Gonçalo dos Santos Martins

A Cooperative SLAM Framework with Efficient Information Sharing over Mobile Ad Hoc Networks

July 2014

· U C ·

Universidade de Coimbra

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

A Cooperative SLAM Framework with Efficient Information Sharing over Mobile Ad Hoc Networks

Gonçalo dos Santos Martins

Coimbra, July 2014

A Cooperative SLAM Framework with Efficient Information Sharing over Mobile Ad Hoc Networks

Supervisor:

Prof. Doutor Rui P. Rocha

Co-Supervisor:

Doutor David B. S. Portugal

Jury:

Prof. Doutor Paulo Jorge Carvalho Menezes

Prof. Doutor João Filipe de Castro Cardoso Ferreira

Prof. Doutor Rui Paulo Pinto da Rocha

Dissertation submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering.

Coimbra, July 2014

Acknowledgements

When one embarks on a journey such as this one, it is tempting to believe that success depends solely on one's own efforts. It is tempting to believe that scholarly articles, on-line sources of information and code-related documentation are enough to form a solid basis on which to build one's work.

However, for as tempting as the notion may be, it is simply untrue. There is no substitute for people, for colleagues and superiors, for people who depend on you and on who you depend. There is no substitute for informal discussion with other researchers. There is no substitute for the ability to "bounce ideas off" your colleagues, and there is no substitute for their feedback. Regardless of how complete our documents are, how well sourced and written are our on-line encyclopedias, and how thorough is our documentation, there is no substitute for human interaction.

In this short text, I would like to acknowledge the various people who have helped me, in one way or another, whether they may have realized it or not. I would also like to apologize in advance to anyone I may have forgotten to include in this musing.

First and foremost, I thank my family, who have unconditionally supported me throughout all my endeavors, big or small, and I thank my girlfriend, Filipa, whose love, care and support were crucial for the completion of this work.

I thank my supervisors, Prof. Rui Rocha and Dr. David Portugal, for always keeping me on track, for their encouragement, and for their guidance, for continuously making me rethink my work, slowly refining it through their interventions, and for giving me the opportunity to undertake this arduous but enlightening mission.

I also give my thanks to my colleagues at the Mobile Robotics Lab. To Pablo Lanillos, for his immensely important feedback on some of my ideas, for helping me conceptually develop some of them, for serving as a "test subject" for much of my figures and illustrations, for indulging me on my lectures on personal and work organization, and, most of all, for his friendship. To Pedro Trindade, Luís Santos and Ricardo Martins, for welcoming me with

open arms to the Lab when I first came in, for their continued support in all smaller tasks, and for our engaging lunch-time conversations. To João Santos, for his help in the embryonic phase of this work. To Rohit Chandra, the latest addition to the team, for making me doubt my work at a fundamental level, causing me to reflect deeply upon it, for enduring my lectures on Software Engineering, Version Control and Portuguese Culture, and also for his friendship. To Francisco Sales, for his continued companionship during the development of this work. To Beatriz Oliveira, the youngest member of the team, for humbling me by reminding me that regardless of our achievements, we were all once young.

Resumo

À medida que o custo das plataformas robóticas mais simples cai, cresce o interesse em soluções que explorem a cooperação entre múltiplos robôs móveis. As técnicas cooperativas apresentam um sem-fim de vantagens sobre as suas equivalentes individuais no que toca à solução de problemas facilmente divisíveis.

A Localização e Mapeamento Simultâneos (do Inglês SLAM - Simultaneous Localization and Mapping) é um dos problemas mais estudados na Robótica recente, e trata do problema de, sem conhecimento *a priori* de um dado ambiente, dotar uma plataforma robótica móvel da capacidade de construir uma representação desse ambiente e de, simultaneamente, se localizar nela.

A exploração de ambientes desconhecidos apresenta-se-nos como um problema particularmente e intuitivamente divisível, pelo menos no domínio humano: resume-se a seccionar a equipa em grupos menores, até individuais, que exploram zonas distintas do ambiente, sendo que mais tarde se reúnem para elaborar uma visão conjunta do espaço.

No entanto, no domínio robótico, esta abordagem apresenta uma série de novos desafios, como a comunicação, o sincronismo da informação contida em cada máquina distinta e a fusão da informação recolhida pelos vários membros da equipa.

Este trabalho foca-se na solução destes novos problemas, conhecidos no seu conjunto como SLAM Multi-Robô. Para tal, foi construída uma solução de software que procura dotar qualquer sistema capaz de correr uma solução SLAM da capacidade de comunicar eficientemente com os seus vizinhos e de construir uma representação global do ambiente com base na sua própria informação e na informação recebida dos seus vizinhos.

A solução foi validada através de testes com dados reais, tanto offline como online, ou seja, tanto com dados previamente gravados como com informação recolhida no momento, e mostrou um desempenho adequado, tanto em termos de escalabilidade como de eficiência na comunicação.

Keywords: SLAM, SLAM Multi-Robô, ROS, MANET, Comunicação Eficiente

Abstract

As the cost of simple robotic platforms plummets, interest in solutions that explore the cooperation between robotic agents grows. Cooperative techniques enjoy a multitude of advantages over their individual counterparts, namely concerning the solution of easily divisible problems.

Simultaneous Localization and Mapping (SLAM) is one of the most researched topics in Robotics. It deals with the problem of endowing a mobile robotic platform with the ability to, with no *a priori* knowledge of the environment, build a representation of its surroundings and, simultaneously, localize itself in it.

The exploration of unknown environments seems to be a particularly and intuitively divisible problem, at least at a human level: we simply divide the team into smaller groups, or even into individuals, which explore different areas in the environment. After exploring, the team reassembles and builds a joint representation of the region. However, this approach gives rise to a new set of problems, such as communication, synchronization and information fusion.

This work focuses in the solution of these problems, known collectively as Multi-Robot SLAM. We propose a novel software solution that seeks to provide any system capable of performing single robot SLAM with the ability to efficiently communicate with its teammates and to build a global representation of the environment based on the information it exchanges with its peers.

This solution was validated through experiments conducted over real-world data, both on- and off-line, *i.e.*, both using prerecorded and data gathered during the experiment. It has shown acceptable performance, both in scalability and communication efficiency.

Keywords: SLAM, Multi-Robot SLAM, ROS, MANET, Efficient Communication

"According to quantum physics, no matter how much information we obtain or how powerful our computing abilities, the outcomes of physical processes cannot be predicted with certainty because they are not determined with certainty. Instead, given the initial state of a system, nature determines its future state through a process that is fundamentally uncertain."

— Stephen Hawking, *The Grand Design*

Contents

List of Acronyms

API	Application Programming Interface
CHOPIN	Cooperation between Human and rObotic teams in catastroPhic INcidents
EKF	Extended Kalman Filter
GPS	Global Positioning System
g^{2}o	General Graph Optimization
KF	Kalman Filter
LIDAR	LIght Detection And Ranging
MANET	Mobile Ad hoc NETwork
MRSLAM	Multi-Robot SLAM
OLSR	Optimized Link State Routing protocol
PF	Particle Filter
PNG	Portable Network Graphics
RANSAC	RANdom SAmple Consensus
RBPF	Rao-Blackwellized Particle Filter
ROS	Robot Operating System
RSSI	Received Signal Strength Indicator
SLAM	Simultaneous Localization And Mapping
SPA	Sparse Pose Ajustment

TORO Tree-based netwORk Optimizer

List of Figures

List of Tables

1 Introduction

Humanity has always felt a need to learn and record the geometry and features of the world around it, either by engraving crude maps into rocks, by crafting elaborate and exquisitely detailed representations of the known world on canvas, or, more recently, by using satellite imagery and computer graphics to create extremely accurate interactive representations of our planet. This behavior follows from our tendency to explore, to be curious about our surroundings, and also from our desire to know where we are, and how the various parts of the world we sense interlock with each other. Maps allow us to have a global view of our environment, to navigate within it; to take advantage of the fact that someone, or something, else has already explored and mapped our surroundings.

Mapping can be a dangerous task. It may require the mapper to spend long periods of time in hazardous conditions, *e.g.* underwater, exposed to extreme temperatures, radiation or other deadly environments. It may even be impossible for a human to enter an enclosed space that needs mapping. Furthermore, while our eyes excel at detecting potential predators and nutrition, we cannot extract precise measurements solely from our visual observations of a location, which makes us generally less than useful at creating accurate maps of geometrically complex locations. Robots, on the other hand, can be made to resist some of the harshest conditions known to man, can be extremely precise in their measurements, can be made to be very small and, in dangerous situations, are less valuable than a human life; robots are expendable and replaceable, whereas humans are most certainly not.

Thus, robots appear to be a perfect choice for mapmakers. In fact, the robotic mapping problem has been an active field of research for the last few decades, and several appropriate approaches exist already. When mapping indoor locations, in the absence of global references such as GPS (Global Positioning System), the mapping robot has no way of knowing where it is when it starts recording data. This fact constitutes a chicken-and-egg problem: we need a map to locate ourselves, and we need to locate ourselves in order to be able to create a map. These operations must be performed simultaneously. The problem of simultaneously

creating a map from range sensor measurements and locating a robot in that same map is usually known as SLAM, Simultaneous Localization And Mapping.

Solving this problem for the 2D case, using a simple robot with odometric and range scanning capabilities, may seem straightforward: we could theoretically determine the position of the robot using the odometry and locate the confines of the environment using the range scans, thus gathering sufficient data to create an accurate representation of the environment. However, realistically, all measurements are significantly affected by noise, and while this simple solution may at first seem viable, its implementation would lead to misaligned data due to cumulative errors and, subsequently, erroneous maps. Noise must be taken into account when we try to solve this problem, which leads to multiple complex solutions.

In the presence of a large, intricate environment, such as an abandoned factory, a cave, or a jungle, humans tend to break up into small groups to try and maximize the gain of new information per unit of time on the structure and accessibility of this new environment. Various groups can later meet, or *rendezvous*, communicate to each other what they have learned from their experience and, together, create an approximate representation of the location. This behavior is extremely beneficial to the group, and is an efficient way of gathering information, as opposed to having a single person exploring, or the entire group together. We can easily imagine, then, that a team of robots would have tremendous advantages over a single robot at mapping certain locations, particularly vast ones. In fact, this is also an open field of research, and this problem is usually known as Multi-Robot SLAM. As with SLAM, this is a deeper challenge than meets the eye. For example, unlike single robot SLAM, Multi-Robot SLAM requires that the robots be able to communicate amongst themselves, so that they can coordinate their exploratory efforts. To achieve this goal, we will be using the robots as nodes of a Mobile Ad-Hoc, infrastructure-less network (MANET).

While exchanging information, efficiency is the key to scalability. When only two humans trade impressions on the layout of an area, they can be as expressive as they like. However, if a third one is added, they must coordinate better so that each has the opportunity to talk. As the group grows, a new protocol for the exchange of information must be developed, perhaps even involving a new way to encode the information that needs to be exchanged. Similarly, in order to build a scalable multi-robot system, the agents need to be able to communicate in an efficient way, so that the addition of new members to the team does not cause failures in the network connecting them, *i.e.* that no information is lost or corrupted in transit. We have also tackled this issue in this work, and propose a solution to the issue

of efficient communication in robotic teams in Chapter 3.

The main goal of this work is to develop, test and refine a new Multi-Robot SLAM technique tailored to the needs of the CHOPIN (**C**ooperation between **H**uman and r**O**botic teams in catastro**P**hic **IN**cidents) Project [32], in which the author was integrated.

1.1 Our Challenge

Given our knowledge of previous work in SLAM and Multi-Robot SLAM, our prime objective is to develop a Multi-Robot SLAM approach in the context of the CHOPIN project .

Our approach should be distributed, *i.e.* must not depend on a centralized processing unit. In this sense, we will make use of preexisting software tools to enable independent robots to communicate through a wireless network.

Our technique must also be robust to failures in communication, *i.e.* the mission should not be compromised by the corruption or loss of a message.

Our approach should be scalable, in the sense that adding robots to the mission should not compromise their performance, within the limits of reason. This should be achieved both by making robots communicate as efficiently as possible, and by carefully planning the execution of our software.

Finally, our approach should be SLAM-technique-agnostic, *i.e.* able to work regardless of the SLAM technique employed.

1.2 Document Overview

This Dissertation is organized as follows: we start by taking a short journey into the world of SLAM, in Chapter 2; in Chapter 3, we discuss the issue of efficient communication in multi-robot missions, and propose an adequate solution; in Chapter 4 we present and describe the complete system we have developed for performing Multi-Robot SLAM, which we then validate experimentally in Chapter 5; finally, in Chapter 6 we reflect upon the advantages and handicaps of our system, as well as on the completion of our objectives and on possible future work.

