally be assisted by a human operator. Visual place recognition (PR), described in Section 2.4.4.2 would provide an ideal source of potential global loop closures. Since I started this research work, PR implementations have become available that would help with map convergence [171, 285, 173, 172], however they are all designed for single robots. Multi-robot PR has been demonstrated by Forster et al. [348], however they relied on a centralized server to build a PR database and generate candidate loop closures. To add decentralized and distributed visual PR to my research work, the first steps would be to divide and distribute the PR database by augmenting the 2-D submaps. Lynen et al. have recently shown how to compress PR databases so they are smaller to share [349], however there remains considerable scope for future work developing decentralized and distributed PR.

Elevation mapping with 2.5-D submaps: Many rough or undulating terrains are not well represented by the flat 2-D submap representations used in this work; however these terrains are often traversable by mobile robots. While full 3-D maps, and SLAM in $\mathbb{SE}(3)$, are attractive solutions to accommodate these terrains, the computational, storage and communication costs may be prohibitive, particularly at the large scales demonstrated here (see Section 2.3.5). A less expensive approach is to augment the cells in each submap with height estimates, and then fuse them along with the occupancy gridmaps to form 2.5-D global elevation gridmaps. While elevation gridmaps are an old concept, I was unable to identify any approaches in the literature that builds them using submaps. Section 5.2 describes how implementing this would incur minimal additional cost. The GPU-based gridmap fusion approach described in Chapter 5 would be ideal to perform efficient blending of height data.

Multilevel submapping: With minimal effort, this approach could be adapted to handle overlapping geometry such as multi-story buildings and bridges. The concept of multilevel maps is not new [350, 149], however combining them with small 2.5-D submaps could be a novel research problem. Each submap's $\mathbb{SE}(2)$ pose would be augmented with a height estimate, and global gridmaps would be built by filtering submaps around a given height range.

Path planning directly in submaps: Beyond a certain size, global path planning in large gridmaps becomes too computationally expensive (discussed in Section 3.3.1). Furthermore, when multilevel submapping is used, overlapping geometry (e.g. a car parking structure) will produce inconsistent and unnavigable gridmaps. A graph of connected submaps provides a topological abstraction that is well-suited to large-scale global path planning. Planning in these hybrid metric/topological spaces has been demonstrated recently in [351] and [352], however only at small scales. There is considerable potential

research in this area, and this capability would be essential to scale the Mapbuilder to larger deployments.

Pose graph optimization algorithm: This work used Levenberg Marquardt with Cholesky factorization. Recent improvements to iSAM by Kaess et al. introduce a Bayes Tree structure that is more efficient than Cholesky factorization for most single robot problems [239]. It would be interesting to evaluate whether iSAM2 performs better than LM and Cholesky in MR-SLAM, particularly when distributed robots are producing frequent loop closures and structural changes across the entire pose graph (and hence Bayes tree).

Pose graph simplification: If long-term deployment (weeks or years) in a particular area is desired, the MR-SLAM system described here would need to be modified to bound computation and memory usage. It is likely that a simple heuristic could discard old and redundant submaps, and their corresponding constraints, however this should be considered in a principled manner similar to Strom and Olson's approach [56]. Several researchers have described how to simplify lidar-based pose graphs [274, 275, 353], and their approaches could be modified for the immutable submaps used here.

GUI usability: Section 7.4.5 reveals a large variation in interaction times required by the operators to manage the MRS and MR-SLAM systems. It hints that there may be an optimum user interface design that maximizes the operators' situational awareness and efficiency, while minimizing cognitive load. There is considerable scope here for future usability studies in user interface designs, particularly when multiple separate teams of robots are exploring in large deployments.

Open source: In [320] we describe an early attempt at porting Mapbuilder to ROS. At the time, however, ROS had no suitable replacements for the DDS middleware that provided decentralized peer-to-peer communications. More recently, it has been suggested that ROS 2 will use DDS for message passing [354]. The current plan is to rebuild Mapbuilder on top of ROS 2 when it becomes available, and then release it as an open source project.

Chapter 5. Efficient Occupancy Gridmap Fusion and Matching:

Gradient descent on likelihood volumes: It would be an interesting research problem to perform SGD (Section 2.5.4.9) directly on the 3-D likelihood volumes generated by the GPU-based correlative submap matching algorithm. Instead of extracting approximate multimodal Gaussian distributions, the GPU would build likelihood volumes for overlapping pairs of submaps. If overlaps changed significantly the likelihood volumes would be recalculated around the new relative pose. I have not seen this described in the literature, and it would be interesting to compare convergence properties with the direct methods used in this work.

GPU-based sensor models: Another interesting experiment would be to use the GPU to implement higher-fidelity lidar sensor models. Beam divergence effects that are typically too expensive to implement using a CPU (Section 2.3.3) could be implemented in real-time using the GPU. Similarly, after recovering the marginals for each submap's pose uncertainty [223], the GPU could be used to render submaps that are blurred according to their pose uncertainty. This blurring could represent occupancy gridmap likelihood distributions more accurately.

Mobile GPU implementation: The current Mapbuilder implementation uses the vendor-neutral OpenGL standard to interface GPUs. Using OpenGL ES 3.0 and minimal modifications, the Mapbuilder back-end could run on the GPUs available in modern *mobile phone* processors. An implementation like this could fast-track the Mapbuilder system for deployment on small and low-powered devices such as robotic vacuums.

Chapter 6. Robust Multimodal Pose Graph Optimization:

$\mathbb{SE}(3)$ **pose-graph SLAM:** The COMBO technique is likely to exhibit desirable convergence properties for multimodal Gaussian constraints in $\mathbb{SE}(3)$ pose graphs also. Perceptual aliasing effects also occur when aligning 3-D lidar point clouds [117], which suggests that COMBO could be useful for complex $\mathbb{SE}(3)$ likelihood distributions also. Evaluating and characterizing the 6-DOF multimodal $\mathbb{SE}(3)$ constraints would be an interesting research problem.

Visual SLAM: While most visual SLAM pipelines estimate both the 3-D position of visual features and the pose of a camera, some newer systems use a SWF to marginalize old feature observations to create a $\mathbb{SE}(3)$ pose graph with $\mathbb{SE}(3)$ constraints [221]. In some environments the unimodal $\mathbb{SE}(3)$ constraints that are typically created could be more accurately represented with multimodal constraints. Marginalizing features observations into multimodal $\mathbb{SE}(3)$ constraints would be an interesting research problem.

8.4. Final Thoughts

In the near future, the majority of mobile robot deployments are expected to be ground-based [355], and as their capabilities and prevalence continue to increase, the market's interest in combining multiple robots into cooperative teams is likely to increase also. These cooperative teams will be expected to have strongly-coordinated autonomous capabilities, and this autonomy will require teams to share localization and mapping information—capabilities provided by multi-robot SLAM systems.

Prior to this research work, there were no MR-SLAM systems described in the literature that enabled MRS deployments where teams of robots could maintain high-level

autonomy despite high-latency, lossy or intermittent communications with centralized servers— particularly at large scales and in unstructured real-world environments such as those encountered in the MAGIC challenge (i.e. 10 to 23 robots in a 500×500 meter indoor/outdoor environment).

To create this capability, the architecture I have proposed distributes the MR-SLAM back-end so that every robot is able to build its own copy of the global gridmap. This distributed approach is highly scalable, since each robot contributes the computational resources it requires to process its own sensor data and maintain its own registration to the global pose graph. The proposed hybrid-decentralized and distributed MR-SLAM architecture provides robust localization and mapping capabilities for large-scale deployments of autonomous teams, while not requiring reliable communications or external localization sensors such as GPS.

My research contributions have been demonstrated together in the Mapbuilder MR-SLAM system, which has been evaluated at large scales and in real-world conditions. While the Mapbuilder system is robust, there is considerable scope to increase its robustness and decrease (or eliminate) the need for operator intervention using one or more of the approaches described in the previous section. Of these approaches, the most effective is likely to be the addition of wide FOV cameras and tightly-coupled VIO— an area I am excited to be working in currently.

In the near future, MR-SLAM systems like Mapbuilder will enable teams of robots to cooperate in GPS-denied environments, with imperfect communications and without *a priori* maps. These capabilities are critical for military, disaster response, agricultural and mining applications where operations span large areas, network communications are unreliable and deployments must scale robustly. The potential utility of these systems, combined with economies of scale, could offer considerable benefits to humanity.

A

MRS Architecture and UGV Design

This appendix describes three MRS and UGV front-end designs (experimental setups) that were used to demonstrate my research contributions and to verify the performance of the Mapbuilder MR-SLAM system. The results in Chapter 7 combine datasets from the MAGIC challenge that were captured by these UGV setups: Section A.1 describes the WAMbot MRS architecture and UGV design. Section A.2 describes TM's UGV design, while Section A.3 describes Penn's UGV design. This appendix is provided for reference and includes contributions from WAMbot, TM and Penn's team members.

A.1. WAMbot MRS Architecture

MAGIC was a complex system-of-systems integration challenge. The WAMbot MRS architecture was necessarily complicated because it combined multiple distributed hardware and software systems and had to meet many challenging requirements. One requirement, for example, was for the centralized GCS to be able to control a team of UGVs up to 300 meters away *without* line of sight— this demanded a robust RF communications architecture. Similarly, requiring two operators to control five or more UGVs simultaneously demanded comprehensive coordination strategies and high-level autonomy.

The MRS requirements related to MR-SLAM are summarized in Section 4.1.3 on page 99. To encourage heterogeneity, the rules defined two classes of UGVs: disposable *sensor* UGVs for exploration and mapping, and expensive *disruptor* UGVs that carried sophisticated bomb defusing equipment.

High-level requirements and aspects of the MRS architecture specific to the high-level ISR mission (e.g. detecting OOIs), are described more thoroughly in [64]. Aspects of the MRS software, communications and hardware designs that are related to Mapbuilder and its hybrid-decentralized and distributed MR-SLAM architecture are described here.

A.1.1. Software Architecture

Our MRS software used a service-oriented architecture (SOA) approach; where the various MRS functionalities were split into a large number of separate software components (executables). Figure A.1 shows how the various software components are interrelated. Each disruptor UGV had 16 components and nine device interfaces, while each sensor UGV had 15 components and eight interfaces. At the GCS a single laptop computer ran the Mapbuilder back-end and GUI and four other components, while additional laptop computers could run additional instances of the GCS software. Chapter 4 describes the Local SLAM front-end and Mapbuilder back-end software components in more detail. The high-level MRS software architecture decisions are described in [68, 64].

The various software components communicated using a data distribution service (DDS) [321] middleware implemented by RTI. DDS was used to define structured message formats and provided libraries that allowed our team to write software components in both C++ and Java. The SOA approach with DDS enabled fast software development, since various software components and hardware interfaces could be shut down, replaced and restarted without having to restart the entire stack. For fast development and configurations, we used a model-driven architecture (MDA) approach. Metamodels were defined and code generation techniques used to create DDS bindings and other software stubs.

The Robot Operating System (ROS) is now a popular framework for UGV development because it brings a large number of libraries and functionalities together within a unified publisher-subscriber messaging system. In 2009, when the WAMbot architecture was designed, ROS lacked a robust decentralized messaging system like DDS. In recent work a small team and I ported the WAMbot MRS to ROS [320], however, the results were less robust. Once ROS version 2 is released it may be a good framework for MR-SLAM since it is being built using DDS for message passing [354].

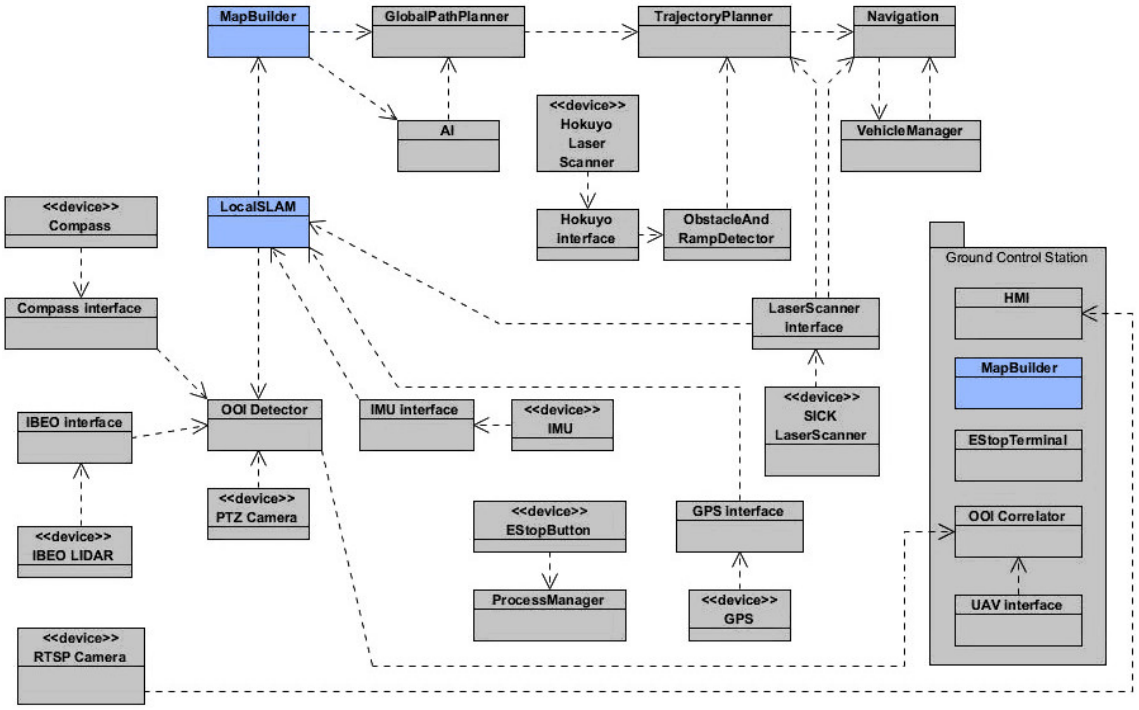


Figure A.1. WAMbot MRS component diagram: 16 software components and 9 device interfaces that ran on each UGV are shown. The components that ran on the GCS computer are inset on the right. The MR-SLAM components (Mapbuilder and Local SLAM) are highlighted in blue. UML diagram courtesy Adrian Boeing [64].

A.1.2. Communications Architecture

Data link and network layer: based on pre-challenge estimates, the maximum range required for wireless communications was about 300 meters, and without line of sight from the GCS wireless relays were necessary. To minimize cost our main wireless communications channel used COTS 802.11g Wi-Fi hardware reprogrammed to form a wireless mesh network. Ubiquity Pico Station 2HP Wi-Fi access points (APs) were flashed with a custom OpenWRT operating system, and configured to provide an OLSR mesh [356] over an ad-hoc Wi-Fi network. With APs mounted on each UGV and at the GCS, this multi-hop mesh network design provided more than the required 300 meter range. When UGVs were positioned as relays it enabled over-the-hill operations, albeit with a small fraction of the theoretical 54 Mbit/sec bandwidth available to 802.11g.

Transport layer: the DDS middleware implemented partitions that controlled which message types could pass over the mesh network to other UGVs and GCS participants. Local partitions restricted data to a single participant, in which case messages were transferred efficiently using local shared memory. Partitions grouped sensor UGVs, disruptor UGVs and the GCS, and enforced data separation rules required for disruptor UGVs. For messages passed between two participants, DDS was configured to use UDP packets. For the MR-SLAM related messages that were shared between all participants, efficient UDP

(a) Our ground control station (GCS) during a test scenario. A single COTS laptop computer provided sufficient processing power.

(b) Five WAMbot UGVs before deployment in the Old Ram Shed Challenge. We operated the GCS from this army tent.

Figure A.2. WAMbot MRS deployment at MAGIC challenge

broadcast packets were used. The DDS middleware handled all participant discovery, message delivery, data marshaling and demarshaling, flow control and quality-of-service (QoS) [321].

Application layer: DDS was used to provide a robust publisher-subscriber framework for real-time communications. Dozens of message data types were defined in the WAMbot MRS, and depending on UGV activities the data flow rates were highly variable. This variability, along with lossy data links, prompted us to make extensive use of DDS's fine-grained QoS model. The DDS QoS models allowed us to design configurable delivery policies for each message type and partition. In these policies, publishers were required to buffer up to n messages until their subscriber has confirmed receipt: *Guaranteed delivery*, where $n = \infty$, was used for critical messages such as command and control or OOI observations. *Best-effort delivery*, with $n \gg 1$, was used for certain MR-SLAM data, and included policies that allowed late-joining subscribers to automatically receive missed data samples. For certain fast-changing message types, policies were used to prevent buffering, i.e. $n = 0$, so that stale messages such as teleoperation commands were not erroneously executed late. Each message type had priorities and bandwidth allocations applied also. MR-SLAM specific message types and policies are described in Chapter 4.

A.1.3. Ground Control Station

Our GCS configuration was minimal and it could be deployed quickly. A single laptop computer running the human machine interface (HMI) software components, while connected to a mesh-enabled wireless AP, was sufficient to control the entire UGV team. To increase operator awareness and efficiency additional screens could be used. During the MAGIC challenge, the second operator had an additional laptop computer running the HMI; this configuration is shown in Figure A.2. Instances of Mapbuilder ran on each GCS

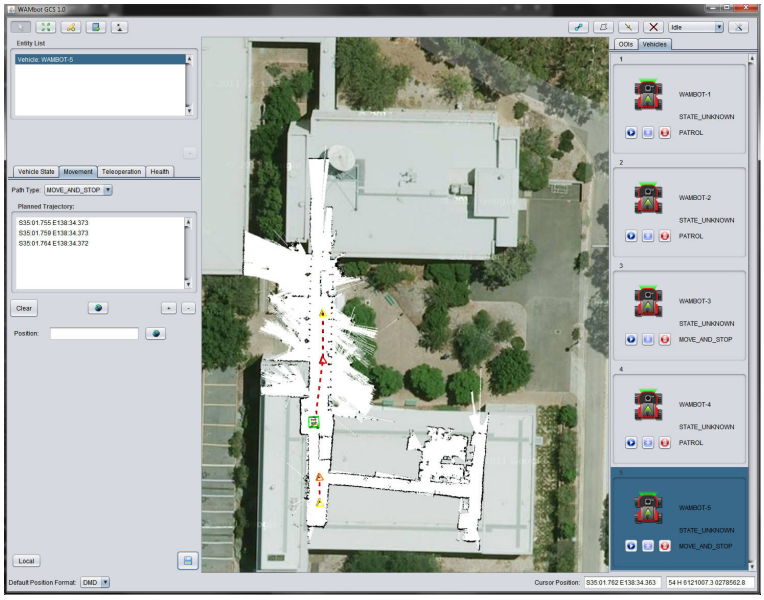


Figure A.3. WAMbot HMI: Screen shot taken during testing at Flinders University in Adelaide. The Mapbuilder occupancy gridmap is overlaid on an aerial image. Taken during debugging, an invalid constraint has caused the bottom part of the gridmap to be slightly misaligned.

computer alongside the HMI. Using DDS, the distributed design with network broadcasts allowed additional Mapbuilder instances to be run with minimal overhead.

Operator situational awareness was provided by the HMI, which displayed an aerial map of the environment with the WAMbot UGVs, OOI observations and exploration boundaries overlaid, similar to the screen shot in Figure A.3. A global occupancy gridmap, updated in real-time by the Mapbuilder back-end, was also overlaid on the aerial map. The HMI allowed operators to efficiently command and control the team of UGVs. While the Mapbuilder GUI provides a more fluid user interface, the HMI also allowed basic interactions with the pose graph, such as adding ground-truth constraints to anchor parts of the gridmap to orthorectified aerial imagery.

A.1.4. UGV Hardware Design

A short development timeline combined with our team's skills being biased towards software encouraged us to develop a UGV hardware design that integrated as many COTS components as possible. Various design trades are described in detail in [64], with cost and limited development time being the most significant factors. We assembled seven UGVs for our MRS, Figure A.2 shows five of them. Each UGV used a Pioneer AT3 base to provide a chassis, wheels, differential drive, motor controllers, batteries and wheel encoders. We fitted each UGV with an Intel Core 2 Duo automotive PC and a range of sensors, as shown in Figure A.4, that were interfaced with either USB 2.0, or 100 Mbit/s Ethernet. Each

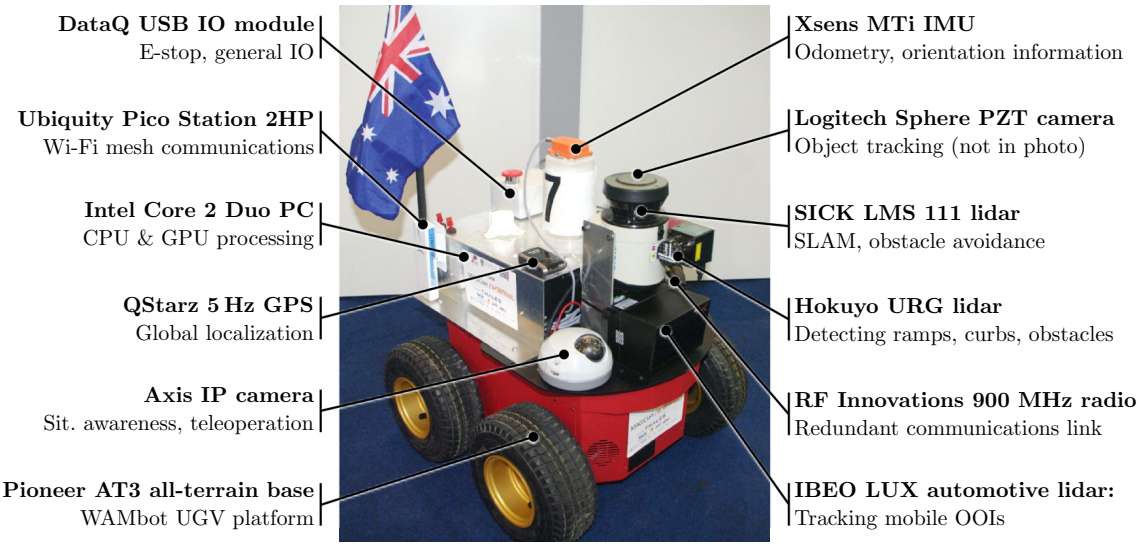


Figure A.4. WAMbot UGVs: The WAMbot hardware design integrated many COTS components.

UGV carried a homogeneous set of sensors that were used by the Local SLAM software component including:

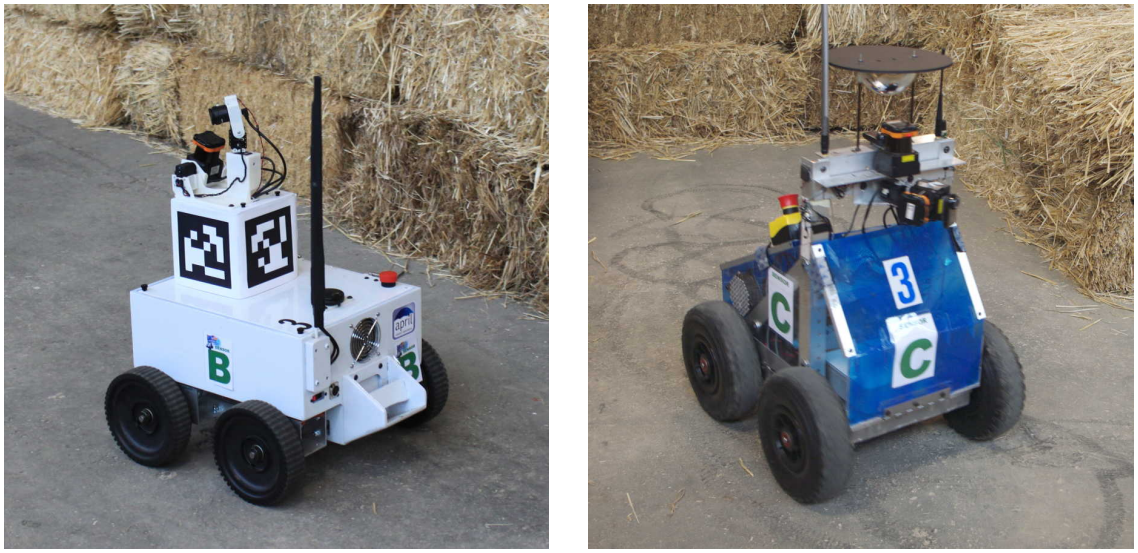
- SICK LMS 111 lidar: mounted horizontally (20 m, 270° range, 0.25° res, 25 Hz)
- Hokuyo URG-04LX lidar: fixed vertically (4 m, 240° range, 0.36° res, 10 Hz)
- Pioneer AT3 base: encoders for wheel odometry, $\hat{\mathbf{u}} = [\hat{x}_u, \hat{y}_u, \hat{\phi}_u]^T$
- Xsens MTi IMU: absolute pitch and roll estimation, relative yaw
- QStarz GPS: noisy global localization, 5 Hz

The Local SLAM front-end used these sensors to provide localization and mapping within submaps (see Section 4.4.1). The SICK lidar was the primary SLAM sensor; it scanned in a fixed horizontal plane 0.5 meters above the ground. Each scan returned 1080 measurements and provided detailed 2-D "slices" of the environment out to ~20 m. The disruptor UGVs included a long-range IBEO LUX multi-plane lidars for tracking OOIs, however this data was not integrated into the front-end due to the challenge's data separation rules.

A.2. Team Michigan UGV Front-End Design

Team Michigan's (TM's) challenge datasets were used in Chapter 7 to verify Mapbuilder MR-SLAM system. This section provides an overview of their MRS design and UGV hardware as it relates to these datasets, their system is described in more depth in [55, 53].

Each of their 15 custom designed differential-drive UGVs were controlled by a single Intel Core i5 laptop computer. Figure A.5 (a) shows one of their sensor UGVs. For communications they used 900 MHz radios that provide a theoretical 115.2 Kb/s bandwidth, and the Lightweight Communications and Marshalling (LCM) library [357]. Their SLAM-related sensors included:



(a) Team Michigan (TM) **(b) University of Pennsylvania (Penn)**

Figure A.5. Penn and TM's UGVs: Custom built for the MAGIC challenge. Datasets from these robots were used extensively in this research work. Photos taken in the maze section of the Old Ram Shed Challenge.

- Hokuyo UTM-30LX lidar: nodding (30 m, 270° range, 0.25° res, 40 Hz)
- Custom 6-DOF MEMS IMU: absolute pitch and roll estimation, relative yaw
- Drive wheel motor controllers: relative odometry estimates
- Garmin 18x GPS: noisy global localization, 5 Hz

TM's lidar was mounted horizontally, however it oscillated in a "nodding" configuration, actuated by a Dynamixel AX-12 servo. Every 1.25 seconds it swept the vertical field-of-view providing a 3-D point cloud with approximately 54,000 measurements. As a result of tight bandwidth restrictions TM's SLAM front-end projected these 3-D points into 2-D binary gridmap "maplets" using the pitch and roll information from the IMU. These 2-D gridmaps were losslessly compressed before being sent back to the GCS. Each UGV maintained a local gridmap used for localization and navigation. To help avoid odometry and scan-matching errors, a sliding window filter was used: lidar scans older than 15 seconds were discarded [53]. As shown in Figure A.5 (a), 2-D visual fiducials were attached to each UGV [166], which could be observed by servo-actuated cameras. This allowed UGVs to recognize each other and generate inter-UGV constraints; these constraints, however, were not used in this research.

TM's challenge datasets included the 2-D gridmaps compressed and stored in LCM structures. Olson and Strom provided two versions of their logs: lossy radio logs recorded at the GCS during the challenge, and also complete logs that were reconstructed after the challenge. To demonstrate robustness, only their lossy logs are used throughout this research.

A.3. University of Pennsylvania UGV Front-End Design

The University of Pennsylvania's (Penn's) challenge datasets were also used in Chapter 7 to verify Mapbuilder MR-SLAM system. This section provides an overview of their MRS design and UGV hardware as it relates to these datasets, their MRS is described thoroughly in [58, 60]. Each of their nine custom designed differential-drive UGVs were controlled by an Intel Core 2 Duo PC and a custom microcontroller board. Figure A.5 (b) shows one of their sensor UGVs. Similar to WAMbot, Penn used COTS Ubiquity 802.11g Wi-Fi routers. Instead of using middleware for message transport, they used raw UDP packets for best-effort delivery and TCP/IP for messages that required additional delivery attempts. SLAM sensors on each UGV included:

- Hokuyo UTM-30LX lidar: fixed horizontally (30 m, 270° range, 0.25° res, 40 Hz)
- Hokuyo UTM-30LX lidar: sweeping vertically (30 m, 270° range, 0.25° res, 40 Hz)
- Custom 6-DOF MEMS IMU: absolute pitch and roll estimation, relative yaw
- Hall effect encoders on motors: relative odometry estimates
- 50 Channel D2523T Helical GPS: noisy global localization, 5 Hz

Penn used one lidar mounted horizontally 0.8 meters above the ground for faster map updates and more reliable localization. A second lidar was mounted vertically and swept left and right to give ±60° horizontal FOV. Their sensor front-end used a Rao-Blackwellized particle filter to perform local SLAM, producing a gridmap with separate layers for each lidar. The first layer was a regular 2-D occupancy gridmap, while the second encoded height obstacles, such as curbs or stairs for traversability analysis. Deltas to the local 2-D gridmaps were transmitted back to the GCS periodically, each update relative to the UGV's local coordinate frame.

Penn also experienced communications losses, which meant that some of the gridmap deltas were not received by the GCS computers. Similar to TM, Penn's dataset included only the gridmaps received by the GCS computers. It is not known how much packet loss occurred, however their challenge datasets appear mostly complete.

B

MAGIC Challenge Datasets

This appendix describes the various phases and their datasets recorded at the MAGIC challenge. These are the test environments used to demonstrate my research contributions in Chapter 7 and to verify the performance of the Map-builder MR-SLAM system. Section B.1 provides an overview of the challenge environment, while Section B.2 provides an overview of each phase highlighting the mobility and perception issues that are "baked" into the challenge datasets. In Section B.3 I conclude with a discussion of the issues experienced with odometry, communications and GPS measurements.

B.1. Overview

The MAGIC challenge was introduced in Section 1.3.4.2 on page 12, while the WAMbot, TM and Penn UGVs are described in Appendix A. A high level overview of the challenge environment is given by Finn et al. in [52]. For reference, a pre-challenge map is shown in Figure B.1, while the post-challenge judges map is shown in Figure 7.1 on page 171. This post-challenge map provides a good overview of the challenge, including phase boundaries and OOI locations. Table B.1 summarizes the phases, including their extents and accessible areas. Figure 7.1 also shows ground-truth data acquired in a survey that was commissioned by the judges before the challenge. This ground-truth data was captured by a well-localized horizontal lidar and rendered into a gridmap. The survey covered about 80 % of both Phase 2 and the Old Ram Shed Challenge (ORSC), while the other phases are sparsely mapped.