2 A Foray Into SLAM and Multi-Robot Systems

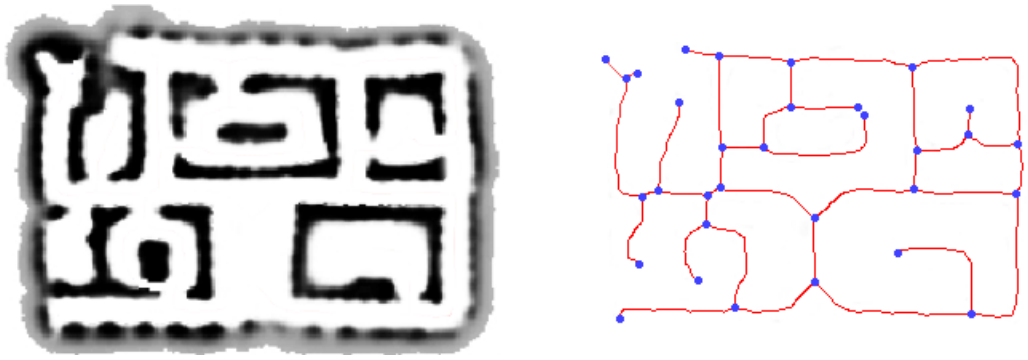
2.1 SLAM

2.1.1 What is SLAM?

SLAM stands for Simultaneous Localization and Mapping. Essentially, this means that a robot performing SLAM is tasked with both creating a map of the environment and localizing itself in it.

In its simplest form, performing SLAM consists of having a system composed of a mobile robotic platform and a processing unit (which may or may not be embedded in the robot itself) exploring a given region and, employing one of several available algorithms, using the data it gathers to estimate the path it takes while exploring and building a representation of that same region: a map.

A map is a representation of the robot's surroundings, and obtaining it is one of the main goals of SLAM. Maps themselves usually fit into one of two categories: metric or topological [38]. Metric maps offer a detailed description of the environment, based on an absolute reference frame. Occupancy grids [10], a type of metric map, are one of the most popular type of map. These are constituted by a matrix of values, each corresponding to a location in space. Higher values indicate a higher probability of that space being occupied, and, analogously, lower values represent a higher probability of the corresponding space being empty. When represented visually, each cell's color indicates the probability that this cell is occupied: white cells correspond to open, traversable space; darker cells indicate the presence of obstacles. Metric maps have the disadvantage of being space-consuming: building a detailed map requires a high resolution grid, which implies a large number of cells and, therefore, a large amount of memory dedicated to maintaining this map.



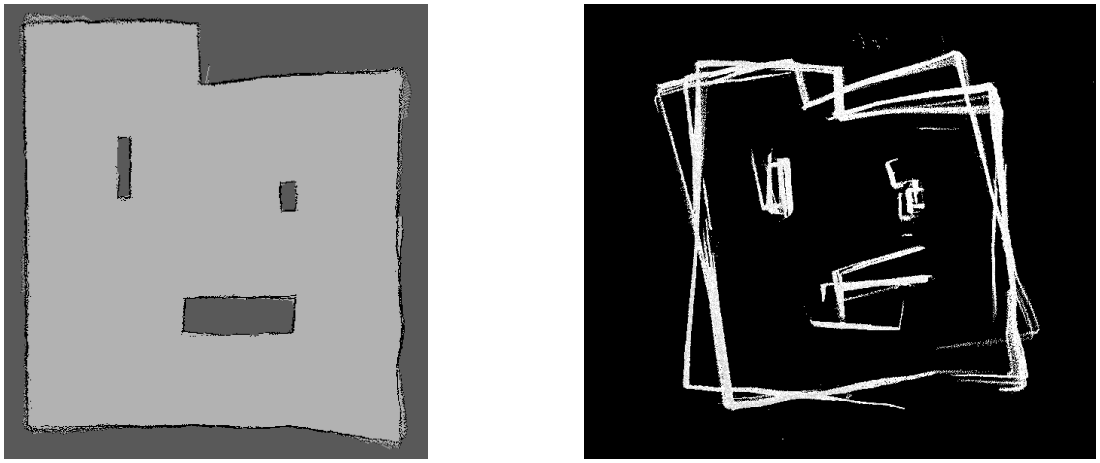
(a) A metric map. (b) A topological map.

Figure 2.1: A comparison of metric and topological maps.

Topological maps represent the environment by means of a graph; they are not scale representations of reality, but are instead a way of representing various places or landmarks and their connectivity. In this context, a node corresponds to a specific landmark or location, and the existence of an edge connecting two nodes indicates the possibility of navigating between them. Fig. 2.1 illustrates the differences between metric and topological maps.

Using its sensors, which usually include range and odometric sensors, the robot gathers data as it explores the world. Here lies the very first issue that the algorithm implemented in the robot must deal with: all data gathered through sensors is plagued by noise and uncertainty. This has led to a strong prevalence of probabilistic solutions to the SLAM problem over mathematically simpler approaches [37] [11]. Odometry, in particular, produces errors that accumulate over time, as shown in Fig. 2.2. When the robot is exploring a previously mapped environment, these errors can be bounded by estimating the robot's location relative to known landmarks. With no assistance from other sensors, however, odometric errors can grow arbitrarily large [13].

As data is received, the robot must be able to relate it to the previous data it has been gathering, *i.e.*, the data must be *aligned*. This is usually known as the *correspondence problem*, and solving it is usually accomplished by means of *feature matching*: the robot extracts a number of features from every scan and tries to match them with features extracted from previous scans. This process is of the utmost importance during *loop closure*. Loop closure consists of the algorithm's ability to recognize a loop in the environment, *i.e.* to recognize the fact that the robot has already visited a certain location, albeit on a different trajectory, and to take that fact into account in its calculations. The inability to recognize a loop in the environment may lead to an erroneous, unintelligible map.



(a) (b)

Figure 2.2: An illustration of how important data alignment is. The map on the right was created using raw odometry data for localization, while the one on the left was rendered after data was aligned using sensor information and loop closure techniques.

2.1.2 Classical Solutions

Filtering Techniques

Most classical solutions to the SLAM problem are based on Bayes Filters [37] [11]. The basic underlying principle of these solutions is the Bayes Rule:

$$p(x \mid d) = \eta p(d \mid x)p(x). \tag{2.1}$$

The first term, $p(d \mid x)$, is called the *generative* or *likelihood* model: it describes the probability of observing d given our hypothesis x. The second term, $p(x)$, is know as *prior*, and represents our willingness to admit *a priori* that x is indeed the correct assumption. Lastly, η is a normalizer that ensures that $p(x \mid d)$ is a valid probability distribution [37] [11].

In the context of mobile robotics, we usually deal with two types of data: sensor measurements, henceforth denoted as z, and controls, which will be represented by u. Sensor measurements are the data the robot gathers: camera images, range measurements, *etc.* Controls are the signals transmitted to the robot's actuators which indicate how it is supposed to move. They represent the way the robot is *intended* to move, not necessarily the way it actually moves, thus it is common to replace controls with odometric measurements which, albeit prone to errors, are more accurate than controls at representing robotic motion [37].

The Bayes Filter is an extension to Bayes Rule to integrate temporal data, and is usually formulated as follows:

$$p(s_t, m \mid z^t, u^t) = \eta p(z_t \mid s_t, m) \int p(s_t \mid u_t, s_{t-1}) p(s_{t-1}, m \mid z^{t-1}, u^{t-1}) ds_{t-1}, \tag{2.2}$$

where s_t is the robot's estimated pose at time t, and m represents the map. Also, $s^t = \{s_1, s_2, s_3, ...\}$, *i.e.* the superscripted symbol represents all the values of that quantity up to time t.

To arrive at this formulation, we assume the map, and therefore the world we are mapping, to be *static*. This is an extremely useful assumption in practical terms; by rendering the map constant, it removes the need to integrate over it [37] [11].

The Kalman Filter (KF), presented for the first time in [18], is a Bayes Filter that uses Gaussian distributions, which can be defined by a small number of parameters to represent the posterior $p(s_t, m \mid z^t, u^t)$. In this context, *maps* are usually the Cartesian coordinates of sets of features, or *landmarks*, whereas *pose* represents the robot's position. In the 2D case, a set of three variables are used: x and y, Cartesian coordinates in the plane the robot explores, and θ, the heading direction. These are usually condensed into a vector $\bar{x} = [x, y, \theta]$.

This approach is set on some basic assumptions: each map and pose must depend linearly on the previous map and pose. Regarding the map, this is true since we assume it to be static. However, a given pose may not linearly depend on the previous pose, since motion is usually governed by nonlinear trigonometric functions [37]. To solve this problem, the nonlinear function is approximated by a truncated Taylor Series. Measured motion is also subdivided into smaller steps, to account for nonlinearity, resulting in a better estimate of real motion. The approach based on these modifications to the Kalman Filter is known as the *Extended Kalman Filter* (EKF).

The Extended Kalman Filter, while a step forward from the standard Kalman Filter, is still unable to deal with ambiguous features, as noted in [37]. A sparse set of distinctive landmarks, either by sensor data or location, is required in order to achieve reliable identification, which is of extreme importance, since errors in the identification of environment features can, and usually do, lead to a complete failure of the mapping process.

Particle Filters (PFs) are recursive Bayes Filters that estimate the posterior of poses conditioned by gathered data representing it by a set of weighted samples, also known as *particles*, instead of a function or a set of parameters [40]. Each particle contains, effectively, a possible solution to the problem at hand, as illustrated for simple localization in Fig. 2.3. A higher number of particles leads to a better, more accurate estimated map. However, computational cost grows with the number of particles used, which means that a trade-off

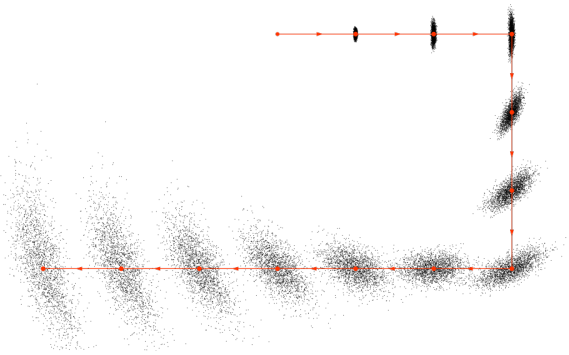


Figure 2.3: An illustration of how the particle distribution evolves over time when using a robot that only measures odometry. As the odometric error compounds, uncertainty about the robot's position increases; however, the densest regions in each group of particles indicate a roughly correct estimate. The solid line displays the motion of the robot, and the samples (dots) represent the robot's belief at different points in time.[1].

between accuracy and computational cost must be found [40]. These approaches assume that each state is dependent only on the previous state, and not on the sequence of states that preceded it, an assumption often referred to as the *Markov assumption.*

Unlike the Kalman and Extended Kalman Filters, Particle Filters are able to deal with non-Gaussian noise. In fact, they can take into account almost arbitrary distributions for not only noise, but also sensor characteristics and motion dynamics, and thus seem like a good solution to one of the main problems observed in Kalman Filters. However, Particle Filtering tends to be more computationally demanding, although it does tend to focus resources in relevant areas. The algorithm is also adaptable to computers with varying resources, by controlling the number of active particles [40].

Graph-Based SLAM

Graph-Based SLAM was firstly introduced in 1997, by F. Lu and E. Milios in [24]. Unlike filtering techniques, in which data is discarded after being processed and incorporated into the filter's state, Graph-Based SLAM techniques keep all gathered data, and a full notion of uncertainty, in the form of a graph. Keeping a complete history of past poses is an important characteristic of this technique. Past poses are used to define local frames of reference, which means that maintaining a complete history of past poses equates to maintaining the structure

[1]Image created by Daniel Lu and used under the terms of the Creative Commons License. More information, including the code used to generate the image, can be found at`https://commons.wikimedia.org/wiki/File:Particle2dmotion.svg`

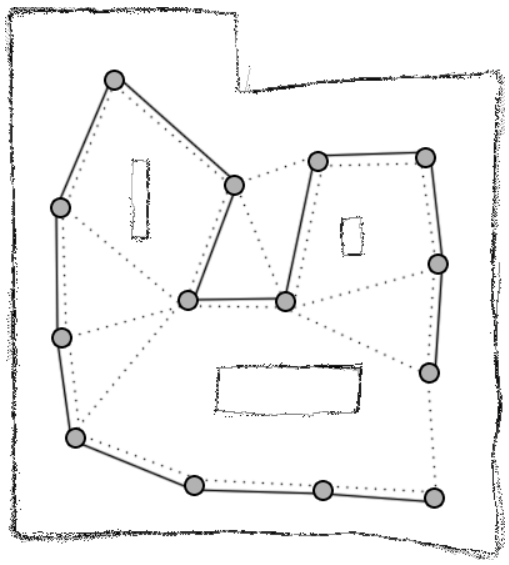


Figure 2.4: A graphical representation of the constraint network used by graph-based SLAM techniques. Please note how links are distributed: there can be various links from each node, and multiple links connecting any two nodes. Strong links are represented by dotted lines while weak links are portrayed as continuous lines.

of the environment. Keeping track of local frames of reference is also an important tool when it comes to rendering the range scans as a map.