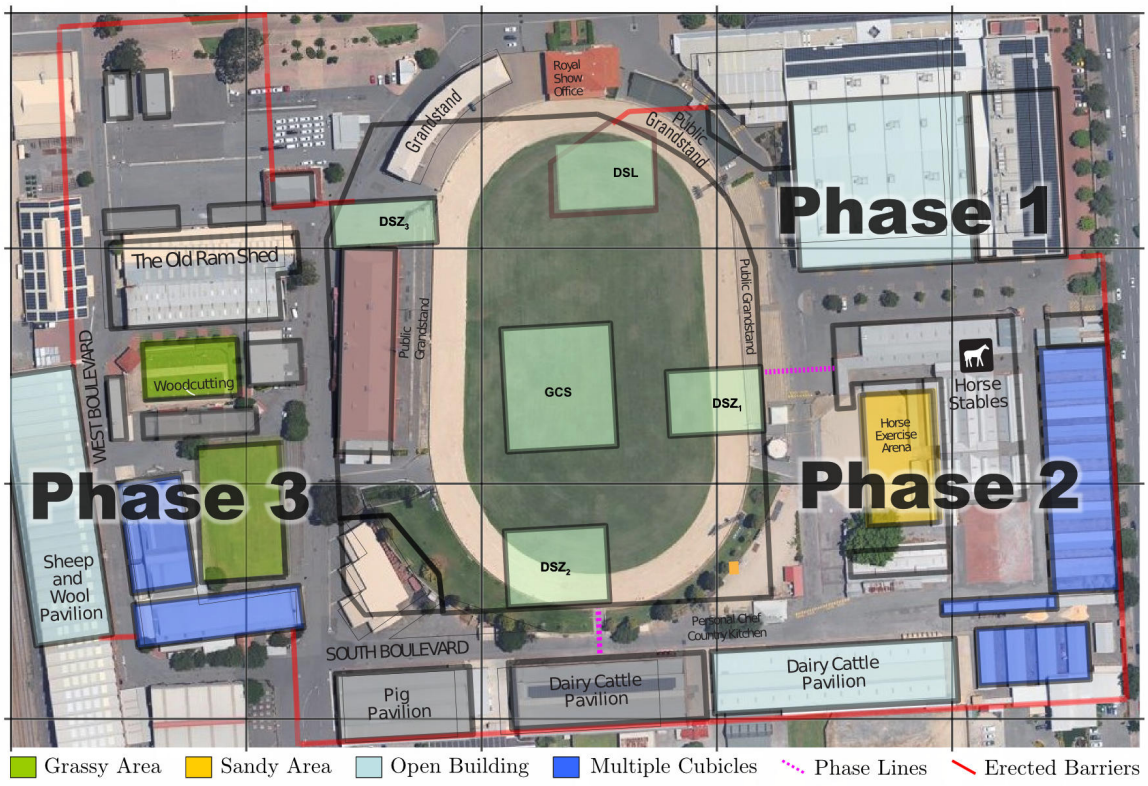


Figure B.1. MAGIC pre-challenge map: The challenge area at the Adelaide Agricultural Showgrounds in South Australia. This map was assembled based on hints and briefings in the weeks before the challenge. A 100 meter grid is overlaid. Aerial image courtesy DSTO.

The largest gridmaps produced by the WAMbot UGVs, overlaid on aerial imagery, are shown in Figure 7.2 on page 171. For reference, Figure B.2 shows TM's gridmaps recorded during their deployment at the challenge— these distorted maps are included to illustrate the various non-Gaussian odometry errors that are included in the datasets. Since the challenge, TM reprocessed their datasets and these odometry errors were removed by their operators [53, 54]. Penn's challenge-day maps were not released; their publications show maps produced using Mapbuilder [58, 358, 54].

	ORSC	Phase 1	Phase 2	Phase 3	Total	Specified
Phase extents (m)	80×40	210×140	210×150	230×300	450×300	500×500
Extents area (m^2)	3,200	29,400	31,500	69,000	135,000	250,000
Accessible area (m^2)	2,700	21,000	22,000	49,000	95,000	-
Percent of challenge	3 %	22 %	23 %	52 %	100 %	-

Static OOI count	5	6	4	3	18	>18
Mobile OOI count	0	0	2	7	9	

Table B.1. MAGIC challenge: Phase summaries

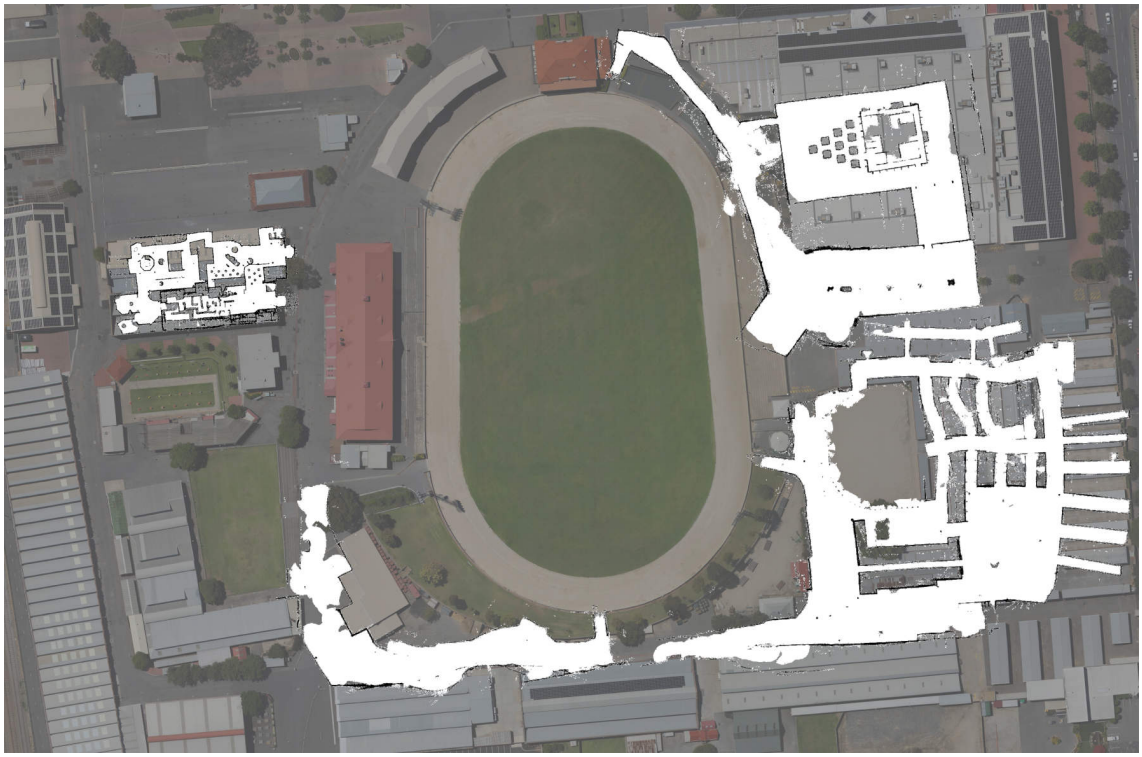


Figure B.2. Team Michigan challenge day results: Occupancy gridmaps superimposed on aerial imagery. TM won the challenge by exploring 100 % of the first two phases and 10 % of the third. TM released more accurate post-processed maps after the challenge. Gridmaps courtesy Ed Olson et al. [53, 54]. Aerial imagery courtesy DSTO.

B.2. Challenge Datasets

B.2.1. Old Ram Shed Challenge

The Old Ram Shed Challenge (ORSC) was a single-day competition held for press and dignitaries, after the main challenge had finished. The ISR mission was simplified to five UGVs searching for a handful of static OOIs distributed around an 80×40 meter indoor area. TM and Penn both explored 100 % of the area. Shown in Figure B.3, WAMbot explored 83 % of the ORSC area, with our UGVs performing better after addressing some of the hardware issues [64]. The ORSC datasets are considerably smaller than the main challenge phases, however combined they form a unique dataset due to the high average node degree and connectivity (Section 3.3.3). For reference, these ORSC datasets are about three times larger than the 45×25 meter indoor area used for the SDR program (Section 1.3.4).

B.2.2. Phase 1 Dataset

Referring to Figure 7.1 on page 171, each team started their UGVs at the point marked DSL and drove into a curving alley that is 5 meters wide. Two doorways entered into the

large Wayville Pavilion; one near the static OOI marked AA and another on the south east wall. Once inside the pavilion another four OOIs were hidden in a maze-like environment containing large wooden ramps on 15° angles. Figure B.2 shows TM's gridmaps after spending 80 minutes in this phase. The UGVs were "over-the-hill" behind a large grandstand for most of this phase, and required a UGV to function as a communications relay.

SLAM-related problems occurred in the first alleyway, where minor perceptual aliasing errors were observed. To the south of the pavilion a sparse 75×30 m outdoor area caused many problems. Wheels slipped on the hard asphalt surface causing odometry errors, while objects in this mostly-empty space were further away than the Hokuyo lidar scanner's range. In Figure B.2, significant distortions can be seen in the south west corner of Phase 1 where lidar scan matching was unable to correct odometry errors.

B.2.3. Phase 2 Dataset

The topology of Phase 2 was more complex, including two large loop closures approximately 200 meters in length, and many smaller loop closures. The UGVs started at the point marked DSZ1 in Figure 7.1 on page 171. After entering through an alley teams explored north and south along East Crescent without attempting to enter the large sandpit. An oblique aerial view is shown in Figure 1.8 on page 15 along with a panorama taken at the start of Phase 2. Visible in this photo is the southern route; a 20 meter wide alley with a v-shaped drain down the middle. Similar to the first phase, non-Gaussian odometry errors occurred that neither TM or Penn's SLAM front-ends could correct using lidar data alone.

South Boulevard, shown at the south edge of Phase 2 in Figure 7.1 on page 171, was another 20 meter wide alley that caused significant problems. Figure B.4 (a) shows an area on the boulevard where a parked car forced the UGVs to drive through a "v-shaped" drain. At many places along the boulevard this drain caused significant odometry errors for both TM and Penn's SLAM front-ends. These errors persisted in both teams' datasets,

Figure B.3. Old Ram Shed Challenge: Our WAMbot UGVs robots exploring the indoor 80×40 meter area. **Left:** First area inside the ORSC. **Center, Right:** A maze of hay bales.

(a) South Boulevard: A car was parked near this v-shaped drain, causing significant odometry errors.

(b) Horse stalls: The damaged asphalt surface caused odometry errors and in some places these relatively featureless alleys to be stretched. Heading estimates were generally maintained due to the close walls.

Figure B.4. Phase 2: Photos taken several days after the challenge.

Figure B.5. Phase 2: Panorama of Red Square, a large 45×45 m area with a mobile OOI.

with Figure B.2 showing TM's gridmaps after 80 minutes of exploration. Some damaged submaps from this location are shown in Figure B.6. The effects of these errors, before being corrected by an operator, are shown in a screen shot of the GUI in Figure 4.9 on page 130.

As the teams progressed their UGVs explored the stables, marked A, B1 and B2 in Figure 7.1 on page 171. Damaged asphalt surfaces, like the ones shown in Figure B.4 (b), caused repeated mobility and odometry issues. Parallel walls caused many perceptual aliasing problems; in a few places the walls were stretched due to uncorrected wheel slippage, however heading estimates were generally maintained. Figure B.4 (b) also shows the large, irregular, building eaves that added confusion while matching submap gridmaps to the aerial imagery.

Further issues in Phase 2 occurred in a large 45×45 m area labeled "Red Square" in

Figure 7.1 on page 171. A panorama of this large open space is shown in Figure B.5. The area was patrolled by a mobile OOI and two of Penn's UGVs drove into the middle of this area where their 30 meter Hokuyo lidars faltered and odometry errors rapidly accumulated. Figure 7.26 on page 209 shows two submaps that were corrupted by the mobile OOIs. Both teams experienced communications issues in the south-east corner of this phase.

B.2.4. Phase 3 Dataset

The third phase had twice the accessible area, and potentially two loop closures over 250 meters long. The ISR mission complexity increased significantly, with a scenario where a hostile OOI walked alongside a non-combatant. No team was able to neutralize the hostile OOI to gain access to the rest of the phase. For reference, WAMbot mapped 5 % and TM mapped 10 % of this phase.

B.3. Post-Challenge Dataset Notes

Wheel slippage: the most frequent issue in the challenge datasets are large irregular wheel-slippages where the SLAM front-ends could not disambiguate the UGV's motion using lidar scan matching. These non-Gaussian odometry errors lead to bad constraints being added to the pose graph, and many submaps being corrupted. A robust pose graph optimizer may cope with erroneous constraints (see Section 6.3), however corrupted

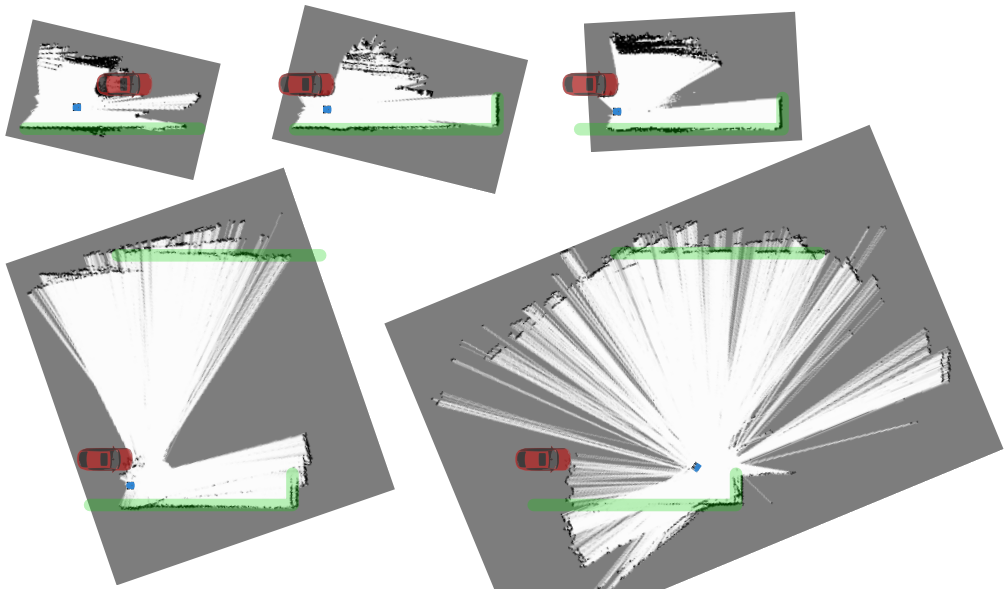
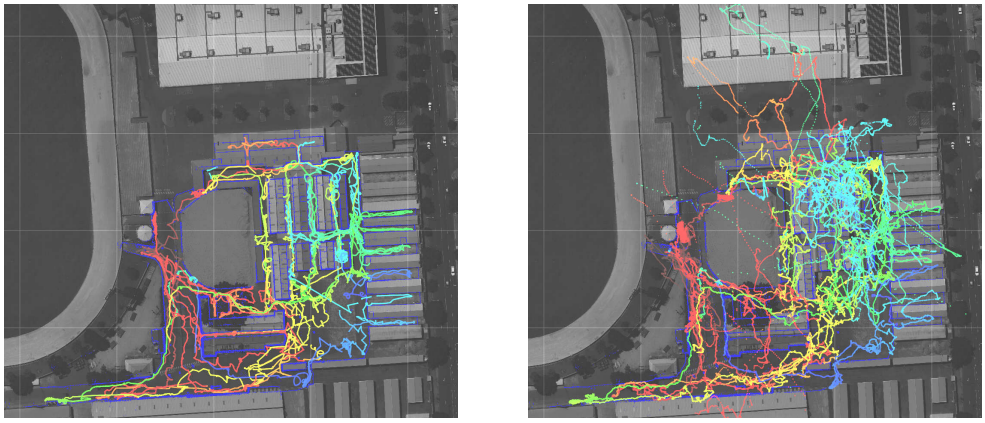


Figure B.6. Phase 2 South Boulevard: Corrupted submaps from a 20 meter wide alley where wheel slippage consistently occurred in a drain. Both TM and Penn's SLAM front-ends produced unusable submaps. **Blue:** UGV. **Red:** Graphic where car was parked. **Green:** Nearby walls.



(a) MR-SLAM trajectories **(b)** GPS sensor data

Figure B.7. Phase 2: TM's reprocessed MR-SLAM map from [328] showing GPS data. Trajectories for 12 UGVs are shown in each figure, with the hue mapped from red at the start to blue after 2 hours. A 50 meter grid is overlaid. In (b) GPS position data can be seen drifting widely, with up to 80 meters error around the horse stables due to multipath effects. Maps courtesy Ryan Morton.

submaps are very hard to detect and ignore if the SLAM front-end continues to express confidence in them. This is addressed by the Local SLAM front-end in Section 4.3.6. The challenge datasets do not contain raw sensor data, and the reduced 2-D gridmap representations frequently have these errors "baked" in. Figure B.6 shows some corrupt submaps from the South Boulevard in the Phase 2 dataset.

Data loss: as described in Appendix A, both TM and Penn's UGVs had communications issues that result in different types of data loss. Both SLAM front-ends used decoupled local coordinate systems that in many cases allows MR-SLAM to continue (see Section 3.2.2). In some places, however, odometry errors had accumulated during communications blackouts, and once communication had been recovered the UGVs jumped tens of meters and became badly localized. A vigilant operator is required to correct these non-Gaussian errors before they corrupt the global map.

GPS errors: while each challenge phase began with a clear view of the sky (ideal conditions for GPS) the UGVs quickly encountered degenerate satellite configurations driving through agricultural sheds. In these situations, buildings with opaque metal roofs blocked the majority of satellites, while intermittent position fixes were obtained from satellites closer to the horizon. Multipath signal reflections became frequent in these predominantly horizontal configurations, causing large step errors in the GPS position measurements. For reference, in Figure B.7 Morton and Olson show up to 80 meters of position error in Phase 2 when comparing their GPS position measurements with post-processed trajectories [328]. The results presented in Chapter 7 ignored the GPS measurements due to a lack of an appropriate uncertainty model. TM also reported not using their GPS measurements during the challenge [53].

Bibliography

[1] L. Nocks, *The Robot: The Life Story of a Technology*. Greenwood Publishing Group, 2007. 1

[2] R. Brooks, *Flesh and Machines: How Robots Will Change Us*. Pantheon Books, ISBN: 978-375420795, 2002. 2

[3] R. Kurzweil, *The Singularity Is Near: When Humans Transcend Biology*. Viking Penguin, ISBN: 0-670-03384-7, 2005. 2

[4] L. Riazuelo, J. Civera, and J. M. M. Montiel, "C2tam: A Cloud framework for cooperative tracking and mapping," *Robotics and Autonomous Systems*, 2013. 3, 16, 92

[5] G. Mohanarajah, V. Usenko, M. Singh, M. Waibel, and R. D'Andrea, "Cloud-based collaborative 3d mapping in real-time with low-cost robots," *IEEE Transactions on Automation Science and Engineering*, 2014. 3, 16, 92

[6] M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 229–241, 2001. 5, 58, 81

[7] F. Dellaert and M. Kaess, "Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing," *The International Journal of Robotics Research*, vol. 25, pp. 1181–1203, Dec. 2006. 5, 24, 55, 67, 77, 90

[8] A. Birk, *Simultaneous Localization and Mapping (SLAM) is NP-Complete*. 2010. 5, 24, 55

[9] R. C. Smith and P. Cheeseman, "On the Representation and Estimation of Spatial Uncertainty," *The International Journal of Robotics Research*, vol. 5, pp. 56–68, Dec. 1986. 5, 57, 81

[10] H. Durrant-Whyte, "Uncertain geometry in robotics," *IEEE Journal of Robotics and Automation*, vol. 4, pp. 23 –31, Feb. 1988. 5

[11] J. Leonard and H. Durrant-Whyte, "Simultaneous Map Building and Localization for an Autonomous Mobile Robot," in *IEEE/RSJ International Workshop on Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems, Proceedings IROS '91*, pp. 1442 –1447 vol.3, Nov. 1991. 5, 32

[12] S. Thrun and J. J. Leonard, "Simultaneous Localization and Mapping," in *Springer Handbook of Robotics* (O. Khatib, ed.), pp. 871–889, Springer Berlin Heidelberg, Jan. 2008. 5, 8

[13] K. Konolige, J. Augenbraun, N. Donaldson, C. Fiebig, and P. Shah, "A low-cost laser distance sensor," in *IEEE International Conference on Robotics and Automation, 2008. ICRA 2008*, pp. 3002 –3008, May 2008. 5

[14] T. Arai, E. Pagello, and L. E. Parker, "Editorial: Advances in multi-robot systems," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 655–661, 2002. 6, 9

[15] L. E. Parker, "Multiple Mobile Robot Systems," in *Springer Handbook of Robotics*, Springer, June 2008. 6

[16] T. Yasuda and K. Ohkura, *Multi-Robot Systems, Trends and Development*. InTech, 2011. 6

[17] Y. Mohan and S. G. Ponnambalam, "An extensive review of research in swarm robotics," in *World Congress on Nature Biologically Inspired Computing, 2009. NaBIC 2009*, pp. 140–145, Dec. 2009. 6

[18] A. Farinelli, L. Iocchi, and D. Nardi, "Multirobot systems: a classification focused on coordination," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 34, pp. 2015 –2028, Oct. 2004. 6, 7

[19] B. P. Gerkey and M. J. Matarić, "A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems," *The International Journal of Robotics Research*, vol. 23, pp. 939–954, Sept. 2004. 6, 9

[20] C. L. Ortiz, R. Vincent, and B. Morisset, "Task inference and distributed task management in the Centibots robotic system," in *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, AAMAS '05, (New York, NY, USA), pp. 860–867, ACM, 2005. 6, 12

[21] K. Lerman, C. Jones, A. Galstyan, and M. J. Matarić, "Analysis of Dynamic Task Allocation in Multi-Robot Systems," *The International Journal of Robotics Research*, vol. 25, pp. 225–241, Mar. 2006. 6

[22] E. Guizzo, "Three engineers, hundreds of robots, one warehouse," *Spectrum, IEEE*, vol. 45, no. 7, pp. 26–34, 2008. 9, 10

[23] A. Morris, D. Ferguson, Z. Omohundro, D. Bradley, D. Silver, C. Baker, S. Thayer, C. Whittaker, and W. Whittaker, "Recent developments in subterranean robotics," *Journal of Field Robotics*, vol. 23, pp. 35–57, Jan. 2006. 9, 18

[24] J. Larsson, J. Appelgren, J. A. Marshall, and T. D. Barfoot, "Atlas Copco Infrastructureless Guidance System for High-Speed Autonomous Underground Tramming," in *Proceedings of the 5th Int. Conference and Exhibition on Mass Mining, MassMin*, pp. 585–594, 2008. 9, 10, 18

[25] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun, "Collaborative multi-robot exploration," in *IEEE International Conference on Robotics and Automation, 2000. Proceedings. ICRA '00*, vol. 1, pp. 476 –481 vol.1, 2000. 9

[26] R. Madhavan, K. Fregene, and L. E. Parker, "Distributed Cooperative Outdoor Multirobot Localization and Mapping," *Autonomous Robots*, vol. 17, pp. 23–39, July 2004. 9, 61

[27] D. Fox, J. Ko, K. Konolige, B. Limketkai, D. Schulz, and B. Stewart, "Distributed Multirobot Exploration and Mapping," *Proceedings of the IEEE*, vol. 94, pp. 1325 – 1339, 2006. 9, 12, 65, 81, 84, 88, 89, 93, 95

[28] K. Konolige, D. Fox, C. Ortiz, A. Agno, M. Eriksen, B. Limketkai, J. Ko, B. Morisset, D. Schulz, B. Stewart, and R. Vincent, "Centibots: Very Large Scale Distributed Robotic Teams," in *Experimental Robotics IX* (M. H. Ang and O. Khatib, eds.), vol. 21, pp. 131–140, Berlin/Heidelberg: Springer-Verlag, 2006. 9, 12, 89, 93, 95, 96, 108

[29] A. Howard, L. E. Parker, and G. S. Sukhatme, "Experiments with a Large Heterogeneous Mobile Robot Team: Exploration, Mapping, Deployment and Detection," *The International Journal of Robotics Research*, vol. 25, pp. 431 –447, May 2006. 9, 12, 86, 89, 93, 95, 105

[30] R. Vincent, D. Fox, J. Ko, K. Konolige, B. Limketkai, B. Morisset, C. Ortiz, D. Schulz, and B. Stewart, "Distributed multirobot exploration, mapping, and task allocation," *Annals of Mathematics and Artificial Intelligence*, vol. 52, no. 2, pp. 229–255, 2008. 9

[31] B. Kim, M. Kaess, L. Fletcher, J. Leonard, A. Bachrach, N. Roy, and S. Teller, "Multiple Relative Pose Graphs for Robust Cooperative Mapping," in *IEEE Intl. Conf. on Robotics and Automation, ICRA*, (Anchorage, Alaska), pp. 3185–3192, May 2010. 9, 81, 88, 92, 94, 95, 108

[32] T. A. Vidal-Calleja, C. Berger, J. Solà, and S. Lacroix, "Large scale multiple robot visual mapping with heterogeneous landmarks in semi-structured terrain," *Robotics and Autonomous Systems*, vol. 59, no. 9, pp. 654–674, 2011. 9

[33] M. Dorigo, D. Floreano, L. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, *et al.*, "Swarmanoid: a novel concept for the study of heterogeneous robotic swarms," *IEEE Robotics & Automation Magazine*, 2012. 9, 10

[34] A. Gautam and S. Mohan, "A review of research in multi-robot systems," in *2012 7th IEEE International Conference on Industrial and Information Systems (ICIIS)*, pp. 1 –5, Aug. 2012. 9

[35] O. S. R. Foundation, "TurtleBot 2," 2014. 10

[36] Q. J. Lindsey, D. Mellinger, and V. Kumar, "Construction of Cubic Structures with Quadrotor Teams," *Robotics: Science and Systems*, June 2011. 10

[37] A. Kushleyev, D. Mellinger, and V. Kumar, "Towards a Swarm of Agile Micro Quadrotors," *Robotics: Science and Systems*, July 2012. 10

[38] S. Weiss, *Vision based navigation for micro helicopters.* PhD thesis, ETH, Zurich, 2012. 10

[39] M. Achtelik, S. Weiss, M. Chli, F. Dellaert, and R. Siegwart, "Collaborative stereo," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 2242–2248, 2011. 10

[40] D. Scaramuzza, M. Achtelik, L. Doitsidis, F. Fraundorfer, E. Kosmatopoulos, A. Martinelli, M. Achtelik, M. Chli, S. Chatzichristofis, L. Kneip, *et al.*, "Vision-controlled micro flying robots: from system design to autonomous navigation and mapping in GPS-denied environments," *IEEE Robotics & Automation Magazine*, pp. 1–10, 2013. 10

[41] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems," in *In Proceedings of the 11th International Conference on Advanced Robotics*, pp. 317–323, 2003. 11

[42] R. Vaughan, "Massively Multi-Robot Simulation in Stage," *Swarm Intelligence*, vol. 2, pp. 189–208, Dec. 2008. 11

[43] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3, pp. 2149–2154, IEEE, 2004. 11

[44] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "USARSim: a robot simulator for research and education," in *Robotics and Automation, 2007 IEEE International Conference on*, pp. 1400–1405, IEEE, 2007. 11

[45] H. Kitano and S. Tadokoro, "RoboCup Rescue: A Grand Challenge for Multiagent and Intelligent Systems," *AI Magazine*, vol. 22, p. 39, Mar. 2001. 11

[46] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, *et al.*, "ARGoS: a modular, multi-engine simulator for heterogeneous swarm robotics," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 5027–5034, 2011. 11

[47] T. Bräunl, *Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems*. 3rd ed., 2008. 11, 43, 81

[48] J. A. Rothermich, M. İ. Ecemiş, and P. Gaudiano, "Distributed Localization and Mapping with a Robotic Swarm," in *Swarm Robotics*, no. 3342 in Lecture Notes in Computer Science, pp. 58–69, Springer Berlin Heidelberg, Jan. 2005. 12

[49] J. Ko, B. Stewart, D. Fox, K. Konolige, and B. Limketkai, "A practical, decision-theoretic approach to multi-robot mapping and exploration," in *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings*, vol. 4, pp. 3232 – 3238 vol.3, Oct. 2003. 12, 89

[50] A. Howard, "Multi-robot Simultaneous Localization and Mapping using Particle Filters," *The International Journal of Robotics Research*, vol. 25, pp. 1243 –1256, Dec. 2006. 12, 65, 81, 95

[51] A. Howard, L. E. Parker, and G. S. Sukhatme, "The SDR Experience: Experiments with a Large-Scale Heterogeneous Mobile Robot Team," in *Experimental Robotics IX*, no. 21 in Springer Tracts in Advanced Robotics, pp. 121–130, Springer Berlin Heidelberg, Jan. 2006. 12, 14, 89, 93, 94, 105, 108

[52] A. Finn, A. Jacoff, M. Del Rose, B. Kania, J. Overholt, U. Silva, and J. Bornstein, "Evaluating autonomous ground-robots," *Journal of Field Robotics*, vol. 29, no. 5, pp. 689–706, 2012. 12, 13, 169, 171, 210, 233

[53] E. Olson, J. Strom, R. Morton, A. Richardson, P. Ranganathan, R. Goeddel, M. Bulic, J. Crossman, and B. Marinier, "Progress toward multi-robot reconnaissance and the MAGIC 2010 competition," *Journal of Field Robotics*, vol. 29, no. 5, pp. 762–792, 2012. 13, 81, 83, 84, 86, 88, 90, 91, 92, 93, 94, 95, 96, 130, 153, 210, 211, 212, 213, 230, 231, 234, 235, 239

[54] E. Olson, J. Strom, R. Goeddel, R. Morton, P. Ranganathan, and A. Richardson, "Exploration and Mapping with Autonomous Robot Teams," *Communications of the ACM*, Dec. 2012. 13, 90, 91, 92, 93, 95, 213, 234, 235

[55] P. Ranganathan, R. Morton, A. Richardson, J. Strom, R. Goeddel, M. Bulic, and E. Olson, "Coordinating a Team of Robots for Urban Reconnaisance," in *Land Warfare Conference*, 2010. 13, 95, 213, 230

[56] J. Strom and E. Olson, "Occupancy grid rasterization in large environments for teams of robots," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 4271–4276, 2011. 13, 81, 91, 95, 139, 213, 214, 222

[57] E. Olson and P. Agarwal, "Inference on networks of mixtures for robust robot mapping," *International Journal of Robotics Research*, Feb. 2013. 13, 81, 95, 125, 153, 160, 161, 200, 214, 215, 216