While filtering techniques are usually additive, in the sense that they add data to the map as the robot explores its surroundings, graph-based techniques perform a *global* optimization over all poses, which means that all data must be processed at once, potentially at every step. When this technique was first introduced, it was regarded as a purely *offline* technique, *i.e.*, the map was generated after all data was gathered, not while the robot was exploring its environment. Growth in computation power, and techniques such as variable elimination [39] now allow us to use this technique in real time.

This approach is known as *graph*-based, a name owed to how data is usually represented in this context and to the optimization problem underlying this technique, not to be confused with how the map is represented. As illustrated in Fig. 2.4, poses are represented by *nodes* and links between poses by lines connecting these nodes, or *edges*. These usually form a *non-complete* graph, *i.e.*, not all nodes are directly connected to one another. There are also commonly two types of edges: those representing *strong*, and those representing *weak* links. Once the graph is built and constraints are derived, a *graph optimization* process takes place, which finds numerical solutions to all the variables involved in the estimation problem, namely all the poses at which the robot performed range scans, a solution also known as a *pose-graph*. The poses returned by the optimization process are usually relative

to a common reference frame, *e.g.* in [24], poses are determined using the first pose as global reference.

Graph-based approaches normally assume a robot equipped with odometric sensors and a range scanner, *i.e.*, a robot capable of estimating its pose as it travels and able to take scans of distances to objects around itself. Fig. 2.4 also presents an illustration of how range scans work. Every point rendered on the image is a distance measured by the range scanner as it sweeps the environment.

As the robot progresses, it periodically takes a scan of the environment. This scan, usually, but not necessarily, performed by laser range scanners, provides an indication of its distance to all the objects that surround it. Each of these scans, along with the traveled distance the robot measured since the last scan (or the relative pose measured from the last to the current pose), constitute a node in the network. Every node introduces a new variable, the pose at which it was taken, and one or more constraints to be integrated into the graph: a *weak link*, derived from the odometric measurements, and possibly various *strong links*, which are obtained through scan matching.

Scan matching is a technique through which we can introduce new constraints into our graph. If the robot takes scans at two nearby locations, those scans will very likely have some overlap. Scan matching detects which scans overlap, and estimates the relation between poses that needs to exist in order for that specific overlap between scans to exist. There are various techniques that can be employed to perform scan matching, for example the one described in [7]. This is a fairly well-studied problem, and usually not the focus of SLAM implementations.

The resulting graph is, then, constituted by nodes, strong and weak links. As described in [24], links can be perceived as springs connecting loose points in space or, in this case, in the 2D plane. Solving this problem is the equivalent of "letting go" of the system, *i.e.*, we aim to minimize the energy contained in these springs, just like a physical system would tend to.

More recently, Graph-Based SLAM solutions began relying on graph optimization frameworks, or graph optimizers. These frameworks can be collaboratively developed, and their development can be completely independent of the development of the remaining of the SLAM solution. Graph optimizers are tools that deal only with the graph optimization, leaving the acquisition of data, scan matching, analysis of the optimized poses, map building, *etc.* to be dealt with by the application in which they are included. Modern Graph-Based SLAM solutions are, then, usually divided in two different pieces of software: the Front-End,

responsible for building the graph (including scan matching), and the Back-End, which deals only with the optimization of the graph.

Graph Optimizers receive as input all the nodes and edges of the graph [4] [3], and return the optimized poses as output. The optimization facilities provided by these tools are usually accessed via an API, which, of course, varies from tool to tool. Examples of these tools include *TORO* [4] [15], one of the earliest implementations of a graph optimizer; *SPA* [19], an improved version of the algorithm originally proposed by Lu and Milios; and *g2o* [3] [20], an open-source C++ framework for general graph optimization.

2.2 Multi-Robot SLAM

2.2.1 Multi-Robot SLAM as an Extension to SLAM

We humans understand that cooperation is a fundamental and intrinsic part of our world. We cooperate every day with other humans in order to achieve much higher goals than we could ever hope to achieve on our own. In fact, cooperation is so intertwined into our daily lives we may not even notice it. Nowadays, seldom are great achievements attained by individuals: companies use cooperating workers to gain wealth; universities use cooperating researchers to delve deeper into the unknown mysteries of the universe; construction companies combine the efforts of numberless individuals to rise higher and higher into the sky. Even menial tasks like taking public transportation or carpooling can be construed as cooperation between humans.

Multi-Robot Systems have a number of advantages over Single-Robot solutions, such as parallelism, distribution in space and time, problem decomposition, reliability, robustness and redundancy. Thus, they are greatly advantageous in the solution of monotonous, repetitive, complex, dangerous, large-scale and dividable problems [29].

Multi-Robot SLAM is, then, a natural extension to the original SLAM problem: if we are able to map a 2D environment using a single mobile robot, why not apply this powerful concept and use multiple robots, so as to achieve our goal in a faster, more reliable and robust fashion?

Despite their many advantages, the usage of multi-robot systems gives way to the rise of multiple new problems. First and foremost, coordination is fundamental and, thus, inter-robot communication becomes a capital factor in the team's performance. Furthermore, in search and rescue scenarios, the ability to communicate information to teammates is hindered

by the non-existence of a network infrastructure. In regards to SLAM, one of the biggest challenges is that of combining the information gathered by multiple robots.

2.2.2 Communicating data

Communication is an essential part of cooperation. In order to cooperate, humans have to be able to communicate their intentions and relevant data, be it verbally, textually or pictorially. Analogously, robots have a need to share information in order to perform their cooperative tasks.

In the context of the CHOPIN project, we will be using infrastructure-less networks, namely a subset of Ad Hoc connections called Mobile Ad Hoc Networks (MANETs). A MANET is a self-configuring, infrastructure-less dynamic network in which nodes, that are assumed to be mobile, also play the role of routers. MANETs offer a considerable challenge for routing algorithms, caused by the mobility of nodes. Routing algorithms, which decide how packets travel, *i.e.* which path (or route) they follow in a network in order to reach their destination, are based on the *cost* of each path, much like how a robot finds the shortest path to a given destination in a topological map. However, mobile nodes cause the links between said nodes to dynamically change: the shortest path, network-wise, is subject to frequent change. Still, for our purposes, MANETs are a convenient solution: they do not rely on an infrastructure, and nodes have the possibility to move in any direction and can be added or removed without disturbing the remaining network.

In a vast or highly occlusive environment, *e.g.* one with an intricate topology that limits the robots' wireless communication range, it is very likely that, sooner or later, our network will no longer encompass all operating robots and that they will form various sub-networks. In this case, it is very important that we be able to guarantee the consistency of the information being exchanged between robots.

Keeping track of which robot has which pieces of information can be a very important step in reducing data traffic between robots. For instance, if two robots meet a significant time after deployment, transferring the entire data they have gathered so far would probably cause a considerable delay before they could resume their tasks. However, if the robots can inform each other of how much data they need from each other, and if they have met a few times before, the total amount of data they need to transfer can be dramatically reduced. This issue is discussed in [27], where the concept of *rendezvous* is used. Briefly, there is a successful rendezvous for time T if all robots have all the relevant data up to time T. This is

an important concept, not only in cooperative robotic mapping, but in multi-robot systems in general.

2.2.3 Information Fusion

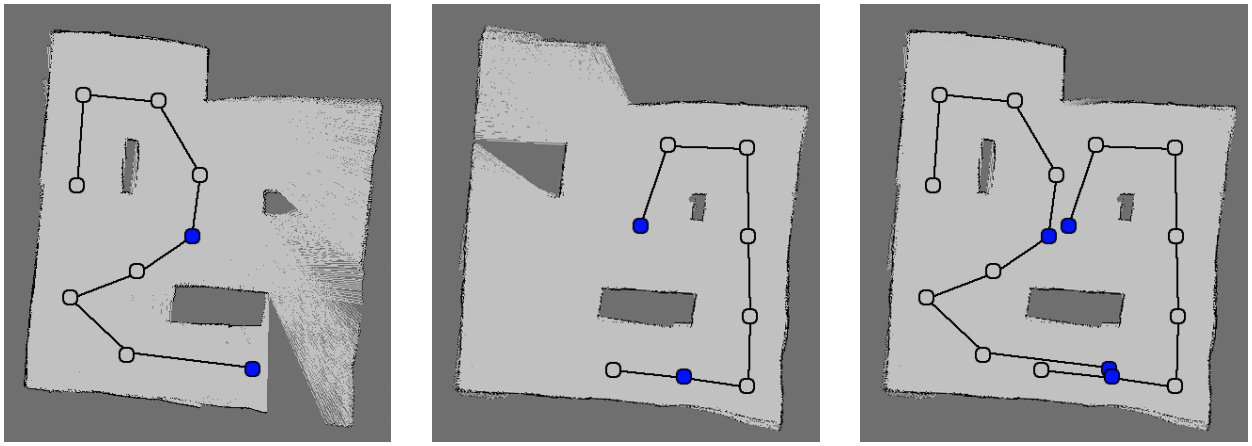
Information fusing is one of the greatest challenges related to Multi-Robot SLAM. After a successful exchange of data, robots need to combine their local information with the received one into a single, consistent representation of the environment. There are various solutions to this problem, and we shall review and discuss a few of them.

Map fusion, *i.e.* stitching together the various contributions into a combined representation, can take place on one of two levels, either by merging data such as poses, landmarks, graphs, *etc* ([1], [23], [12]), or by merging the rendered occupancy grids themselves [6].

In [1], map fusion is assisted by what is called *rendezvous measurements*. While exploring, robots often pass by each other, going in the same or different directions. When this occurs, it is possible to measure the relative pose between them using robot-to-robot methods, such as the visual detection system described in [41], in which virtual landmarks mounted on the robots are detected by cameras also mounted on the robots. These measurements are then used to estimate a transform, called the *base node*, defined by $\mathbf{b} = [x, y, \phi]^T$, which contains the rotation and translation needed to transform one robot's local coordinates into the other's. Once the base node is determined, merging local maps is a trivial matter of matching features present in both maps and building the combined map. This method suffers from the need of precision measurements of relative poses, which is not always possible, making this technique unfit for our particular case.

The method introduced in [23] approaches the problem in a different manner: robots carry a camera, which creates a stream of images as the robot explores. From these images, features can be extracted such that each location on the map can be identified in an unambiguous way. Thus, each time a robot needs to merge two maps, it can search for locations that exist in both maps, and thus extract translational information that relates them. This technique, however, requires that *model images* exist previously, taken in ideal conditions, which will be used by robots in location matching.

Fox *et al.* proposed, in [12], a complete method for Graph-Based SLAM with multiple robots, extending the classical Graph-Based SLAM paradigm. This approach merges maps on the graph-level, *i.e.*, maps are merged by combining the graph representations created by various robots, through relative pose measurement and *colocation* (a location visited by



(a) (b) (c)

Figure 2.5: An illustration of the technique used in [12]. (a) and (b) are representations of the local maps created by two robots, in which the blue nodes represent locations matched between maps. These matches are introduced into the pose graph as new constraints, allowing the data to be fused and re-optimized, resulting in a globally consistent map (c).

multiple robots, a region in space common to various graphs), as illustrated in Fig. 2.5. As robots explore the environment, Graph-Based SLAM dictates that constraints be created between each pose at which a range scan is taken. Classically, these constraints are derived from odometric measurements, scan matching and loop closure. The authors introduce a new type of constraint, which is derived by matching poses between various local maps. It is also important to note that, unlike the classical graph-based approach, this technique uses a local representation for pose relations. This way, pose relations are independent of the world coordinate system and, thus, invariant with rigid transformations (such as rotation and translation).

Once colocation has been defined, *i.e.* once we have found, on the graph, a location that both robots have visited, their graphs can be joined at that point. Then, a *zippering* process takes place, in which scans from different robots are matched in order to find additional constraints. This technique, in its proposed form, does not linearize the constraints, which makes it impossible to solve the optimization problem through classical methods. However, the authors propose alternative methods which, given a "close enough" estimate, can reportedly solve relatively large problems in under a second. This technique is able to achieve remarkable precision, even surpassing manual methods of measurement in this field.

Lastly, in [6], an image-based method is proposed as the solution to the map alignment and merging problem. This approach operates on occupancy grids, commonly produced by

SLAM implementations.