[58] J. Butzke, K. Daniilidis, A. Kushleyev, D. D. Lee, M. Likhachev, C. Phillips, and M. Phillips, "The University of Pennsylvania MAGIC 2010 multi-robot unmanned vehicle system," *Journal of Field Robotics*, vol. 29, no. 5, pp. 745–761, 2012. 13, 81, 84, 86, 88, 90, 91, 92, 93, 232, 234

[59] J. Butzke and M. Likhachev, "Planning for multi-robot exploration with multiple objective utility functions," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3254 –3259, Sept. 2011. 13

[60] J. Butzke, K. Daniilidis, V. Kumar, A. Kushleyev, D. D. Lee, M. Likhachev, C. Phillips, and M. Phillips, "University of Pennsylvania MAGIC 2010 Final Report," tech. rep., Jan. 2011. 13, 93, 232

[61] A. Lacaze, K. Murphy, M. Del Giorno, and K. Corley, "Reconnaissance and Autonomy for Small Robots (RASR) team: MAGIC 2010 challenge," *Journal of Field Robotics*, vol. 29, no. 5, pp. 729–744, 2012. 13, 81

[62] A. Lacaze, K. Murphy, M. Giorno, and K. Corley, "Reconnaissance and Autonomy for Small Robots (RASR): MAGIC 2010 Challenge," in *TARDEC Quarterly Robotics Workshop. Michigan*, 2011. 13

[63] A. Lacaze, K. Murphy, and M. Del Giorno, "MAGIC 2010 RASR Team," tech. rep., Dec. 2010. 13

[64] A. Boeing, M. Boulton, T. Bräunl, B. Frisch, S. Lopes, A. Morgan, F. Ophelders, S. Pangeni, R. Reid, K. Vinsen, N. Garel, C. S. Lee, M. Masek, A. Attwood, M. Fazio,

and A. Gandossi, "WAMbot: Team MAGICian's entry to the Multi Autonomous Ground-robotic International Challenge 2010," *Journal of Field Robotics*, vol. 29, no. 5, pp. 707–728, 2012. 13, 81, 88, 90, 92, 93, 95, 170, 171, 211, 226, 227, 229, 235

[65] A. Boeing, S. Pangeni, T. Bräunl, and C. S. Lee, "Real-time tactical motion planning and obstacle avoidance for multi-robot cooperative reconnaissance," in *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 3117 –3122, Oct. 2012. 13

[66] R. Reid and T. Bräunl, "Large-scale multi-robot mapping in MAGIC 2010," in *2011 IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, pp. 239–244, IEEE, Sept. 2011. 13, 81, 88, 90, 92, 93, 95, 146, 150, 190, 208

[67] A. Boeing, T. Bräunl, R. Reid, A. Morgan, and K. Vinsen, "Cooperative Multi-Robot Navigation and Mapping of Unknown Terrain," in *2011 IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, pp. 234–238, IEEE, Sept. 2011. 13, 95

[68] A. Boeing, M. Boulton, T. Bräunl, B. Frisch, S. Lopes, A. Morgan, F. Ophelders, S. Pangeni, R. Reid, K. Vinsen, N. Garel, C. S. Lee, M. Masek, A. Attwood, M. Fazio, and A. Gandossi, "Team MAGICian," in *Land Warfare Conference*, 2010. 13, 213, 226

[69] A. Erdener, E. O. Ari, Y. Ataseven, B. Deniz, K. G. Ince, U. Kazancioğlu, T. A. Kopanoğlu, T. Koray, K. M. Koşaner, A. Özgür, Ç. Ç. Özkök, T. Soncul, H. O. Irin, I. Yakin, S. Biddlestone, L. Fu, A. Kurt, Ü. Özgüner, K. Redmill, Ö. Aytekin, and I. Ulusoy, "Team Cappadocia Design for MAGIC 2010 (The ASELSAN Team)," in *Land Warfare Conference 2010*, Nov. 2010. 13

[70] J. C. Mankins, "Technology Readiness Levels," *White Paper, April*, vol. 6, 1995. 14

[71] N. Michael, S. Shen, K. Mohta, Y. Mulgaonkar, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida, K. Ohno, E. Takeuchi, and S. Tadokoro, "Collaborative mapping of an earthquake-damaged building via ground and aerial robots," *Journal of Field Robotics*, vol. 29, no. 5, pp. 832–841, 2012. 16

[72] R. R. Murphy, K. L. Dreger, S. Newsome, J. Rodocker, B. Slaughter, R. Smith, E. Steimle, T. Kimura, K. Makabe, K. Kon, H. Mizumoto, M. Hatayama, F. Matsuno, S. Tadokoro, and O. Kawase, "Marine heterogeneous multirobot systems at the great Eastern Japan Tsunami recovery," *Journal of Field Robotics*, vol. 29, no. 5, pp. 819–831, 2012. 16

[73] J. Dietsch, "DARPA Entices Roboticists to Take the Next Step," *IEEE Robotics Automation Magazine*, vol. 19, pp. 9 –10, Sept. 2012. 16

[74] Y. Liu and G. Nejat, "Robotic Urban Search and Rescue: A Survey from the Control Perspective," *Journal of Intelligent & Robotic Systems*, Mar. 2013. 16

[75] P. Velagapudi, P. Scerri, K. Sycara, H. Wang, M. Lewis, and J. Wang, "Scaling effects in multi-robot control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008. IROS 2008*, pp. 2121 –2126, Sept. 2008. 16

[76] G.-J. M. Kruijff, F. Colas, T. Svoboda, J. van Diggelen, P. Balmer, F. Pirri, and R. Worst, "Designing Intelligent Robots for Human-Robot Teaming in Urban Search and Rescue," in *AAAI Spring Symposium: Designing Intelligent Robots*, 2012. 16

[77] Van Riper, "A Concept for Future Military Operations on Urbanized Terrain.," tech. rep., United States Marine Corps, Defense Technical Information Center, 1999. 17

[78] N. O. Tippenhauer, C. Pöpper, K. B. Rasmussen, and S. Capkun, "On the requirements for successful GPS spoofing attacks," in *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, (New York, NY, USA), pp. 75–86, ACM, 2011. 17, 51

[79] A. J. Kerns, D. P. Shepard, J. A. Bhatti, and T. E. Humphreys, "Unmanned Aircraft Capture and Control Via GPS Spoofing: Unmanned Aircraft Capture and Control," *Journal of Field Robotics*, vol. 31, pp. 617–636, July 2014. 17, 51, 81

[80] J. Billingsley and M. Schoenfisch, "The successful development of a vision guidance system for agriculture," *Computers and Electronics in Agriculture*, vol. 16, pp. 147–163, Jan. 1997. 17

[81] A. Stentz, C. Dima, C. Wellington, H. Herman, and D. Stager, "A system for semi-autonomous tractor operations," *Autonomous Robots*, vol. 13, no. 1, pp. 87–104, 2002. 17

[82] D. Oetomo, J. Billingsley, and J. F. Reid, "Editorial: Agricultural robotics," *Journal of Field Robotics*, vol. 26, pp. 501–503, June 2009. 18

[83] M. Bryson, A. Reid, F. Ramos, and S. Sukkarieh, "Airborne Vision-Based Mapping and Classification of Large Farmland Environments," *Journal of Field Robotics*, 2010. 18

[84] A. Barrientos, J. Colorado, J. d. Cerro, A. Martinez, C. Rossi, D. Sanz, and J. Valente, "Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots," *Journal of Field Robotics*, vol. 28, pp. 667–689, Sept. 2011. 18

[85] M. Bryson and S. Sukkarieh, "Architectures for Cooperative Airborne Simultaneous Localisation and Mapping," *J. Intell. Robotics Syst.*, vol. 55, no. 4-5, pp. 267–297, 2009. 18, 81, 89, 92, 93

[86] R. González, F. Rodríguez, J. Sánchez-Hermosilla, J. Donaire, and others, "Navigation Techniques for Mobile Robots in Greenhouses," *Applied Engineering in Agriculture*, vol. 25, no. 2, p. 153, 2009. 18

[87] D. Bellamy and L. Pravica, "Assessing the impact of driverless haul trucks in Australian surface mining," *Resources Policy*, vol. 36, pp. 149–158, June 2011. 18

[88] J. Barnes, C. Rizos, J. Wang, D. Small, G. Voigt, and N. Gambale, "Locata: A new positioning technology for high precision indoor and outdoor positioning," *Proceedings, ION GNSS, Portland, OR, CD ROM*, pp. 1119–1128, 2003. 18

[89] A. Chehri, P. Fortier, and P. M. Tardif, "UWB-based sensor networks for localization in mining environments," *Ad Hoc Networks*, vol. 7, pp. 987–1000, July 2009. 18

[90] A. Nuchter, H. Surmann, K. Lingemann, J. Hertzberg, and S. Thrun, "6d SLAM with an application in autonomous mine mapping," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 2, pp. 1998–2003, 2004. 18

[91] W. Wang, W. Dong, Y. Su, D. Wu, and Z. Du, "Development of Search-and-Rescue Robots for Underground Coal Mine Applications: Search and Rescue Robot for Underground Coal Mine," *Journal of Field Robotics*, vol. 31, pp. 386–407, May 2014. 18

[92] J. P. Grotzinger, J. Crisp, A. R. Vasavada, R. C. Anderson, C. J. Baker, R. Barry, D. F. Blake, P. Conrad, K. S. Edgett, B. Ferdowski, R. Gellert, J. B. Gilbert, M. Golombek, J. Gómez-Elvira, D. M. Hassler, L. Jandura, M. Litvak, P. Mahaffy, J. Maki, M. Meyer, M. C. Malin, I. Mitrofanov, J. J. Simmonds, D. Vaniman, R. V. Welch, and R. C. Wiens, "Mars Science Laboratory Mission and Science Investigation," *Space Science Reviews*, vol. 170, pp. 5–56, Sept. 2012. 19

[93] I. Thomson, "NASA's Opportunity rover celebrates 10 years on Mars," 2014. 19

[94] L. Matthies, "Stereo vision for planetary rovers: Stochastic modeling to near real-time implementation," *International Journal of Computer Vision*, vol. 8, no. 1, pp. 71–91, 1992. 19

[95] R. Li, K. Di, A. Howard, L. Matthies, J. Wang, and S. Agarwal, "Rock modeling and matching for autonomous long-range Mars rover localization," *Journal of Field Robotics*, vol. 24, pp. 187–203, Mar. 2007. 19

[96] M. Maimone, Y. Cheng, and L. Matthies, "Two years of Visual Odometry on the Mars Exploration Rovers," *Journal of Field Robotics*, vol. 24, pp. 169–186, Mar. 2007. 19

[97] J. Leitner, "Multi-robot Cooperation in Space: A Survey," in *Advanced Technologies for Enhanced Quality of Life, 2009. AT-EQUAL '09.*, pp. 144–151, July 2009. 19

[98] R. A. Brooks and A. M. Flynn, "Fast, Cheap and Out of Control," tech. rep., Dec. 1989. 19

[99] J. Castillo-Rogez, M. Pavone, I. Nesnas, and J. Hoffman, "Expected science return of spatially-extended in-situ exploration at small Solar system bodies," in *2012 IEEE Aerospace Conference*, pp. 1–15, 2012. 19

[100] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, ISBN: 978-0387310732, 2007. 19, 56, 68, 73, 74, 75, 150, 158

[101] Y. Bar-Shalom, X. Rong Li, and T. Kirubarajan, *Estimation with Applications to Tracking and Navigation*. Wiley-Interscience, June 2001. 19, 24, 58

[102] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Fluids Engineering*, vol. 82, no. 1, pp. 35–45, 1960. 24, 57, 58

[103] H. Durrant-Whyte and T. Bailey, "Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms," *Robotics and Automation Magazine*, vol. 13, pp. 99–110, 2006. 24, 25, 119

[104] T. Bailey and H. Durrant-Whyte, "Simultaneous Localisation and Mapping (SLAM): Part II State of the Art," *Robotics and Automation Magazine*, vol. 13, pp. 108–117, 2006. 24, 25, 60, 81, 119

[105] V. G. Ivancevic and T. T. Ivancevic, "Lecture Notes in Lie Groups," *arXiv preprint arXiv:1104.1106*, 2011. 26

[106] R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. CRC press, 1994. 26, 28

[107] W. Fulton and J. Harris, *Representation Theory: A First Course*. New York: Springer, corrected edition ed., Aug. 1999. 26

[108] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," *Autonomous Robot Vehicles*, 1990. 27, 28

[109] E. Olson, *Robust and Efficient Robotic Mapping*. PhD thesis, Massachusetts Institude Of Technology, 2008. 29, 67, 78

[110] J. D. Tardós, J. Neira, P. M. Newman, and J. J. Leonard, "Robust mapping and localization in indoor environments using sonar data," *The International Journal of Robotics Research*, vol. 21, no. 4, p. 311, 2002. 29, 32, 48, 49, 59, 81

[111] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, second ed., 2003. 30, 33, 65, 77

[112] J. J. Leonard and H. F. Durrant-Whyte, *Directed sonar sensing for mobile robot navigation*, vol. 448. Kluwer Academic Publishers Dordrecht, 1992. 32

[113] J. Ring, "The laser in astronomy," *New Scientist*, vol. 18, p. 344, 1963. 32

[114] G. Weiss and E. V. Puttkamer, "A Map Based On Laserscans Without Geometric Interpretation," tech. rep., CiteSeerX, 1995. 32

[115] F. Lu and E. Milios, "Robot pose estimation in unknown environments by matching 2d range scans," *Journal of Intelligent and Robotic Systems*, vol. 18, no. 3, pp. 249–275, 1997. 32, 38

[116] P. Newman, D. Cole, and K. Ho, "Outdoor SLAM using visual appearance and laser ranging," 2006. 32

[117] M. Pfingsthorn and A. Birk, "Simultaneous localization and mapping with multimodal probability distributions," *The International Journal of Robotics Research*, vol. 32, pp. 143–171, Feb. 2013. 32, 81, 153, 160, 161, 214, 216, 223

[118] M. Bosse and R. Zlot, "Continuous 3d scan-matching with a spinning 2d laser," in *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pp. 4312–4319, 2009. 32

[119] J. Bohren, T. Foote, J. Keller, A. Kushleyev, D. Lee, A. Stewart, P. Vernaza, J. Derenick, J. Spletzer, and B. Satterfield, "Little Ben: The Ben Franklin Racing Team's entry in the 2007 DARPA Urban Challenge," *Journal of Field Robotics*, vol. 25, pp. 598–614, Sept. 2008. 33

[120] S. Thrun, "Google's driverless car," *TED Talk*, 2011. 33

[121] F. Amzajerdian, M. Vanek, L. Petway, D. Pierrottet, G. Busch, and A. Bulyshev, "Utilization of 3-D imaging flash lidar technology for autonomous safe landing on planetary bodies," in *Proc. of SPIE Vol*, vol. 7608, pp. 760828–1, 2010. 33

[122] W. Zhou, V. Miro, and G. Dissanayake, "Information-driven 6d SLAM based on ranging vision," in *Intelligent Robots and Systems*, pp. 2072–2077, 2008. 33

[123] G. Von Wichert, "Mobile robot localization using a selforganized visual environment representation," in *Advanced Mobile Robots, 1997. Proceedings., Second EUROMICRO workshop on*, pp. 29–36, IEEE, 1997. 33, 53, 81

[124] A. Davison, I. Reid, N. Molton, and O. Stasse, "MonoSLAM: Real-Time Single Camera SLAM," *IEEE Trans. On Pattern Analysis And Machine Intelligence*, 2007. 33, 45, 46

[125] J. Civera, A. Davison, and J. Montiel, "1-Point RANSAC for EKF Filtering. Application to Real-Time Structure from Motion and Visual Odometry," *Journal of Field Robotics*, vol. to appear, Sept. 2010. 33, 46, 60, 220

[126] T. Botterill, S. Mills, and R. D. Green, "Correcting Scale Drift by Object Recognition in Single-Camera SLAM.," *IEEE T. Cybernetics*, vol. 43, no. 6, pp. 1767–1780, 2013. 33

[127] K. Boyer and A. Kak, "Color-Encoded Structured Light for Rapid Active Ranging," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, pp. 14–28, Jan. 1987. 33

[128] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of kinect depth data for indoor mapping applications," *Sensors*, vol. 12, no. 2, pp. 1437–1454, 2012. 33, 81

[129] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 127–136, 2011. 33, 81, 220

[130] G. Reina, J. Underwood, G. Brooker, and H. Durrant-Whyte, "Radar-based perception for autonomous outdoor vehicles," *Journal of Field Robotics*, 2011. 34, 81

[131] J. Mullane and E. Jose, *Robotic Navigation and Mapping with Radar*. Artech House, 2012. 34

[132] D. Vivet, F. Gerossier, P. Checchin, L. Trassoudaine, and R. Chapuis, "Mobile Ground-Based Radar Sensor for Localization and Mapping: An Evaluation of two Approaches," *International Journal of Advanced Robotic Systems*, p. 1, 2013. 34

[133] F. Pomerleau, A. Breitenmoser, M. Liu, F. Colas, and R. Siegwart, "Noise characterization of depth sensors for surface inspections," in *Applied Robotics for the Power Industry (CARPI), 2012 2nd International Conference on*, pp. 16–21, IEEE, 2012. 34, 35, 81

[134] H. Alismail, L. D. Baker, and B. Browning, "Continuous Trajectory Estimation for 3d SLAM from Actuated Lidar," in *ICRA*, 2014. 36, 81

[135] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, ISBN: 978-0262201629, 2005. 36, 37, 40, 41, 47, 53, 58, 59, 66, 81, 119, 140

[136] M. Ruhnke, R. Kummerle, G. Grisetti, and W. Burgard, "Highly Accurate Maximum Likelihood Laser Mapping by Jointly Optimizing Laser Points and Robot Poses," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2812 –2817, May 2011. 36, 79, 81

[137] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta Informatica*, vol. 4, pp. 1–9, Mar. 1974. 38, 81

[138] P. Besl and H. McKay, "A method for registration of 3-D shapes," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 14, no. 2, pp. 239–256, 1992. 38

[139] A. Censi, "An ICP variant using a point-to-line metric," in *IEEE International Conference on Robotics and Automation, 2008. ICRA 2008*, pp. 19–25, May 2008. 38

[140] J. Guivant, E. Nebot, and S. Baiker, "Autonomous Navigation and Map building Using Laser Range Sensors in Outdoor Applications," *Journal of Robotic Systems*, vol. 17, no. 10, pp. 565–583, 2000. 39, 63, 81

[141] J. E. Guivant and E. M. Nebot, "Optimization of the Simultaneous Localization and Map-Building Algorithm for Real-Time Implementation," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 242–257, 2001. 39, 61

[142] T. Bailey, *Mobile Robot Localisation and Mapping in Extensive Outdoor Environments*. PhD thesis, Australian Center for Field Robotics, University of Sydney, 2002. 39, 60, 81, 145, 153

[143] M. R. Walter, R. M. Eustice, and J. J. Leonard, "Exactly Sparse Extended Information Filters for Feature-based SLAM," *The International Journal of Robotics Research*, vol. 26, pp. 335–359, Apr. 2007. 39

[144] J. Neira and J. Tardós, "Data association in stochastic mapping using the joint compatibility test," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 890–897, 2001. 39, 60, 81, 153

[145] Y.-H. Choi, T.-K. Lee, and S.-Y. Oh, "A line feature based SLAM with low grade range sensors using geometric constraints and active exploration for mobile robot," *Autonomous Robots*, vol. 24, pp. 13–27, Jan. 2008. 39, 81

[146] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *1985 IEEE International Conference on Robotics and Automation. Proceedings*, vol. 2, pp. 116–121, Mar. 1985. 39

[147] A. Elfes, "Sonar-based real-world mapping and navigation," *IEEE Journal of Robotics and Automation*, vol. 3, pp. 249–265, June 1987. 39, 40

[148] J. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, no. 1, pp. 25–30, 1965. 41

[149] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: an efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, pp. 189–206, Apr. 2013. 42, 81, 221

[150] G. Dudek and M. Jenkin, *Computational Principles of Mobile Robotics*. Cambridge University Press, July 2010. 43, 45, 81

[151] G. Campion, G. Bastin, and B. D'Andrea-Novel, "Structural properties and classification of kinematic and dynamic models of wheeled mobile robots," *IEEE transactions on robotics and automation*, vol. 12, no. 1, pp. 47–62, 1996. 43, 81

[152] R. Mahony, T. Hamel, and J.-M. Pflimlin, "Nonlinear Complementary Filters on the Special Orthogonal Group," *IEEE Transactions on Automatic Control*, vol. 53, pp. 1203–1218, June 2008. 45

[153] H. Chung, L. Ojeda, and J. Borenstein, "Accurate mobile robot dead-reckoning with a precision-calibrated fiber-optic gyroscope," *IEEE Transactions on Robotics and Automation*, vol. 17, pp. 80–84, Feb. 2001. 45

[154] D. Nistér, "An efficient solution to the five-point relative pose problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 756–770, 2004. 46

[155] J. A. Castellanos, J. Neira, and J. D. Tardós, "Limits to the Consistency of EKF Based SLAM," 2004. 48, 60, 61, 81

[156] T. Bailey, J. Nieto, J. Guivant, M. Stevens, and E. Nebot, "Consistency of the EKF-SLAM algorithm," in *International Conference on Intelligent Robots and Systems*, 2006. 48, 60, 61, 81

[157] T. Bailey, J. Nieto, and E. Nebot, "Consistency of the FastSLAM Algorithm," in *IEEE Int Conference on Robotics and Automation*, 2006. 48, 64, 81

[158] K. Iagnemma and C. C. Ward, "Classification-based wheel slip detection and detector fusion for mobile robots on outdoor terrain," *Autonomous Robots*, vol. 26, pp. 33–46, Jan. 2009. 50, 81

[159] V. Kubelka, L. Oswald, F. Pomerleau, F. Colas, T. Svoboda, and M. Reinstein, "Robust Data Fusion of Multimodal Sensory Information for Mobile Robots," *Journal of Field Robotics*, Aug. 2014. 50

[160] M. Li and A. I. Mourikis, "High-precision, consistent EKF-based visual-inertial odometry," *The International Journal of Robotics Research*, vol. 32, pp. 690–711, May 2013. 50, 220

[161] R. B. Langley, "Dilution of precision," *GPS world*, vol. 10, no. 5, pp. 52–59, 1999. 51

[162] A. El-Rabbany, *Introduction to GPS: The Global Positioning System*. Artech House, Jan. 2002. 51

[163] J. A. Farrell, T. D. Givargis, and M. J. Barth, "Real-time differential carrier phase GPS-aided INS," *Control Systems Technology, IEEE Transactions on*, vol. 8, no. 4, pp. 709–721, 2000. 51

[164] M. Joerger and B. Pervan, "Measurement-Level Integration of Carrier-Phase GPS and Laser-Scanner for Outdoor Ground Vehicle Navigation," *Journal of Dynamic Systems, Measurement, and Control*, vol. 131, pp. 021004–11, Mar. 2009. 51, 81

[165] M. Fiala, "Comparing ARTag and ARToolkit Plus fiducial marker systems," Oct. 2005. 52, 54

[166] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3400–3407, IEEE, May 2011. 52, 54, 81, 231

[167] S. Engelson and D. McDermott, "Error correction in mobile robot map learning," in *IEEE International Conference on Robotics and Automation, 1992. Proceedings*, pp. 2555–2560 vol.3, May 1992. 53

[168] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo localization for mobile robots," *Artificial Intelligence*, vol. 128, pp. 99–141, May 2001. 53

[169] D. Fox, S. Thrun, W. Burgard, and F. Dellaert, "Particle Filters for Mobile Robot Localization," in *Sequential Monte Carlo Methods in Practice* (A. Doucet, N. d. Freitas, and N. Gordon, eds.), Statistics for Engineering and Information Science, pp. 401–428, Springer New York, Jan. 2001. 53

[170] M. Cummins and P. Newman, "FAB-MAP: Probabilistic localization and mapping in the space of appearance," *The International Journal of Robotics Research*, vol. 27, no. 6, p. 647, 2008. 53, 81, 95

[171] M. Cummins, *Probabilistic Localization and Mapping in Appearance Space*. PhD thesis, University of Oxford, 2009. 53, 81, 221

[172] M. Cummins and P. Newman, "Appearance-only SLAM at large scale with FAB-MAP 2.0," *The International Journal of Robotics Research*, Nov. 2010. 53, 54, 81, 94, 95, 221

[173] T. Botterill, *Visual Navigation for Mobile Robots using the Bag-of-Words Algorithm*. PhD thesis, University of Canterbury, 2010. 53, 81, 221

[174] S. Siltanen and T. T. Valtion, *Theory and applications of marker-based augmented reality*. 2012. 54, 81

[175] D. S. Berkovitz, *System characterization and online mass property identification of the SPHERES formation flight testbed*. PhD thesis, Massachusetts Institute of Technology, 2008. 54

[176] A. Angeli, D. Filliat, S. Doncieux, and J.-A. Meyer, "Fast and Incremental Method for Loop-Closure Detection Using Bags of Visual Words," *IEEE Transactions on Robotics*, vol. 24, pp. 1027–1037, Oct. 2008. 55, 81

[177] G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," in *IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2007. 55

[178] Z. Chen, "Bayesian filtering: From Kalman filters to particle filters, and beyond," *Statistics*, vol. 182, no. 1, pp. 1–69, 2003. 57

[179] J. A. Castellanos, J. M. M. Montiel, J. Neira, and J. D. Tardós, "The SPmap: A probabilistic framework for simultaneous localization and map building," *Robotics and Automation, IEEE Transactions on*, vol. 15, no. 5, pp. 948–952, 1999. 58

[180] L. Paz, J. Tardós, and J. Neira, "Divide and Conquer EKF SLAM in O(n)," *IEEE Transactions on Robotics*, vol. 24, pp. 1107–1120, Oct. 2008. 59, 60, 61, 81

[181] J. A. Castellanos, R. Martinez-Cantin, J. D. Tardós, and J. Neira, "Robocentric map joining: Improving the consistency of EKF-SLAM," *Robotics and Autonomous Systems*, vol. 55, no. 1, pp. 21–29, 2007. 60, 61, 81

[182] B. Williams and I. Reid, "On combining visual SLAM and visual odometry," in *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3494–3500, IEEE, May 2010. 60

[183] M. Chli and A. J. Davison, "Active Matching for Visual Tracking," *Robotics and Autonomous Systems*, vol. 57, pp. 1173–1187, Dec. 2009. 60, 81

[184] R. Reid, "Jointly Compatible Pair Linking for Visual Tracking with Probabilistic Priors," in *Australasian Computer Science Conference (ACSC 2011)*, vol. 113 of *CRPIT*, (Perth, Australia), pp. 35–42, ACS, 2011. 60, 81

[185] V. Bonato, E. Marques, and G. A. Constantinides, "A floating-point extended kalman filter implementation for autonomous mobile robots," *Journal of Signal Processing Systems*, vol. 56, no. 1, pp. 41–50, 2009. 60, 81

[186] J. J. Leonard and H. J. Feder, "A computationally efficient method for large-scale concurrent mapping and localization," in *Robotics Research: the Ninth International Symposium*, vol. 9, pp. 169–178, 2000. 61, 93, 95

[187] J. Knight, A. Davison, and I. Reid, "Towards constant time SLAM using postponement," in *Proc. IEEE/RSJ Conf. on Intelligent Robots and Systems*, pp. 406–412, 2001. 61

[188] P. Piniés, L. M. Paz, D. Gálvez-López, and J. D. Tardós, "CI-Graph simultaneous localization and mapping for three-dimensional reconstruction of large and complex environments using a multicamera system," *Journal of Field Robotics*, vol. 27, pp. 561–586, Aug. 2010. 61, 81, 95

[189] S. B. Williams, G. Dissanayake, and H. Durrant-Whyte, "Towards multi-vehicle simultaneous localisation and mapping," vol. 3, pp. 2743–2748, 2002. 61, 88

[190] S. Williams, G. Dissanayake, and H. Durrant-Whyte, "An efficient approach to the simultaneous localisation and mapping problem," in *IEEE International Conference on Robotics and Automation, 2002. Proceedings. ICRA '02*, vol. 1, pp. 406–411 vol.1, 2002. 61

[191] X. S. Zhou and S. I. Roumeliotis, "Multi-robot SLAM with unknown initial correspondence: The robot rendezvous case," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 1785–1792, IEEE, 2006. 61, 81

[192] G. P. Huang, N. Trawny, A. I. Mourikis, and S. I. Roumeliotis, "On the consistency of multi-robot cooperative localization," *MARS Lab, University of Minnesota, Minneapolis, MN, Tech. Rep., Jan*, 2009. 61, 88

[193] H. Strasdat, J. M. M. Montiel, and A. J. Davison, "Real-Time Monocular SLAM: Why Filter?," in *Int. Conf. on Robotics and Automation*, 2010. 61, 81

[194] S. Julier, J. Uhlmann, and H. Durrant-Whyte, "A New Method for the Nonlinear Transformation of Means and Covariances in Filters and Estimators," *IEEE Transactions on Automatic Control*, vol. 45, pp. 477 –482, Mar. 2000. 61, 81

[195] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004. 61, 81

[196] S. Holmes, G. Klein, and D. W. Murray, "A square root unscented Kalman filter for visual monoSLAM," in *Proc Int Conf on Robotics and Automation*, pp. 3710–3716, 2008. 61

[197] S. Julier and J. Uhlmann, "A non-divergent estimation algorithm in the presence of unknown correlations," in *American Control Conference, 1997. Proceedings of the 1997*, vol. 4, pp. 2369–2373 vol.4, June 1997. 62

[198] S. J. Julier and J. K. Uhlmann, "Using covariance intersection for SLAM," *Robotics and Autonomous Systems*, vol. 55, no. 1, pp. 3–20, 2007. 62

[199] S. Thrun, Y. Liu, D. Koller, A. Y. Ng, Z. Ghahramani, and H. Durrant-Whyte, "Simultaneous Localization and Mapping with Sparse Extended Information Filters," *The International Journal of Robotics Research*, vol. 23, pp. 693–716, Aug. 2004. 62, 78

[200] R. M. Eustice, H. Singh, and J. J. Leonard, "Exactly sparse delayed-state filters for view-based SLAM," *IEEE Transactions on Robotics*, vol. 22, no. 6, pp. 1100–1114, 2006. 62, 63

[201] Y. Liu and S. Thrun, "Results for outdoor-SLAM using sparse extended information filters," in *International Conference on Robotics and Automation*, vol. 1, pp. 1227–1233, 2003. 62, 63

[202] I. Mahon, S. B. Williams, O. Pizarro, and M. Johnson-Roberson, "Efficient view-based SLAM using visual loop closures," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1002–1014, 2008. 63, 66