Given two equivalent but misaligned occupancy grids, the matter of visually aligning them is trivial: we simply try a few rotations, and see which one matches both maps so that they appear to be a single map. Analogously, merging two different grids which have some degree of juxtaposition, *i.e.* have one or a few common locations, also seems simple: we identify these common locations and superpose them.

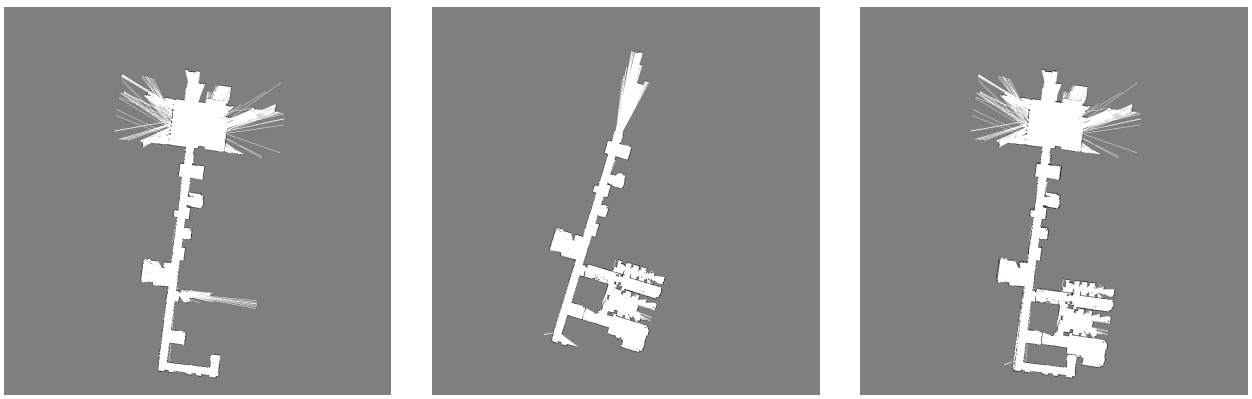
It is, then, relatively easy to align two occupancy grids: we simply have to rotate and superpose them until we can find common locations that match or, in other words, we find *agreement* between the grids. Computationally, all we have to do is attempt every possible combination or rotation and translation and, given some common areas, we can guarantee that we will find a combination of rotation and translation that correctly merges both maps. However, trying every single one of these possibilities would take a brutal amount of time and would be a prohibitively long process. Thus, we need a faster and more elegant way of addressing this issue.

The Hough Transform is an established method for detecting lines and other parametric-natured forms, such as circles or ellipses. The algorithm detailed in [6] uses the Hough Spectrum, as described in [7] for use in scan matching, to conduct its spectra-based determination of rotations.

The output of this method is a set of candidate transformations. Since multiple solutions are to be expected, we need a way of differentiating them; we need a metric that sets them apart and gives us an idea of which better fits the available data. Thus, the authors define an *acceptance index*, ω:

$$\omega(M1, M2) = \frac{agr(M1,M2)}{agr(M1,M2)+dis(M1,M2)},$$

where M1 and M2 are maps, M1 being the original map, and M2 being the second map, rotated and translated so that it, supposedly, fits the first map. *agr* is defined as the *agreement* between the maps, *i.e.*, the number of cells of M1 and M2 that are both free or both occupied. Analogously, *dis* is defined as the disagreement between the maps, which is the number of cells such that in a map one is occupied and free in the other, and vice-versa. Thus, given the maps we wish to merge, and applying a candidate transformation to the second, this metric allows us to distinguish between the results without having to visually inspect every candidate map. Fig. 2.6 briefly illustrates this approach.



(a) (b) (c)

Figure 2.6: An example of the results given by the approach described in [6]. (a) and (b) are individual maps, with a visible region in common; (c) is the final, fused map. Image taken from [6].

2.2.4 Previous Work on Multi-Robot SLAM

In [16], one of the first attempts at a Multi-Robot SLAM system is described. This technique is a generalization of the Rao-Blackwellized Particle Filter (RBPF) technique to the Multi-Robot SLAM problem. Although not clearly stated, this approach assumes that only one instance of the filter is running at any given time, and that data from all robots is processed in a centralized fashion.

Initially, it is assumed that initial poses are known, either by deploying all robots from the same place or by externally monitoring the mapping process. The algorithm is, then, generalized for the case where initial poses are unknown.

This algorithm does not explicitly use a map alignment/merging technique, rather merging data at the landmark and pose level. This merging is executed when robots "bump into" each other, *i.e.* when they find each other and measure their relative pose. This issue is discussed in [41], where the mutual detection and relative pose estimation problems are further explored.

The authors of [5] describe an interesting technique, also based on Rao-Blackwellized Particle Filters, which assumes each robot as an isolated entity, *i.e.* unlike in [16], robots take the steps of performing SLAM in an isolated manner, and occasionally and discretely exchange and merge information.

As before, merging occurs at the data level (as opposed to merging rendered maps), and relative poses are found using pan-tilt cameras. When two robots rendezvous, they exchange all data, transform it according to the measured relative pose, and integrate it into

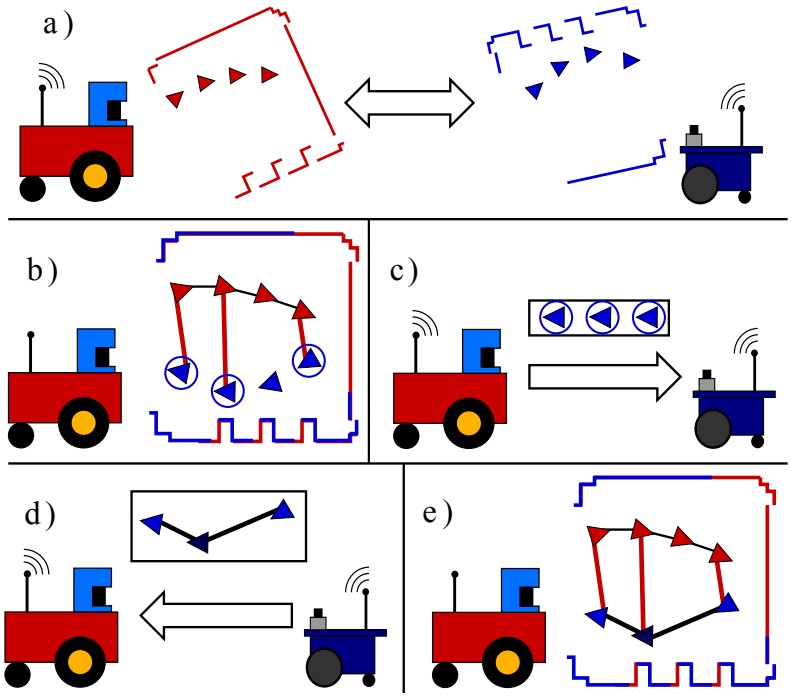


Figure 2.7: A visual explanation of the technique presented in [22]. Robots begin by trading poses and range scans (a). The first robot then determines which of the poses the first robot has provided are relevant to its mapping effort (b). This list is sent to the second robot (c) which replies with the relevant section from its graph (d). The first robot then integrates that section in its own graph (e).[2]

their respective filters.

In [22], the authors present an interesting approach to Multi-Robot Graph-Based SLAM. This technique is based on *condensed graphs* and on a novel efficient communication model. The authors assume that no communication infrastructure is present, *i.e.*, all communication is conducted *peer to peer*, robot to robot.

Local maps are transmitted under the form of a single range scan, the latest acquired, and a set of up-to-date, adjusted poses. Robots perform this operation at each step, *i.e.* at every new node. The transmission of up-to-date poses at each step enables each robot to provide its peers with the best estimate it can generate of its past poses. Additionally, by transmitting all range scans, one at a time, the authors ensure that all communicating robots have knowledge of each other's local maps, and that communication is as reliable as possible, since it is assumed that the probability of a message being successfully delivered decreases with its size. This assumption is not directly supported by evidence in [22], but is quite simple to support: long messages have to be segmented into smaller ones, known as

[2]Image taken from [22] with the author's permission.

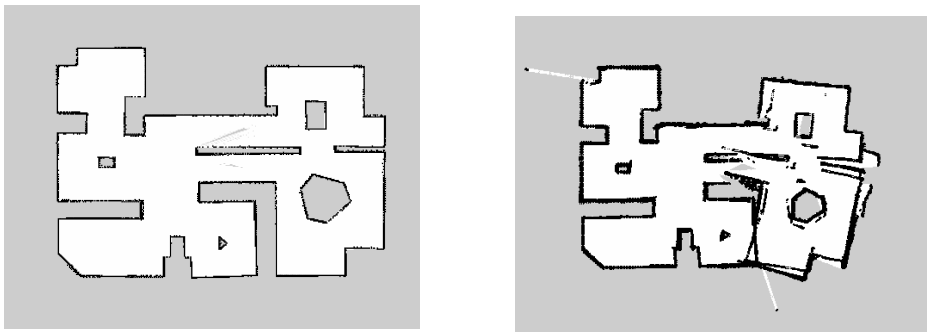
packets. Since all packets have a non-zero probability of being lost, and since the loss of a single packet implies the failure of the transmission of the whole message, it is easy to see that the probability of a message being successfully delivered decreases with message size.

Map fusion is based on keeping, on each robot, a list of candidate edges between the robot's graph and each teammate's graph. These edges are found by employing a correlative scan matching technique, which consists of matching scans originating from different robots, in a way similar to the process mentioned in [12] and in Fig. 2.5. Each of these new edges suggest a transformation between graphs. A RANSAC (RANdom SAmple Consensus) technique is used to determine a consensus among these transformations. The edges that are considered inliers by the RANSAC method are then communicated to the robot with which the map is currently being aligned, which then replies with a series of nodes, a *condensed graph* containing only the nodes affected by these edges. These new nodes are, then, added to the first robot's graph, resulting in a higher consistency of the map in the overlapping area and in the knowledge of relative poses in that area, which allows for the merging of the two maps.

In [26], a preliminary Multi-Robot SLAM approach is proposed in the context of the CHOPIN project [32]. This approach uses *GMapping* [14], an RBPF approach, as its core SLAM algorithm. Essentially, this technique consists of having multiple robots running *GMapping* and communicating over a MANET. Periodically, if certain conditions are met, pairs of robots exchange their local maps under the form of occupancy grids. Then, these are aligned by means of *image registration*, via the preexisting *mapstitch* ROS package, which was slightly adapted to serve the author's needs, namely by employing the SURF (Speeded Up Robust Features) class available on the OpenCV toolkit.

While being a preliminary, relatively simplistic approach, this implementation has shown that Multi-Robot SLAM using MANETs, and in a completely distributed fashion, is indeed possible, an important result for our work. However, this approach does not take efficient communication into account, which is counterproductive in the context of the CHOPIN project: we are interested in making all communication efforts as efficient as possible, to account for the possibility of unreliable communication.

Lastly, this implementation did not show satisfactory performance in real-world tests: the map alignment and merging technique used did not provide results at par with other current image-based techniques, such as [6], as shown in Fig. 2.8. While noise is to be expected, the merged maps show considerable misalignment due to limitations in the alignment method used.



(a) Simulation Results. (b) Real-World Results.

Figure 2.8: An illustration of the results given by the map alignment/merging process used in [26]. Images taken from [26].

2.3 ROS: Robot Operating System

2.3.1 What is ROS?

Outside the realm of theoretical speculation and analysis, techniques must exist as compilable code so that they can be experimentally validated. In order for an approach to be effortlessly and effectively testable by the robotics community, a common software framework must be established, with the goal of reducing to a minimum the time it takes to prepare a solution to be tested. ROS, standing for Robot Operating System [30], achieves just that. It is a software framework that aims to ease the development of modular code targeting robots, and a very widespread tool in the field of robotics. This is the software framework used in the context of the CHOPIN project, and is consequently the one we will use in our implementation.

As described in [30], ROS establishes a communication layer running on top of an ordinary host operating system, which allows for the exchange of messages between multiple *ROS nodes*. ROS nodes are programs which use the facilities provided by ROS to communicate and perform their tasks. ROS nodes operate independently and concurrently, and need not even be running on the same computer.

Communication is achieved through two main mechanisms: *topics* and *services*, which both carry *messages*. Topics are a means of asynchronous communication: a node publishes messages in a topic, to which other nodes may or may not have subscribed. Once a node publishes a message on a topic, all nodes that have subscribed to that topic receive that message. Services, on the other hand, provide synchronous communication between two nodes, and require that both a request and a response message be transmitted between the

communicating nodes in order to be successful.

ROS nodes can be implementations of all kinds of functions: data management, database access, mathematical functions, or anything else that can be implemented in any of the languages supported by ROS. Hardware drivers are a prime example of just how powerful ROS's modularity is: for a given robot, it is possible to develop ROS nodes which subscribe to a set of standard topics to receive commands, and that implement the low-level code needed to relay those commands to the robot. Thus, it is possible to directly use code developed for other robots to, for example, implement a given exploration algorithm on our robot. ROS allows us to abstract away the hardware intricacies of many robots and to develop as if we were writing code that targeted a standardized robot. ROS promotes code reutilization and has almost become a *de facto* standard in Robotics.

2.3.2 ROS Packages

ROS software is distributed in packages, which generally contain code, messages types and other support files. Once obtained, usually via a repository, these are compiled using ROS's CMAKE toolchain, producing executable code. We can find several ROS packages that implement SLAM algorithms, and we shall look through some examples in this section.

SLAM packages in ROS usually deal with two kinds of data: range scans and odometry. In other words, the ROS node in charge of the mapping process subscribes to topics which are published by the node(s) in charge of relaying information from the robot's sensors. Given this data, the SLAM node then creates a map, usually in occupancy grid form, and publishes it into a dedicated topic, thus transmitting it to any subscribing nodes. The SLAM node is also responsible for determining the robot's position in the map it created, thus constituting a solution to the complete SLAM problem.

ROS Packages for Performing SLAM

Examples of ROS packages dedicated to SLAM algorithms include *slam_gmapping*, *slam_karto* and *hector_slam*.

GMapping[3] is one of the most well-established SLAM algorithms available in ROS, and is described in [14]. *GMapping* implements a Rao-Blackwellized Particle Filter SLAM approach using an adaptive resampling technique [35]. This way, it is able to achieve acceptable results with a surprisingly low number of particles (about 30), which translates into low

[3] *GMapping* can be found on the ROS wiki at `http://wiki.ros.org/gmapping`.

computational requirements.

On the other hand, *slam_karto*[4] is a ROS package implementing SLAM using SRI International's *KARTO* package. This approach achieves results on par with *GMapping* [35]. However, unlike *GMapping*, this technique is graph-based, and relies on an approach similar to SPA to optimize its graph.

Finally, *hector_slam*[5] is a SLAM solution with an interesting characteristic: it does not rely on, or even require, odometric measurements in order to operate, which makes it capable of being used in mobile robots without odometric capabilities, such as quadcopters, ground platforms operating in uneven terrain, *etc.* Instead, Hector relies heavily on having a modern LIDAR (LIght Detection And Ranging, a device that functions similarly to RADAR but using LASER) scanner with a high scan rate, ideally 40Hz or higher.

2.3.3 Multimaster Communication

The *roscore*[6], a collection of nodes and programs that are pre-requisites to any ROS-based system, including the ROS Master, is a critical part of the ROS framework, and is responsible of handling the communication between nodes. Each ROS system uses a single *roscore* to overview all operation, and it is of the utmost importance that no node ever lose the ability to communicate with it. Multi-Robot systems in ROS are usually comprised of a central machine running the *roscore*, and multiple slave machines installed on each robot, each running the necessary nodes for the operation of their robot. ROS *namespaces* are instrumental in achieving this. Namespaces allow us to nest all the nodes and topics we need for a single robot under a common name, *e.g. robot_1*, and then replicate this system, as is, under different prefixes. It is as if each robot were a folder, and all we needed to do in order to add a new robot for our team were to copy all the necessary files to that new folder; in fact, we do not even have to copy anything, ROS handles that for us, too.

However, one of the premises of the CHOPIN project is that communication is not completely reliable. As such, we need a system in which robots are able to cooperate but also capable of working alone if communication becomes impossible. We need, then, multiple ROS systems, one for each robot, each with its own *roscore*, in order to have a scalable, fault-tolerant system.

Communication between multiple *roscores* is not supported by ROS out-of-the-box: ROS

[4]slam_karto can be found on the ROS wiki at `http://wiki.ros.org/slam_karto`

[5]hector_slam can be found on the ROS wiki at `http://wiki.ros.org/hector_slam`

[6]More information on the *roscore* can be found at `http://wiki.ros.org/roscore`

systems always assume the existence of a single core that manages all communication between nodes. There is, however, interest in multi-core (also known as *multimaster*) systems[7].

In the context of the CHOPIN project, a workaround based on the *wifi_comm*[8] is currently being used [26]. This package is used to propagate messages between various independent ROS systems, by mirroring a topic on a remote machine: any messages published to that topic on the sending robot will be broadcast, under the same topic, on the receiving robot(s). This allows for the exchange of arbitrary data between different ROS networks and, thus, the implementation of Multi-Robot SLAM systems in conditions in which communication is not reliable: by isolating a ROS system in each robot, lack of communication with the rest of the network does not implicate a lack of communication with the *roscore*.

The *wifi_comm* package can be used with a multiplicity of routing algorithms. By default, it uses the Optimized Link State Routing (OLSR) protocol [36]. In the context of the CHOPIN project, a custom routing algorithm was designed, known as CHOPIN Routing Algorithm, or simply CHOPIN [28]. The *wifi_comm* package has been successfully tested using this routing algorithm [28].

2.4 Summary

In this chapter, we have taken an overview of the bases upon which the upcoming work is set.

We have briefly described the theoretical foundations of SLAM, as well as taken a look at various classical approaches. We have also discussed several important issues inexistent in Single-Robot SLAM, such as communication difficulties, Mutual Detection and Map Alignment and Merging. We have also discussed a few recent practical implementations of the Multi-Robot SLAM paradigm, including a preliminary technique already developed in the context of the CHOPIN project.

Finally, we have briefly discussed the inner workings of the ROS platform, as well as some of the SLAM techniques already created for it. The next chapter will discuss the issue of Efficient Multi-Robot Communication, and the usage of compression as a possible solution.

[7] As seen in `http://wiki.ros.org/sig/Multimaster`.

[8] The *wifi_comm* package can be found at `http://wiki.ros.org/wifi_comm`.

3 Efficient Communication in Multi-Robot Systems

3.1 The Need for Efficient Communication

Cooperation among mobile robots almost always involves the use of a wireless network. Commonly, this network is taken for granted and little care is taken in minimizing the amount of data that flows through it, namely to assist the robot's navigation through the environment.

However, in harsher scenarios, such as search and rescue operations considered in the scope of the CHOPIN project, constrained connectivity can become an issue, and caution must be taken to avoid overloading the network. Additionally, in real-world applications, the navigational effort can be but a small part of the tasks that must be dealt with by a complete robotic system [32]. Therefore, it should operate as efficiently as possible. An efficient model of communication is also a key element of a scalable implementation: as the number of robots sharing the network increases, the amount of data that needs to be communicated does as well. Thus, greater care in preparing data for transmission is needed, so as to avoid burdening the network by transmitting redundant or unnecessary data.

Efficient inter-robot communication is not an area devoid of research. Other works, such as [2], [22] and [8], have worked on a solution for this issue by creating new models of communication for robotic teams, *i.e.* by developing new ways of representing the data needed to accomplish the mission. Other research efforts focused on developing information utility metrics, *e.g.* by using information theory [31], which the robot can use to avoid transmitting information with a utility measure below a certain threshold. We could find none, however, that applied compression to further increase their optimization gains. These techniques, while successful in their intended purpose, rely on modifications to the inner workings of their respective approaches. In our case, we intend to create a generalized

optimization solution, *i.e.* that does not depend on modifications to the intricacies of the underlying techniques.

General-purpose data compression techniques, on the other hand, are able to deal with any kind of data, with varying success. Compression methods are widely used in the transmission and storage of bulky data, such as large numbers of small files, logs, sound and video, even being used by default in specific file systems, and their main objective is to represent data in as few bytes as possible, regardless of its contents. At first sight, these offer us the solution to our problem: a way of achieving efficient communication without having to rework the SLAM technique's basic functionality, so that we can build our approach to be as general as possible.

3.2 Is Data Compression a Suitable Solution?

To ascertain the suitability of data compression as a solution to the problem of optimizing multi-robot communication, we must first explore how these techniques work.

Data compression is a process through which we aim to represent a given piece of digital data using fewer bytes than the original data. This process can be viewed as a way of trading excess CPU time for reduced transmission and storage requirements. Compression methods are divided into two main groups: *lossless* methods, which make it possible to reconstruct the original data without error; and *lossy* methods, which make use of the way humans perceive signals to discard irrelevant data. [34]

Lossy compression algorithms are commonly used in the compression of signals intended for human perception, such as image and sound. For example, given that the human hearing's capability ranges from about 20Hz to about 20kHz, sound compression techniques can remove any signal components outside that frequency range. Although the compressed data should be significantly smaller than the original, humans hearing sound reconstructed from lossy compressed data should experience much the same. The original signal, however, cannot be recovered.

Lossless compression, on the other hand, compresses data in a way that it is later fully recoverable. It is important that the algorithms we are employing be fully lossless, *i.e.* that the compressed data can be used to reconstruct the original data. For example, lossy image-based compression techniques, such as *JPEG*, could be used to reduce the size of an occupancy grid, processing it as an image. However, compression artifacts and other inaccuracies could lead to an erroneous representation of the environment, either by distorting its

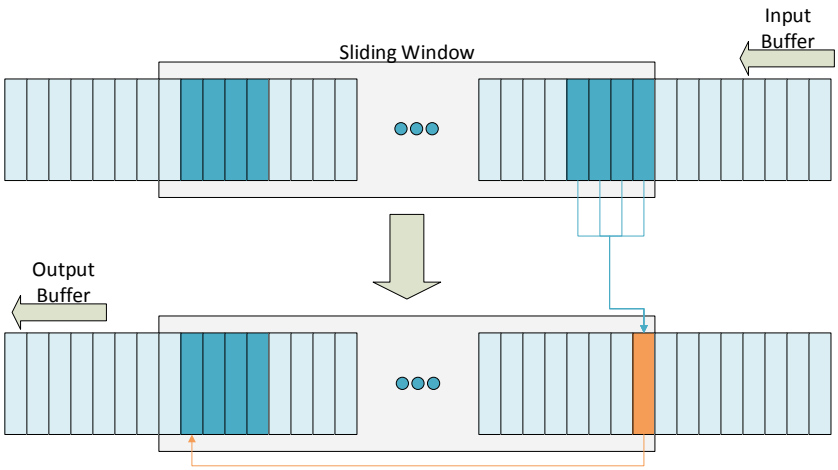


Figure 3.1: LZ77 operates by running a sliding window over the data. When a sequence in the input data is matched to data that is still inside the window, it is replaced with an offset-length pair that points to the previous instance of that data. In this figure, the dark blue segments were matched, and the second one is replaced with the orange, smaller segment, that points to the first copy of the matched segment.

features or by hindering other aspects of the multi-robot mapping effort, such as occupancy grid image-based alignment and merging [6].

In 1977 [42] and 1978 [43], Abraham Lempel and Jacob Ziv developed two closely related algorithms which were to become the basis for most of the lossless, general-purpose compression algorithms currently in use. LZ77 and LZ78, as their works were to become known, are methods of dictionary-based lossless compression. Summarily, the LZ77 and LZ78 algorithms keep a *dictionary* of byte chains encountered throughout the uncompressed data, and replace repetitions of those chains with *links* to entries in the dictionary, thus reducing the size of the data. The operation of these algorithms is illustrated in Figs. 3.1 and 3.2.

As we have seen in Chapter 2, occupancy grids are metric representations of the environment. In their simplest form, they are composed of only three values, one for occupied cells, one for free cells and a third for cells which state is unknown. Thus, they are repetitive by nature [38]. In larger environments, or at greater resolutions, this translates into the existence of large matrices filled with only three different values, often containing very long chains of repeated cells. These chains can potentially make the occupancy grids very compressible: an entire chain can be replaced by a much smaller offset-length pair that references an entry in the compression dictionary. Thus, compression seems a likely adequate solution to this problem.

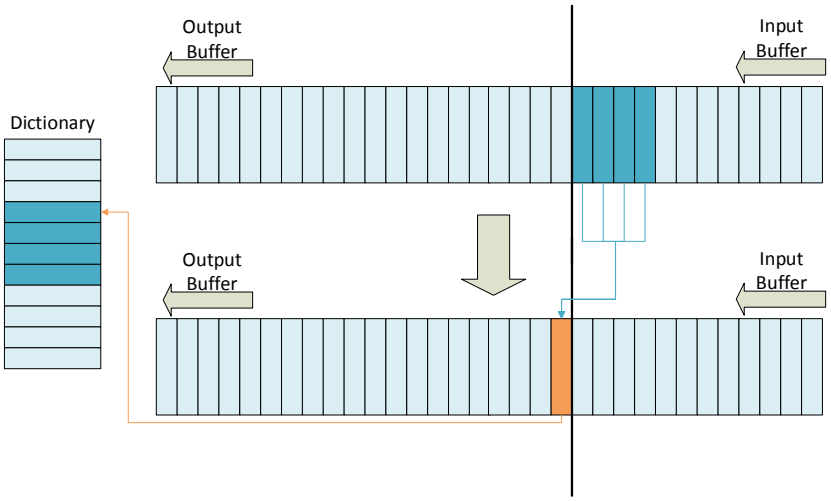


Figure 3.2: LZ78 operates by building an explicit dictionary. As the input data is consumed, the algorithm attempts to match each input sequence with an existing sequence in the dictionary. If the matching operation fails, the new data is added to the dictionary. This illustration shows the case where a match is found. In that case, the dark blue segments are matched to an entry in the dictionary, and replaced in the output buffer with the orange, shorter segment that points to the correct entry in the dictionary.