[203] S. Thrun and Y. Liu, "Multi-Robot SLAM With Sparse Extended Information Filters," in *Robotics Research: The Eleventh International Symposium*, 2005. 63, 81

[204] M. A. Paskin, "Thin junction tree filters for simultaneous localization and mapping," *Computer*, 2002. 63, 78, 81, 96

[205] U. Frese, "A proof for the approximate sparsity of SLAM information matrices," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 329–335, IEEE, 2005. 63

[206] U. Frese, "Treemap: An O (log n) algorithm for indoor simultaneous localization and mapping," *Autonomous Robots*, vol. 21, no. 2, pp. 103–122, 2006. 63

[207] U. Frese and L. Schroder, "Closing a million-landmarks loop," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 5032–5039, 2006. 63, 81, 95, 96

[208] K. P. Murphy, "Bayesian Map Learning in Dynamic Environments.," in *NIPS*, pp. 1015–1021, 1999. 64, 119

[209] D. Hahnel, W. Burgard, D. Fox, and S. Thrun, "An efficient FastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements," in *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings*, vol. 1, pp. 206 – 211 vol.1, Oct. 2003. 64, 81, 95, 119

[210] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem," in *Proceedings Of The National Conference On Artificial Intelligence*, pp. 593–598, 2002. 64

[211] 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 *International Joint Conference On Artificial Intelligence*, vol. 18, 2003. 64

[212] M. Montemerlo and S. Thrun, "FastSLAM 2.0," *FastSLAM: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics*, pp. 63–90, 2007. 64, 81

[213] G. Grisetti, C. Stachniss, and W. Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007. 64

[214] D. Fox, W. Burgard, H. Kruppa, and S. Thrun, "A Probabilistic Approach to Collaborative Multi-Robot Localization," *Autonomous Robots*, vol. 8, pp. 325–344, June 2000. 65

[215] L. Carlone, M. Kaouk Ng, J. Du, B. Bona, and M. Indri, "Simultaneous Localization and Mapping Using Rao-Blackwellized Particle Filters in Multi Robot Systems," *Journal of Intelligent & Robotic Systems*, 2010. 65

[216] G. Sibley, L. Matthies, and G. Sukhatme, "Sliding Window Filter with Application to Planetary Landing," *Journal of Field Robotics*, vol. 27, pp. 587–608, July 2010. 65, 81

[217] P. Newman, G. Sibley, M. Smith, M. Cummins, A. Harrison, C. Mei, I. Posner, R. Shade, D. Schroeter, D. Cole, and I. Reid, "Navigating, recognizing and describing urban spaces with vision and lasers," *The International Journal of Robotics Research*, vol. 28, no. 11-12, p. 1406, 2009. 66, 81, 220

[218] C. Bibby and I. Reid, "Simultaneous localisation and mapping in dynamic environments (SLAMIDE) with reversible data association," in *Proceedings of Robotics: Science and Systems*, 2007. 66, 81

[219] C. Mei, G. Sibley, M. Cummins, P. Newman, and I. Reid, "A Constant-Time Efficient Stereo SLAM System," in *Proceedings of the British Machine Vision Conference (BMVC)*, 2009. 66

[220] C. Mei, G. Sibley, M. Cummins, P. Newman, and I. Reid, "RSLAM: A System for Large-Scale Mapping in Constant-Time Using Stereo," *International Journal of Computer Vision*, pp. 1–17, 2010. 66, 81, 86, 94, 95, 220

[221] H. Strasdat, A. Davison, J. Montiel, and K. Konolige, "Double window optimisation for constant time visual SLAM," in *2011 IEEE International Conference on Computer Vision (ICCV)*, pp. 2352 –2359, Nov. 2011. 66, 83, 220, 223

[222] K. Konolige, J. Bowman, J. D. Chen, P. Mihelich, M. Calonder, V. Lepetit, and P. Fua, "View-based maps," *The International Journal of Robotics Research*, 2010. 66

[223] M. Kaess and F. Dellaert, "Covariance Recovery from a Square Root Information Matrix for Data Association," *Robotics and Autonomous Systems*, 2009. 66, 81, 120, 223

[224] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, vol. 4, no. 4, pp. 333–349, 1997. 67, 81, 89, 120

[225] J.-S. Gutmann and K. Konolige, "Incremental mapping of large cyclic environments," in *1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation, 1999. CIRA '99. Proceedings*, pp. 318 –325, 1999. 67, 120

[226] A. Howard, M. Matark, and G. Sukhatme, "Localization for mobile robot teams using maximum likelihood estimation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002*, vol. 1, pp. 434–439 vol.1, 2002. 67, 120

[227] A. Howard and L. Kitchen, "Cooperative Localisation and Mapping," in *International Conference on Field and Service Robotics*, pp. 92–97, 1999. 67, 88

[228] S. Thrun and M. Montemerlo, "The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures," *The International Journal of Robotics Research*, vol. 25, pp. 403–429, May 2006. 67, 81, 90, 95

[229] F. Dellaert, J. Carlson, V. Ila, K. Ni, and C. E. Thorpe, "Subgraph-preconditioned conjugate gradients for large scale slam," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 2566–2571, IEEE, 2010. 67, 68, 81, 95

[230] A. Howard, M. J. Mataric, and G. Sukhatme, "Relaxation on a mesh: a formalism for generalized localization," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 2, pp. 1055–1060, IEEE, 2001. 67, 105

[231] K. Konolige, "Large-scale map-making," in *Proceedings of the 19th national conference on Artifical intelligence*, AAAI'04, pp. 457–463, AAAI Press, 2004. 67

[232] J. Folkesson and H. Christensen, "Graphical SLAM - A Self-Correcting Map," in *IEEE International Conference on Robotics and Automation*, vol. 1, 2004. 67

[233] E. Olson, J. Leonard, and S. Teller, "Fast Iterative Alignment of Pose Graphs with Poor Initial Estimates," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2262–2269, 2006. 67, 78, 81, 90, 200

[234] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon, "Bundle Adjustment - A Modern Synthesis," *Lecture Notes In Computer Science*, pp. 298–372, 1999. 67, 76, 77, 126, 155

[235] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Fast incremental smoothing and mapping with efficient data association," in *2007 IEEE International Conference on Robotics and Automation*, pp. 1670–1677, 2007. 67, 77, 81, 95

[236] M. Kaess, *Incremental Smoothing and Mapping*. Ph.D., Georgia Institute of Technology, Dec. 2008. 67, 77, 81, 95, 200, 205

[237] K. Konolige, G. Grisetti, R. Kummerle, B. Limketkai, and R. Vincent, "Efficient Sparse Pose Adjustment for 2d Mapping," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010. 67, 76, 77, 81, 83

[238] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A General Framework for Graph Optimization," in *IEEE Int. Conf. on Robotics and Automation*, 2011. 67, 73, 76, 77, 95, 117, 125, 200, 216

[239] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the Bayes tree," *The International Journal of Robotics Research*, 2012. 68, 81, 95, 96, 222

[240] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, pp. 498–519, Feb. 2001. 68

[241] F. Dellaert, "Factor graphs and GTSAM: A hands-on introduction," 2012. 68

[242] P. C. Mahalanobis, "On the generalized distance in statistics," in *Proceedings of the National Institute of Sciences of India*, 1936. 71

[243] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A Tutorial on Graph-Based SLAM," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31 –43, 2010. 75, 83, 120, 125, 166

[244] N. Sunderhauf, *Robust Optimization for Simultaneous Localization and Mapping*. PhD thesis, TU Chemnitz, 2012. 76, 81, 159, 160, 200

[245] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg, "Hierarchical optimization on manifolds for online 2d and 3d mapping," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2010. 77, 81

[246] M. Bosse, P. Newman, J. Leonard, M. Soika, W. Feiten, and S. Teller, "An Atlas framework for scalable mapping," in *IEEE International Conference on Robotics and Automation*, vol. 2, 2003. 78, 79, 95, 107

[247] M. Bosse, P. Newman, J. Leonard, and S. Teller, "Simultaneous Localization and Map Building in Large-Scale Cyclic Environments Using the Atlas Framework," *The International Journal of Robotics Research*, vol. 23, no. 12, p. 1113, 2004. 78, 79, 81, 95, 159

[248] M. Bosse, *ATLAS: A Framework for Large Scale Automated Mapping and Localization*. PhD thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 2004. 78, 79, 95, 153

[249] C. Estrada, J. Neira, and J. Tardós, "Hierarchical SLAM: real-time accurate mapping of large environments," *IEEE Transactions on Robotics*, vol. 21, pp. 588–596, Aug. 2005. 79, 90, 93, 95

[250] C. Estrada, J. Neira, and J. Tardós, "Finding good cycle constraints for large scale multi-robot SLAM," in *IEEE International Conference on Robotics and Automation, 2009. ICRA '09*, pp. 395 –402, May 2009. 79, 81, 90, 93, 95, 153

[251] K. Ni, D. Steedly, and F. Dellaert, "Tectonic SAM: Exact, Out-of-Core, Submap-Based SLAM," in *2007 IEEE International Conference on Robotics and Automation*, pp. 1678 –1685, Apr. 2007. 79, 81, 95

[252] K. Ni and F. Dellaert, "Multi-level submap based SLAM using nested dissection," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 2558–2565, IEEE, 2010. 79, 95

[253] J. Blanco, J. Fernandez-Madrigal, and J. Gonzalez, "Towards a Unified Bayesian Approach to Hybrid Metric–Topological SLAM," *IEEE Transactions on Robotics*, vol. 24, pp. 259–270, Apr. 2008. 79, 81, 95, 153

[254] M. Montemerlo and S. Thrun, "Simultaneous localization and mapping with unknown data association using FastSLAM," in *IEEE International Conference on Robotics and Automation, 2003. Proceedings. ICRA'03*, vol. 2, 2003. 81

[255] D. Hähnel, S. Thrun, B. Wegbreit, and W. Burgard, "Towards lazy data association in SLAM," in *International Symposium on Robotics Research (ISRR)*, pp. 421–431, 2003. 81

[256] E. Olson, "Recognizing places using spectrally clustered local matches," *Robotics and Autonomous Systems*, vol. 57, no. 12, pp. 1157–1172, 2009. 81, 153, 159, 160

[257] A. Handa, M. Chli, H. Strasdat, and A. Davison, "Scalable Active Matching," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2010. 81

[258] J. Sola, *Towards Visual Localization, Mapping and Moving Objects Tracking by a Mobile Robot: a Geometric and Probabilistic Approach.* PhD thesis, 2007. 81

[259] C. C. Wang, C. Thorpe, S. Thrun, M. Hebert, and H. Durrant-Whyte, "Simultaneous localization, mapping and moving object tracking," *The International Journal of Robotics Research*, vol. 26, no. 9, pp. 889–916, 2007. 81

[260] S. J. Davey, "Simultaneous localization and map building using the probabilistic multi-hypothesis tracker," *Robotics, IEEE Transactions on*, vol. 23, no. 2, pp. 271–280, 2007. 81, 160

[261] M. Bosse and R. Zlot, "Keypoint design and evaluation for place recognition in 2d lidar maps," *Robot. Auton. Syst.*, vol. 57, pp. 1211–1224, Dec. 2009. 81

[262] Y. Latif, C. Cadena, and J. Neira, "Realizing, Reversing, Recovering: Incremental Robust Loop Closing over time using the iRRR algorithm," in *IEEE/RSJ Int.Conf.on Intelligent Robots and Systems*, 2012. 81, 153

[263] Y. Latif, C. Cadena, and J. Neira, "Robust Loop Closing Over Time," in *Proceedings of Robotics: Science and Systems (RSS)*, 2012. 81, 159

[264] K. Konolige and M. Agrawal, "FrameSLAM: From bundle adjustment to real-time visual mapping," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1066–1077, 2008. 81, 95

[265] C. Früh and A. Zakhor, "An automated method for large-scale, ground-based city model acquisition," *International Journal of Computer Vision*, vol. 60, no. 1, pp. 5–24, 2004. 81, 100, 132

[266] R. Kümmerle, B. Steder, C. Dornhege, A. Kleiner, G. Grisetti, and W. Burgard, "Large Scale Graph-Based SLAM Using Aerial Images as Prior Information," *Autonomous Robots*, vol. 30, no. 1, pp. 25–39, 2011. 81, 95, 100, 132

[267] T. Senlet and A. Elgammal, "Satellite image based precise robot localization on sidewalks," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 2647–2653, IEEE, 2012. 81, 100, 132

[268] J. Neira, J. Tardós, and J. Castellanos, "Linear time vehicle relocation in SLAM," in *IEEE International Conference on Robotics and Automation*, vol. 1, pp. 427–433, 2003. 81

[269] B. Williams, G. Klein, and I. Reid, "Real-time SLAM Relocalisation," in *IEEE International Conference on Computer Vision*, pp. 1–8, 2007. 81

[270] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard, "A Tree Parameterization for Efficiently Computing Maximum Likelihood Maps Using Gradient Descent," in *Proc. of Robotics: Science and Systems (RSS)*, 2007. 81

[271] G. Grisetti, C. Stachniss, and W. Burgard, "Non-linear constraint network optimization for efficient map learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 3, pp. 428–439, 2009. 81

[272] S. Huang, Y. Lai, U. Frese, and G. Dissanayake, "How far is SLAM from a linear least squares problem?," in *International Conference on Intelligent Robots and Systems*, 2010. 81

[273] H. Wang, G. Hu, S. Huang, and G. Dissanayake, "On the Structure of Nonlinearities in Pose Graph SLAM," in *2012 Robotics: Science and Systems Conference (RSS)*, 2012. 81

[274] V. Ila, J. M. Porta, and J. Andrade-Cetto, "Information-Based Compact Pose SLAM," *IEEE Transactions on Robotics*, 2010. 81, 96, 222

[275] C. Stachniss and H. Kretzschmar, "Pose Graph Compression for Laser-Based SLAM," in *The 15th International Symposium on Robotics Research (ISRR)*, 2011. 81, 222

[276] N. Sünderhauf and P. Protzel, "Switchable Constraints vs. Max-Mixture Models vs. RRR–A Comparison of Three Approaches to Robust Pose Graph SLAM," 2012. 81, 159, 160

[277] Y. Latif, C. Cadena, and J. Neira, "Robust Graph SLAM Back-ends: A Comparative Analysis," 2014. 81, 153

[278] D. Rosen, M. Kaess, and J. Leonard, "Robust incremental online inference over sparse factor graphs: Beyond the Gaussian case," in *Proc. of Intl. Conf. on Robotics and Automation (ICRA)*, 2013. 81

[279] K. Konolige and K. Chou, "Markov localization using correlation," in *International Joint Conference on Artificial Intelligence*, vol. 16, pp. 1154–1159, 1999. 81, 145

[280] S. Carpin, "Fast and accurate map merging for multi-robot systems," *Autonomous Robots*, vol. 25, no. 3, pp. 305–316, 2008. 81, 139, 145

[281] M. Bosse and R. Zlot, "Map Matching and Data Association for Large-Scale Two-dimensional Laser Scan-based SLAM," *The International Journal of Robotics Research*, vol. 27, pp. 667 –691, June 2008. 81, 145

[282] E. B. Olson, "Real-time correlative scan matching," in *IEEE International Conference on Robotics and Automation*, 2009. 81, 146, 150, 212

[283] P. Piniés and J. D. Tardós, "Scalable SLAM Building Conditionally Independent Local Maps," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3466–3471, 2007. 81

[284] P. Piniés and J. Tardós, "Large-Scale SLAM Building Conditionally Independent Local Maps: Application to Monocular Vision," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1094–1106, 2008. 81, 95

[285] M. Milford and G. Wyeth, "Persistent Navigation and Mapping using a Biologically Inspired SLAM System," *The International Journal of Robotics Research*, vol. 29, pp. 1131–1153, Aug. 2010. 81, 221

[286] L. A. Andersson and J. Nygards, "On multi-robot map fusion by inter-robot observations," in *Information Fusion, 2009. FUSION'09. 12th International Conference on*, pp. 1712–1721, IEEE, 2009. 81, 90, 93

[287] M. Haley, "Multi-Autonomous Ground-robotic International Challenge (MAGIC) 2010 The Chiba Team," tech. rep., Analytical Software Inc, 2010. 81

[288] S. Sukkarieh, E. Nettleton, J.-H. Kim, M. Ridley, A. Goktogan, and H. Durrant-Whyte, "The ANSER Project: Data Fusion Across Multiple Uninhabited Air Vehicles," *The International Journal of Robotics Research*, vol. 22, pp. 505–539, July 2003. 81, 87, 88, 89, 92

[289] I. Rekleitis, V. Lee-Shue, A. P. New, and H. Choset, "Limited communication, multi-robot team based coverage," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 4, pp. 3462–3468, IEEE, 2004. 81

[290] E. Nettleton, S. Thrun, H. Durrant-Whyte, and S. Sukkarieh, "Decentralised SLAM with Low-Bandwidth Communication for Teams of Vehicles," in *Field and Service Robotics*, pp. 179–188, 2006. 81, 88

[291] N. Trawny, S. Roumeliotis, and G. Giannakis, "Cooperative multi-robot localization under communication constraints," in *IEEE International Conference on Robotics and Automation, 2009. ICRA '09*, pp. 4394 –4400, May 2009. 81

[292] M. E. Liggins, C. Y. Chong, I. Kadar, M. G. Alford, V. Vannicola, S. Thomopoulos, and others, "Distributed fusion architectures and algorithms for target tracking," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 95–107, 1997. 81, 89

[293] A. Makarenko and H. Durrant-Whyte, "Decentralized data fusion and control in active sensor networks," in *Proceedings of the Seventh International Conference on Information Fusion*, vol. 1, pp. 479–486, 2004. 81, 87, 89, 106

[294] K. Y. K. Leung, T. D. Barfoot, and H. H. T. Liu, "Decentralized Localization of Sparsely-Communicating Robot Networks: A Centralized-Equivalent Approach," *IEEE Transactions on Robotics*, vol. 26, pp. 62–77, Feb. 2010. 81, 91, 93

[295] A. Makarenko, S. Williams, and H. Durrant-Whyte, "Decentralized certainty grid maps," in *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings*, vol. 4, pp. 3258–3263 vol.3, Oct. 2003. 81, 89

[296] A. Makarenko, A. Brooks, T. Kaupp, H. Durrant-Whyte, and F. Dellaert, "Decentralised data fusion: A graphical model approach," in *Information Fusion, 2009. FUSION'09. 12th International Conference on*, pp. 545–554, IEEE, 2009. 81, 88, 89

[297] A. Cunningham, M. Paluri, and F. Dellaert, "DDF-SAM: Fully distributed SLAM using Constrained Factor Graphs," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3025 –3030, Oct. 2010. 81, 91, 93

[298] K. Y. K. Leung, T. D. Barfoot, and H. H. Liu, "Distributed and decentralized cooperative simultaneous localization and mapping for dynamic and sparse robot networks," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 3841–3847, IEEE, 2011. 81, 91, 93

[299] P. Agarwal, *Robust Graph-Based Localization and Mapping*. PhD thesis, University of Freiburg, 2015. 81, 153, 160, 161, 214

[300] E. Olson and P. Agarwal, "Inference on networks of mixtures for robust robot mapping," in *Proceedings of Robotics: Science and Systems (RSS)*, (Sydney, Australia), July 2012. 81

[301] P. Agarwal, G. D. Tipaldi, L. Spinello, C. Stachniss, and W. Burgard, "Robust Map Optimization using Dynamic Covariance Scaling," in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2013. 81, 153, 160

[302] J. Wang and E. Olson, "Robust Pose Graph Optimization Using Stochastic Gradient Descent," 2014. 81, 200

[303] N. Karlsson, E. Di Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. E. Munich, "The vSLAM algorithm for robust localization and mapping," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 24–29, IEEE, 2005. 83

[304] D. Moore, A. S. Huang, M. Walter, E. Olson, L. Fletcher, J. Leonard, and S. Teller, "Simultaneous Local and Global State Estimation for Robotic Navigation," in *IEEE International Conference on Robotics and Automation*, 2009. 86

[305] L. A. Andersson and J. Nygards, "C-SAM: Multi-robot SLAM using square root information smoothing," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 2798–2805, IEEE, 2008. 88, 90, 93

[306] K. Singh and K. Fujimura, "Map making by cooperating mobile robots," in *, 1993 IEEE International Conference on Robotics and Automation, 1993. Proceedings*, pp. 254–259 vol.2, May 1993. 88

[307] D. T. Cole, P. Thompson, A. H. Göktoğan, and S. Sukkarieh, "System Development and Demonstration of a Cooperative UAV Team for Mapping and Tracking," *The International Journal of Robotics Research*, vol. 29, pp. 1371–1399, Sept. 2010. 89, 92, 93

[308] K. Konolige, J. Gutmann, and B. Limketkai, "Distributed map-making," in *Workshop on Reasoning with Uncertainty in Robotics, Int. Joint Conf. on Artificial Intelligence, Acapulco, Mexico*, 2003. 89

[309] H. J. Chang, C. G. Lee, Y. C. Hu, and Y.-H. Lu, "Multi-robot SLAM with topological/metric maps," in *IROS*, pp. 1467–1472, 2007. 90

[310] M. Pfingsthorn, B. Slamet, and A. Visser, "A Scalable Hybrid Multi-robot SLAM Method for Highly Detailed Maps," in *RoboCup 2007: Robot Soccer World Cup XI* (U. Visser, F. Ribeiro, T. Ohashi, and F. Dellaert, eds.), vol. 5001 of *Lecture Notes in Computer Science*, pp. 457–464, Springer Berlin / Heidelberg, 2008. 90

[311] D. D. Lee, "UPenn Multi-Robot Unmanned Vehicle System (MAGIC)," tech. rep., DTIC Document, 2014. 90, 91, 92, 96, 211

[312] A. Cunningham, K. Wurm, W. Burgard, and F. Dellaert, "Fully distributed scalable smoothing and mapping with robust multi-robot data association," in *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1093 –1100, May 2012. 91

[313] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," 1981. 92, 121

[314] A. Cunningham, V. Indelman, and F. Dellaert, "DDF-SAM 2.0: Consistent distributed smoothing and mapping," in *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5220–5227, May 2013. 92

[315] A. G. Cunningham, *Scalable online decentralized smoothing and mapping.* PhD thesis, 2014. 92, 93

[316] J. Guivant, S. Cossell, M. Whitty, and J. Katupitiya, "Internet-based operation of autonomous robots: The role of data replication, compression, bandwidth allocation and visualization," *Journal of Field Robotics*, vol. 29, no. 5, pp. 793–818, 2012. 92

[317] F. Dellaert, A. Fathi, A. Cunningham, B. Paluri, and K. Ni, "Local Exponential Maps: Towards Massively Distributed Multi-robot Mapping," tech. rep., Georgia Institute of Technology, 2010. 92

[318] L. Paz, P. Piniés, J. Tardós, and J. Neira, "Large-Scale 6-DOF SLAM With Stereo-in-Hand," *IEEE Transactions on Robotics*, vol. 24, pp. 946–957, Oct. 2008. 95

[319] G. Sibley, C. Mei, I. Reid, and P. Newman, "Vast-scale Outdoor Navigation Using Adaptive Relative Bundle Adjustment," *The International Journal of Robotics Research*, vol. 29, pp. 958–980, July 2010. 95, 220

[320] R. Reid, A. Cann, C. Meiklejohn, L. Poli, A. Boeing, and T. Bräunl, "Cooperative Multi-Robot Navigation, Exploration, Mapping and Object Detection with ROS," in *2013 IEEE Intelligent Vehicles (IV)*, IEEE, June 2013. 95, 222, 226

[321] G. Pardo-Castellote, "OMG data-distribution service: Architectural overview," in *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, pp. 200–206, IEEE, 2003. 102, 226, 228

[322] C. F. F. Karney, "Transverse Mercator with an accuracy of a few nanometers," *Journal of Geodesy*, vol. 85, pp. 475–485, Feb. 2011. 106

[323] H. Moritz, "Geodetic reference system 1980," *Bulletin géodésique*, vol. 54, pp. 395–405, Sept. 1980. 106

[324] P. J. Leach, M. Mealling, and R. Salz, "A Universally Unique Identifier (UUID) URN Namespace," 2005. 107

[325] J. N. Gray, "Notes on data base operating systems," in *Operating Systems* (R. Bayer, R. M. Graham, and G. Seegmüller, eds.), no. 60 in Lecture Notes in Computer Science, pp. 393–481, Springer Berlin Heidelberg, 1978. 116

[326] A. Censi, "An accurate closed-form estimate of ICP's covariance," in *2007 IEEE International Conference on Robotics and Automation*, pp. 3167–3172, Apr. 2007. 121, 145

[327] J. M. Coughlan and A. L. Yuille, "Manhattan world: Orientation and outlier detection by bayesian inference," *Neural Computation*, vol. 15, no. 5, 2003. 121

[328] R. Morton and E. Olson, "Robust sensor characterization via max-mixture models: GPS sensors," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 528–533, IEEE, 2013. 133, 239

[329] D. Rodriguez-Losada, P. San Segundo, M. Hernando, P. de la Puente, and A. Valero-Gomez, "GPU-Mapping: Robotic Map Building with Graphical Multiprocessors," *Robotics & Automation Magazine, IEEE*, vol. 20, no. 2, pp. 40–51, 2013. 136, 137, 139, 146

[330] J. Kessenich, "The OpenGL® Shading Language v1.30," tech. rep., 2009. 137

[331] J. Stone, D. Gohara, and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in Science Engineering*, vol. 12, pp. 66–73, May 2010. 137

[332] M. Yguel, O. Aycard, and C. Laugier, "Efficient GPU-based construction of occupancy grids using several laser range-finders," *International Journal of Vehicle Autonomous Systems*, vol. 6, pp. 48–83, Jan. 2008. 139

[333] F. Homm, N. Kaempchen, J. Ota, and D. Burschka, "Efficient occupancy grid computation on the GPU with lidar and radar for road boundary detection," in *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pp. 1006–1013, IEEE, 2010. 139

[334] A. Birk and S. Carpin, "Merging occupancy grid maps from multiple robots," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1384–1397, 2006. 139

[335] H. Li, M. Tsukada, F. Nashashibi, and M. Parent, "Multivehicle Cooperative Local Mapping: A Methodology Based on Occupancy Grid Map Merging," *IEEE Transactions on Intelligent Transportation Systems*, vol. Early Access Online, 2014. 139

[336] R. Wright, B. Lipchak, and N. Haemel, *OpenGL SuperBible: Comprehensive Tutorial and Reference*. Pearson Education, June 2007. 141, 142, 143, 148

[337] S. Saeedi, L. Paull, M. Trentini, M. Seto, and H. Li, "Map merging for multiple robots using Hough peak matching," *Robotics and Autonomous Systems*, 2014. 145

[338] N. Sunderhauf and P. Protzel, "Towards a robust back-end for pose graph SLAM," in *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1254 –1261, May 2012. 153

[339] Y. Latif, C. Cadena, and J. Neira, "Detecting the correct graph structure in pose graph SLAM," in *ICRA Workshop on Robust and Multimodal Inference in Factor Graphs*, 2013. 153

[340] Y. Latif, C. Cadena, and J. Neira, "Robust loop closing over time for pose graph SLAM," *The International Journal of Robotics Research*, Oct. 2013. 159

[341] Z. Zhang, "Parameter estimation techniques: A tutorial with application to conic fitting," *Image and vision Computing*, vol. 15, no. 1, pp. 59–76, 1997. 159, 160, 161, 168

[342] N. Sünderhauf and P. Protzel, "Switchable Constraints for Robust Pose Graph SLAM," 2012. 159, 160

[343] M. Pfingsthorn, A. Birk, F. Ferreira, G. Veruggio, M. Caccia, and G. Bruzzone, "Large-scale image mosaicking using multimodal hyperedge constraints from multiple registration methods within the Generalized Graph SLAM framework," in *Intelligent Robots and Systems (IROS)*, pp. 4564–4570, IEEE, 2014. 160, 214

[344] E. Olson, "M3rsm: Many-to-Many Multi-Resolution Scan Matching," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, June 2015. 161, 212, 213

[345] T. Bailey, S. Julier, and G. Agamennoni, "On conservative fusion of information with unknown non-Gaussian dependence," in *2012 15th International Conference on Information Fusion (FUSION)*, pp. 1876–1883, July 2012. 163

[346] C. M. Humphrey, C. Henk, G. Sewell, B. W. Williams, and J. A. Adams, "Assessing the scalability of a multiple robot interface," in *Human-Robot Interaction (HRI), 2007 2nd ACM/IEEE International Conference on*, pp. 239–246, IEEE, 2007. 210

[347] C. H. Tong, S. Anderson, H. Dong, and T. D. Barfoot, "Pose Interpolation for Laser-based Visual Odometry: Pose Interpolation for Laser-based Visual Odometry," *Journal of Field Robotics*, vol. 31, pp. 787–813, Sept. 2014. 220

[348] C. Forster, M. Pizzoli, and D. Scaramuzza, "Air-ground localization and map augmentation using monocular dense reconstruction," in *Intelligent Robots and Systems (IROS)*, pp. 3971–3978, IEEE, 2013. 221

[349] S. Lynen, T. Sattler, M. Bosse, J. Hesch, M. Pollefeys, and R. Siegwart, "Get Out of My Lab: Large-scale, Real-Time Visual-Inertial Localization," in *RSS*, 2015. 221

[350] R. Triebel, P. Pfaff, and W. Burgard, "Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing," pp. 2276–2282, Oct. 2006. 221

[351] K. Konolige, E. Marder-Eppstein, and B. Marthi, "Navigation in Hybrid Metric–Topological Maps," in *International Conference on Robotics and Automation*, 2011. 221

[352] R. Valencia, M. Morta, J. Andrade-Cetto, and J. M. Porta, "Planning reliable paths with pose SLAM," *Robotics, IEEE Transactions on*, vol. 29, no. 4, pp. 1050–1059, 2013. 221

[353] N. Carlevaris-Bianco, M. Kaess, and R. Eustice, "Generic Node Removal for Factor-Graph SLAM," *IEEE Transactions on Robotics*, pp. 1–15, 2014. 222

[354] W. Woodall, "ROS on DDS," 2015. 222, 226

[355] Bank of America Merrill Lynch, "Robot Revolution – Global Robot & AI Primer," tech. rep., 2015. 223

[356] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized link state routing protocol for ad hoc networks," in *Multi Topic*

Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century. Proceedings. IEEE International, pp. 62–68, 2001. 227

[357] A. Huang, E. Olson, and D. Moore, "LCM: Lightweight Communications and Marshalling," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2010. 230

[358] K. Nagatani, A. Kushleyev, and D. D. Lee, "Sensor Information Processing in Robot Competitions and Real World Robotic Challenges," *Advanced Robotics*, vol. 26, no. 14, pp. 1539–1554, 2012. 234