The usage of compression techniques, however, creates a new problem: since there are several techniques in existence, which one is the most suitable as a solution to our specific problem? To answer this question, we have developed a new compression technique benchmarking utility, which is presented in the next section.

3.3 Selection of a Data Compression Technique

3.3.1 Lossless Compression Algorithms

We have restricted our choice of algorithms to those based on Lempel and Ziv's work, for their focus on reducing redundancy by exploiting repetition and for their *lossless* nature.

LZ77 and LZ78 inspired multiple general-purpose lossless compression algorithms, widely used today as Free and Open Source Software (FOSS) implementations. We have collected the ones that we believe are the most suitable as solutions to our problem, given their availability, use and features. We will summarily discuss them next.

DEFLATE[1], presented in [9], is the algorithm behind many widely used compressed file

[1] *zlib* is available at `http://www.zlib.net/`

formats such as *zip* and *gzip*, compressed image formats such as *PNG*, and lossless compression libraries such as *zlib*, which will be the implementation through which DEFLATE will be tested. This algorithm combines the LZ77 algorithm with Huffman Coding [17]. The data is first compressed using LZ77, and later encoded into a Huffman tree. Being widely used, this technique was one of the very first to be considered as a possible solution to this problem.

LZMA[2], which stands for Lempel-Ziv-Markov Chain Algorithm, is used by the open-source compression tool *7-zip*. To test this algorithm, we have used the reference implementation distributed as the *LZMA SDK*. No extensive specification for this compressed format seems to exist, other than its reference implementation.

LZ4[3] is an LZ77-based algorithm focused on compression and decompression speed. It has been integrated into the Linux kernel and is used on the BSD-licensed implementation of ZFS [33], OpenZFS, as well as other projects.

QuickLZ[4] is claimed to be "the world's fastest compression library". However, the benchmark results provided by its authors do not compare this technique to either LZ4 or LZMA, warranting it a place in our comparison.

Finally, Snappy[5], created by Google, is a lightweight compression library that aims at maximizing compression and decompression speed. As such, and unlike other techniques, it does not employ an entropy encoder like the Huffman Coding technique used in DEFLATE.

There are several examples of compression benchmarking tools[6]. However, we found none that focus on the algorithms' ability to optimize inter-robot communication, namely the compression of occupancy grids. Their main focus is on comparing the techniques' performance on the compression and decompression of standard datasets, such as long sections of text, random numbers, *etc.* The need to test these techniques in the compression of specific, Robotics-related datasets, as well as the need to do so in a methodical, unbiased way, compelled us to create our own benchmarking solution.

[2]The LZMA SDK used is available at `http://www.7-zip.org/sdk.html`

[3]LZ4 is available at `http://code.google.com/p/lz4/`

[4]QuickLZ is freely available for non-commercial purposes at `http://quicklz.com/`

[5]Snappy is available at `https://code.google.com/p/snappy/`.

[6]Such as Squeeze Chart (`http://www.squeezechart.com/`) and Compression Ratings (`http://compressionratings.com/`).

3.3.2 Benchmarking Methodology

Compression benchmarking tools usually focus on either looking for the fastest technique, or for the one that achieves the highest compression ratio, as defined by:

$$R = \frac{L_U}{L_C}, \tag{3.1}$$

where R is the compression ratio, L_U is the length of the uncompressed data, and L_C is the length of the compressed data.

When choosing among a collection of compression techniques, compression ratio is a metric of capital importance, since the better the ratio, the less information the robots have to send and receive to complete their goal. However, the techniques' compression and decompression speeds are also important; an extremely slow, frequent compression may jeopardize mission-critical computations. Thus, we cannot simply find the technique that maximizes one of these measures; there is a need to define a new, more suitable performance metric, in order to find an acceptable trade-off.

Therefore, we define:

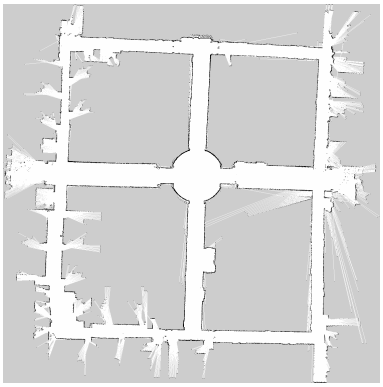
$$E = \frac{R}{T_c + T_d}, \tag{3.2}$$

in which E is the technique's *temporal efficiency*. It is determined by dividing the compression ratio achieved by the technique, R, by the total time needed to compress and decompress the data, T_c and T_d, respectively. The purpose of this quantity is to provide an indication of how efficiently the technique at hand uses its computational time. The algorithm that achieves the highest temporal efficiency, while at the same time achieving acceptable compression ratio, is a strong candidate for integration in work that requires an efficient communication solution, provided that its absolute compression ratio is acceptable.

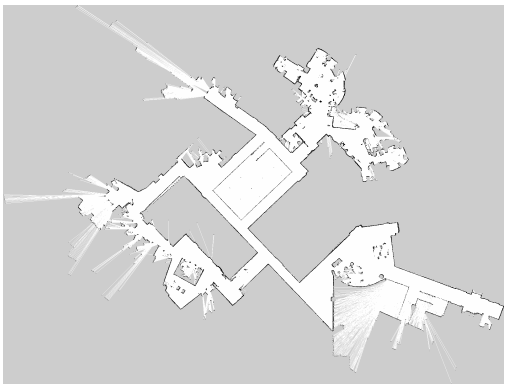
In order to test these techniques, a benchmarking tool[7] was developed. Given a number of compression techniques, the tool executes them over occupancy grids generated by SLAM algorithms, outputting all the necessary data to a file. This tool allows us to both apply the techniques to the very specific type of data we wish to compress, as well afollowings test them all in the same controlled environment. It was designed to be simple and easily extensible. As such, the addition of a new technique to the benchmark should be trivial for any programmer with basic experience.

[7]The tool is publicly available under the BSD license at `https://github.com/gondsm/mrgs_compression_benchmark`.

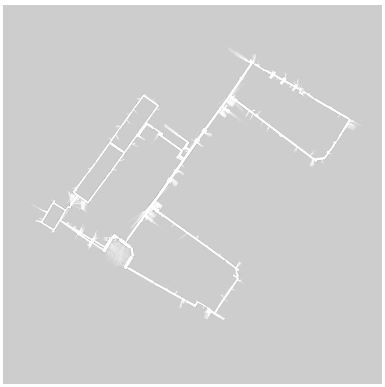
(a) Intel's Research Lab, measuring 753,078 bytes uncompressed.



(b) ACES Building, measuring 1,280,342 bytes uncompressed.



(c) MIT CSAIL Building, measuring 1,929,232 bytes uncompressed.



(d) MIT Killian Court, measuring 9,732,154 bytes (low resolution rendering) and 49,561,658 bytes (high resolution rendering) uncompressed.

following

Figure 3.3: A rendering of each dataset used in our experiments. These were obtained by performing SLAM over logged sensor data.

In order to test the effectiveness of compression algorithms in treating typical occupancy grids, and given the intention of studying, at least to some degree, how each algorithm behaves depending on the dataset's size, five grids of different environments were chosen: Intel's Research Lab in Seattle; the ACES building, in Austin; MIT's CSAIL building and, finally, MIT's Killian Court, rendered in two different resolutions, so that differing sizes were obtained. The datasets are illustrated in Fig. 3.3. The occupancy grids we present were obtained from raw sensor logs using the *GMapping*[8] [14] SLAM algorithm, running on the ROS [30] framework. The logs themselves have been collected using real hardware by teams working at the aforementioned environments, and are typically used for benchmarking

[8]A description of the version of GMapping can be found at `http://wiki.ros.org/slam_gmapping`.

SLAM techniques [21], being publicly available.[9]

To account for the randomness in program execution and interprocess interference inherent to modern computer operating systems, each algorithm was run over the data 100 times, so that we could extract statistically relevant results that were as isolated as possible from momentary phenomena, such as a processor usage peak, but that reflected the performance we could expect to obtain in real-world usage. Results include the average and standard deviation of the compression and decompression times for each technique and dataset, as well as the compression ratio achieved for each case. These results can be seen in Figs. 3.4 and 3.5, or textually in Table 3.1. Each technique was tested using their default, slowest and fastest modes, except for QuickLZ and Snappy, which only provide one mode of operation, and LZ4, which only provides a fast (default) and a slow, high compression mode.

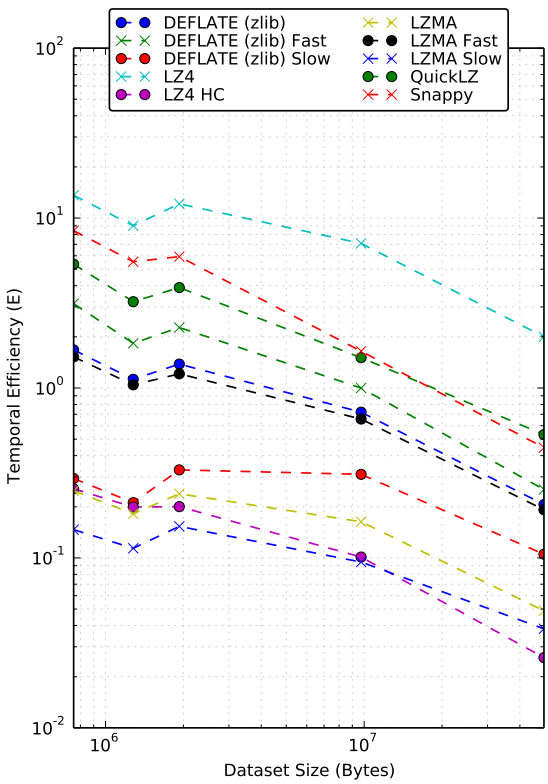
All tests were run on an Intel Core i7 M620 CPU, with 8 GB of RAM, under Ubuntu Linux 12.04.

3.3.3 Results and Discussion

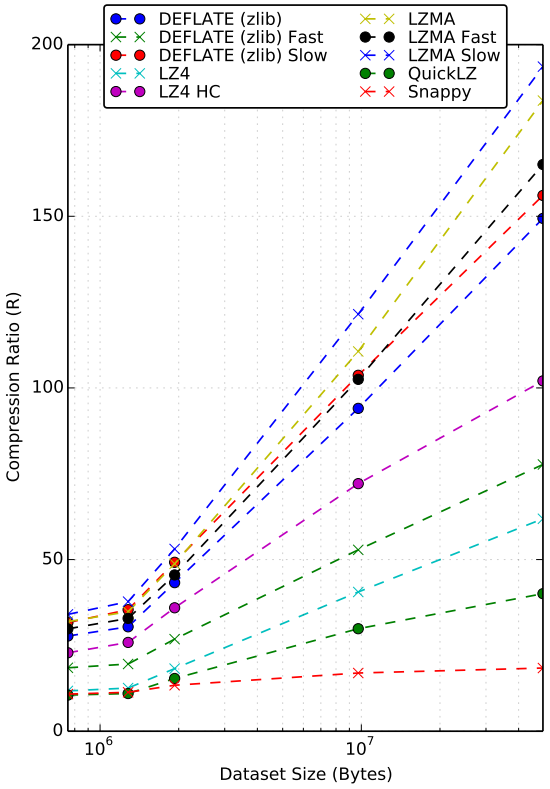
Fig. 3.4 and Table 3.1 illustrate the obtained results. In Fig. 3.4a, we show the general trend in temporal efficiency for each technique as the size of the map grows. The general tendency is for efficiency to decrease as the data increases in size. However, in Fig. 3.4b, we can observe that the compression ratio achieved tends to grow with data size. This effect can be attributed to the fact that, as the map grows, there are longer sequences of repetitive data, such as large open or unknown areas. It can also be explained, to a much smaller degree, by the fact that every compression technique adds control information to the compressed data, and that the size of this control data tends to be less significant as the uncompressed data grows. These figures lack error bars or other uncertainty representations due to the small dispersion of results, illustrated in Table 3.1 by the small values of standard deviation.

As expected, slower techniques generally achieve higher compression ratios. However, our results show that some techniques are indeed superior to others, in both temporal efficiency and compression ratio. LZ4 has shown both a higher temporal efficiency and compression ratio than that of QuickLZ and Snappy, making it a clearly superior technique, in this case. However, LZ4 HC, LZ4's slower mode of operation, is an inferior technique, both in temporal efficiency and ratio, when compared to LZMA and DEFLATE in the compression

[9]The raw log data used to create these maps is available at `http://kaspar.informatik.uni-freiburg.de/~slamEvaluation/datasets.php`.

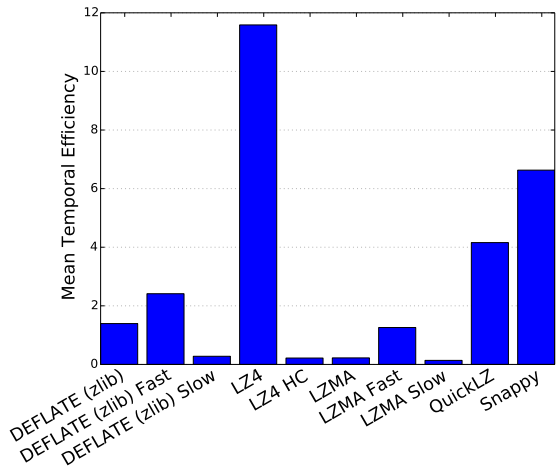


(a) Temporal efficiency for each of the techniques and datasets.

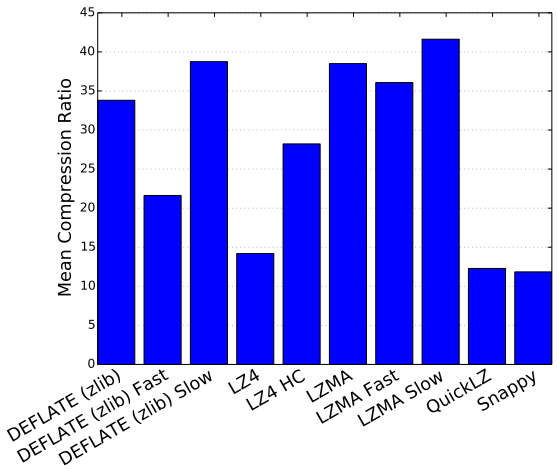


(b) Ratio achieved by each technique for each dataset.

Figure 3.4: A graphical illustration of each technique's performance on all datasets. Each of the dotted lines connect data points for the same technique, so that trends become evident. Note the logarithmic scale in some of the axes.



(a) Mean temporal efficiency achieved by each technique for the three smaller datasets.



(b) Mean compression ratio achieved by each technique for the three smaller datasets.

Figure 3.5: A graphical illustration of each technique's performance on smaller datasets.

Table 3.1: Results obtained by processing the smallest, intermediate and largest datasets 100 times with each technique. E stands for Temporal Efficiency, as defined in Sec. 3.3.2. σ_c and σ_d correspond to the standard deviation of the compression and decompression times, respectively. $\bar{T}_c$ and $\bar{T}_d$ correspond to the average compression and decompression times, respectively.

(a) Raw results obtained for the Intel Research Lab dataset.

	Ratio	E	$\bar{T}_c$ (ms)	σ_c	$\bar{T}_d$ (ms)	σ_d
DEFLATE (zlib)	27.727	1.675	15.130	1.179	1.423	0.140
DEFLATE (zlib) Fast	18.474	3.136	4.503	0.736	1.388	0.241
DEFLATE (zlib) Slow	31.633	0.293	106.519	4.167	1.306	0.195
LZ4	11.741	13.616	0.452	0.064	0.410	0.064
LZ4 HC	22.850	0.255	89.312	3.721	0.241	0.028
LZMA	31.920	0.248	126.282	7.315	2.364	0.287
LZMA Fast	29.825	1.524	17.080	1.156	2.487	0.181
LZMA Slow	34.029	0.147	229.789	13.086	2.290	0.242
QuickLZ	10.519	5.355	1.222	0.153	0.742	0.069
Snappy	10.807	8.427	0.753	0.128	0.529	0.100

(b) Raw results obtained for the MIT CSAIL Building dataset.

	Ratio	E	$\bar{T}_c$ (ms)	σ_c	$\bar{T}_d$ (ms)	σ_d
DEFLATE (zlib)	43.274	1.383	27.927	1.203	3.370	0.172
DEFLATE (zlib) Fast	26.818	2.269	9.100	0.382	2.717	0.178
DEFLATE (zlib) Slow	49.205	0.330	146.207	1.760	3.027	0.069
LZ4	18.236	12.129	0.779	0.052	0.725	0.090
LZ4 HC	35.953	0.200	179.027	2.698	0.432	0.087
LZMA	48.763	0.239	200.306	11.911	4.142	0.302
LZMA Fast	45.522	1.211	33.280	0.448	4.304	0.105
LZMA Slow	53.088	0.153	342.213	8.815	4.019	0.261
QuickLZ	15.359	3.898	2.533	0.117	1.407	0.088
Snappy	13.387	5.930	1.250	0.059	1.008	0.048

(c) Raw results obtained for the largest MIT Killian Court dataset.

	Ratio	E	$\bar{T}_c$ (ms)	σ_c	$\bar{T}_d$ (ms)	σ_d
DEFLATE (zlib)	94.044	0.721	111.906	1.738	18.610	0.492
DEFLATE (zlib) Fast	52.831	1.000	41.207	3.083	11.647	0.846
DEFLATE (zlib) Slow	103.676	0.310	316.500	5.208	17.499	0.717
LZ4	40.553	7.093	2.920	0.198	2.797	0.406
LZ4 HC	72.116	0.101	710.753	32.165	1.992	0.147
LZMA	110.622	0.163	663.896	15.645	13.595	0.527
LZMA Fast	102.493	0.657	141.536	1.216	14.580	0.316
LZMA Slow	121.472	0.095	1269.680	158.155	14.937	1.938
QuickLZ	29.856	1.508	14.027	2.274	5.774	0.612
Snappy	16.951	1.647	5.192	0.751	5.101	0.492

of larger datasets. Its temporal performance diminishes significantly with the growth in map dimensions, with an insufficient increase in compression ratio.

In applications where compression ratio is secondary relatively to speed, LZ4 is a strong candidate, and clearly the best among the techniques that were tested. It strongly leans towards speed and away from compression ratio, but offers acceptable ratios (around 15 for smaller maps, reaching 50 in larger ones) given its extremely fast operation. In other words, for applications which rely on transmitting occupancy grids, a very significant reduction of data flow can be achieved by employing this relatively low-footprint technique, which makes it suitable for use in real-time missions. As Fig. 3.4(a) shows, this technique is, by far, the most efficient at utilizing resources, achieving the best results in terms of temporal efficiency among the techniques that we have tested.

If further reduction in bandwidth is required, other techniques offer better compression ratios, at the expense of computational time. LZMA's fast mode offers one of the best ratios that we have observed, while still being acceptably fast. For the smallest dataset, this technique took, on average, about 15ms for compression, and achieved a ratio of 29.8. Depending on the application, 15ms of processor time per compression may be acceptable, given that this technique achieves a ratio that is almost three times as large as LZ4's, which achieved a ratio of 11.7 in 0.45ms, as visible on Table 3.1a.

In Fig. 3.5, we explore the case of the exchange of smaller maps, by averaging the temporal efficiency and ratio for each technique when operating over the smaller datasets. Smaller maps are commonly transmitted between robots at the beginning of the mission, when there is still little information about the environment. In these conditions, we note, as mentioned before, a generalized decrease in total compression ratio, and a narrowing of the gap between slow and fast techniques in terms of compression ratio: all techniques produce results within the same order of magnitude. However, the relationships between approaches in terms of temporal efficiency remain much the same. Thus, for smaller data, faster techniques appear to be a better option, since they achieve results that are comparable to those of their slower counterparts, at a much smaller cost in computational resources.

Larger maps, such as our largest examples, are very uncommonly transmitted during multi-robot missions, and hence unworthy of a closer analysis. Additionally, for these larger datasets, the multi-robot SLAM technique employed may make use of delta encoding techniques for transmission, transmitting only, for example, the updated sections of the map. In this case, we expect that the compression techniques applied to the map sections have the same performance as those applied to the smaller datasets in this test, since they will

effectively be compressing smaller maps.

It is important to note that even the worse-performing techniques have achieved significant compression ratios, with a minimum ratio of about 10. Consequently, by using data compression, we can reduce the total data communicated between robots during a mapping mission by at least a factor of 10, which shows the viability of compression as a solution for the problem of exchanging occupancy grids in a multi-robot system.

In our particular case, we want to use the technique that best utilizes our resources, as long as it demonstrates an acceptable compression ratio. By far, the best technique in terms of temporal efficiency is LZ4. Additionally, the compression ratio it achieves for smaller datasets is acceptable for our purposes; it still means a 10x reduction in necessary bandwidth.

3.4 Summary

In this chapter, we have explored the issue of efficiently communicating data through a multi-robot network.

We have summarily described the inner workings of dictionary-based compression algorithms, thus presenting them as a possible solution for our problem. We have then presented and discussed the results obtained during experiments in which these techniques were used in the compression of simple occupancy grid maps, with favorable results, concluding by selecting LZ4 as the most suitable technique for use in our Multi-Robot SLAM approach.

It is important to note that a paper resulted from this work [25], which was accepted for presentation at the ICINCO 2014 conference, and is available in this dissertation as Appendix A.

In the next chapter, we will present and describe a novel software solution for the Multi-Robot SLAM problem, which makes use of the results obtained in this chapter.

4 Proposed System

4.1 Overview

Throughout this chapter, we will describe the solution[1] for Multi-Robot SLAM that we have implemented, which is summarized in Fig. 4.1. The system is completely modular, and its main functionality is divided into five modules. These modules are distributed into four ROS packages, forming a distributable ROS stack, called *mrgs*, which is the main contribution of our work, and was its main focus.

The system can be run on top of any other ROS system that performs SLAM, as long as it conforms to ROS's standards, as illustrated in Fig. 4.2. In other words, this system can quickly enable any system running SLAM to run Multi-Robot SLAM.

Local occupancy grids generated by the SLAM technique in use enter the system one by one via the *Map Dam* node. This node crops out any excess empty space, attaches to the maps the local robot's local pose within the map, generated by the SLAM technique, under the form of a transformation between reference frames, and sends the newly-received map to the *Data Interface* node. These are then prepared to be sent to the team members, first by being compressed and then by attaching to them the local robot's MAC address. Finally, they are sent into the network.

As maps are being sent, the *Data Interface* node is also receiving grids sent from other robots, which are running the same process. These are all, both the local and the foreign grids, packed into a vector and sent to the *Complete-Map Generation* node. They are then iteratively fused into a single representation of the environment at the *Complete-Map Generation* node, which stores the fused occupancy grids into a tree-like data structure. The fusion itself takes place at the *Map Fusion* node, which provides a service for fusing occupancy grids in pairs.

The system does not impose a limit to the team size, nor does it depend on *a pri-*

[1]The implemented code can be freely accessed at `https://http://git.isr.uc.pt/mrl/mrgs`.

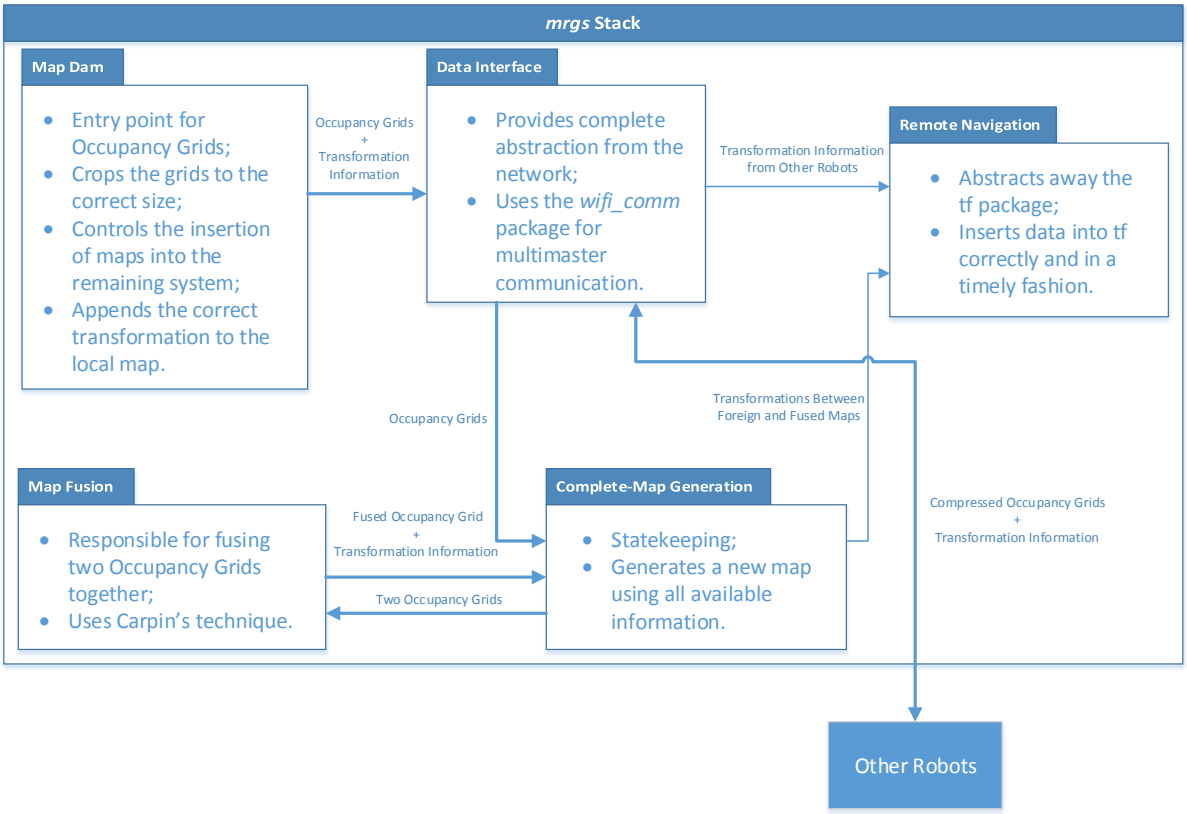


Figure 4.1: An overview of our system's design and data flow. This system is replicated in each of the team members.

ori knowledge of the team's composition. Both the *Data Interface* and the *Complete-Map Generation* nodes are able to deal with any number of teammates and maps, respectively.

This ROS stack also includes a fifth, support package, whose purpose is to hold various scripts and configuration files that, while an important part of the system in the practical sense, are not vital to its operation, and are not the result of a significant research effort; they are solutions for minor problems related to the solution's implementation.

From Section 4.2 onwards, we will present the various components of the system, divided by how they were segmented into ROS packages, in no particular order.

4.1.1 Modes of Operation

While the system's requirements only state that it should be able to operate in a distributed fashion, this implementation supports a total of three modes of operation by dividing the team members into three different *classes*: *distributed* robots, *central* robots and *mapping* robots. All classes are able to collaborate with one another, which makes this approach extremely flexible.

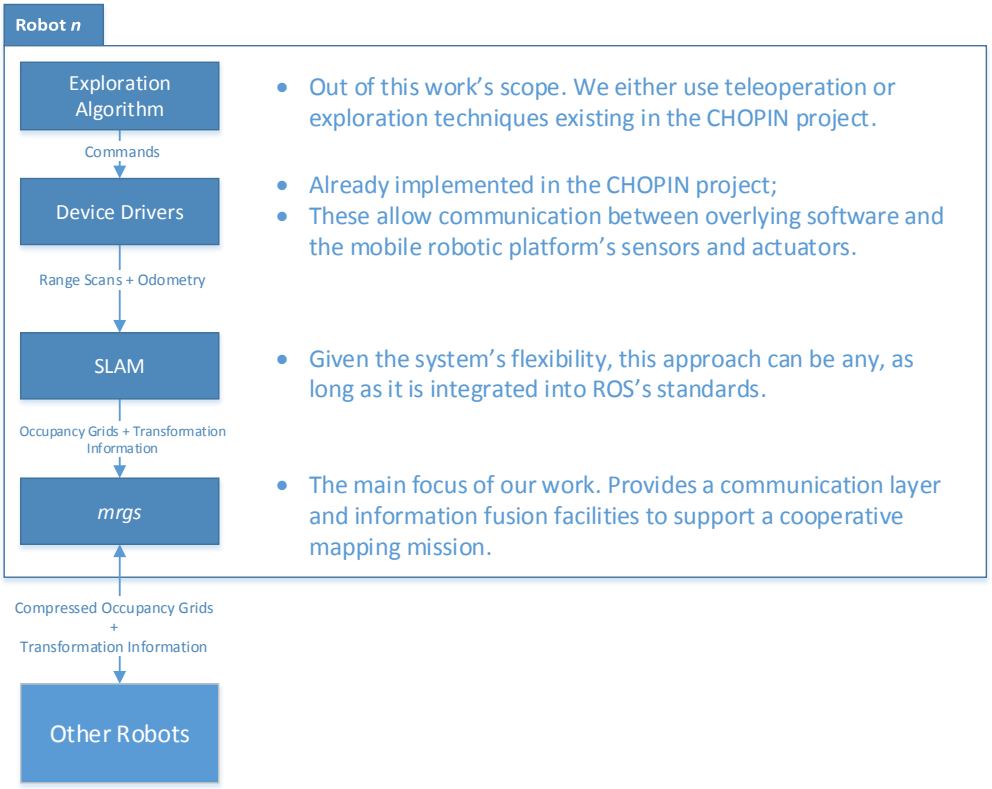
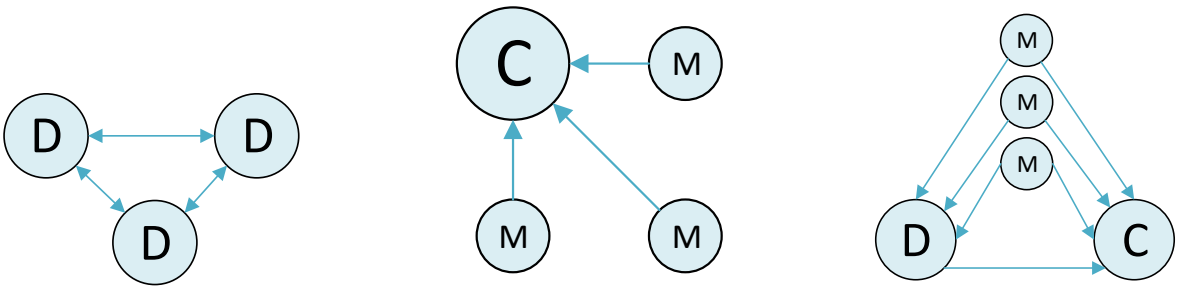


Figure 4.2: A general overview of the complete system that runs on each robot. The focus of our work is on the *mrgs* block, a ROS stack containing our framework for Multi-Robot SLAM. This stack enables any robot running Single-Robot system to communicate with its peers and to build a global representation of the environment.

The distributed mode of operation initially required is achieved by populating the team solely with robots running in the corresponding mode. These execute the full ROS stack: they gather, propagate and fuse data. This is the most common use case: we use a homogeneous team of robots to explore an unknown environment.

Mapping robots are the simplest of the three, they simply run a SLAM technique and use the *Map Dam* and *Data Interface* nodes to propagate the maps they have build. Central nodes, on the other hand, are assumed to be computationally powerful; they run the full stack, except for the *Map Dam* node, do not produce their own maps, and are solely tasked with building a global map based on information gathered by the mapping robots. Combining these two classes of robots produces the *centralized* mode of operation, which can be useful if the network we are using is reliable, fast and widespread. In this case, the central node may even run on a *base station*, *i.e.* in a powerful desktop computer.

Finally, a third mode of operation can be achieved by combining the use of robots run-



(a) Fully distributed mode, in which every robot gathers and fuses data.

(b) Centralized mode, where mapping robots gather data and central robots fuse it.

(c) Mixed mode, which can employ up to three classes of robots.

Figure 4.3: An illustration of the three modes of operation supported by the technique. In this figure, "D" nodes are distributed nodes, "C" nodes are central nodes and "M" nodes are mapper nodes. The blue arrows represent the flow of local maps.

ning in distributed mode with robots running in mapping mode. This mode combines the simplicity of the mapping robots with the versatility of the distributed ones, so that all robots in the field produce their own local map. Optionally, this mode of operation can employ central nodes, for added redundancy in the data fusion process. All three modes of operation are illustrated in Fig. 4.3.

4.1.2 Design and Implementation Principles

This stack was designed and built with efficiency, scalability and maintainability in mind. As such, all software strictly follows ROS's C++ guidelines[2], as well as Software Engineering-related good practices. The system was purposefully segmented into concurrently-executing modules so as to explore the flexibility of multi-core systems, as well as, for example, guaranteeing that the robot remains able to communicate during the map fusion process. Additionally, having a well-segmented system, *i.e.* not divided into too little or too many modules, maximizes the utility of the code by making it reusable.

Aside from inter-robot communication, all inter-module communication relies as much as possible solely on standard ROS data structures, in order to maximize compatibility with other systems and code reusability. Even custom communication structures, such as the messages exchanged between the *Data Interface* and the *Complete-Map Generation* nodes,

[2]The guidelines are available at `http://wiki.ros.org/CppStyleGuide`

have their roots in ROS's standard data types, ensuring that reusing specific nodes from this system within other ROS systems is a fairly straightforward process. All ROS topics used solely by our system are packed into their own *namespace*, to enable the user to quickly identify which topics belong to our system, as well as minimize possible interference with other systems running on the machine.

As a preliminary approach, the technique presented in [26] was a good candidate for code reuse. However, an analysis of the already existing code revealed that, in its current state, it would not be useful given the philosophy of the new approach that was designed. As such, it was decided that this new implementation would be built from the ground up. In our implementation, code reuse is restricted to the usage of the reference implementation of Stefano Carpin's *mapmerge* [6], all other code is completely original.

4.2 The *Data Interface* Node

This ROS node, existing in its own package, is responsible for dealing with one of the biggest challenges we have proposed to tackle: it is responsible for communicating all the relevant data between robots, and for doing so in an efficient manner. This node's operation is summarized in Fig. 4.4.

In the proposed approach, we have decided not to rely on explicit *rendezvous* events, since actively seeking a *rendezvous* with another robot is a complex and costly operation [27]. Instead, data is published into the shared topic, and received by all robots within range. This approach greatly simplifies the transmission of information: robots do not need to trade control messages, instead relying on the capabilities of the *wifi_comm* package to deliver all data.

One of our goals related to communication was that it be relatively robust to network failure. To partially solve this issue, this node uses MAC addresses to identify particular robots. MAC addresses are a better identification key than IP address, since they are not repeatable, not only in the same network but across all networks of physical devices; MAC addresses are tied to the hardware itself and are not subject to change during the network device's lifespan. This way, we guarantee that a robot is always able to clearly and indubitably identify the sender of a given message, regardless of the possible changes the sender's IP may suffer due to network failures. Additionally, ROS's transport layer protocol, TCPROS, makes use of TCP in the transmission of messages. Thus, we expect that messages are either delivered correctly or completely lost, since TCP applies error-checking measures

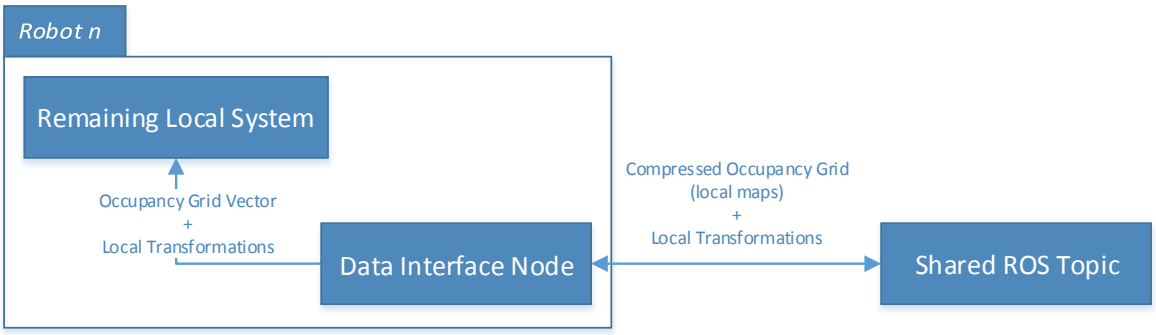


Figure 4.4: An illustration of the functionality of the *Data Interface* node. This node publishes into and subscribes to a topic that is shared among all robots. All information published into this topic is broadcast to all known robots in range in order to propagate the latest local map and transformation information.

that should prevent any messages from getting corrupted. Furthermore, a lost message does not compromise the mission; the system does not depend on a reliable connectivity with its peers, instead processing data as it arrives. If the connection to one of the peers is lost, the latest information it sent is used in the map fusion process.

Furthermore, this node is able to deal with a team of unknown size. Thanks to the combination of OLSR, *wifi_comm* and the usage of MAC addresses as identifiers, we are able to dynamically create a list of teammates that is able to grow as long as robots are added to the team.

The usage of OLSR [36] further strengthens our solution's robustness to network failure. Being a mesh networking routing algorithm, it is designed to recover from failures in links between nodes, and also to be able to establish new links between nodes as needed.

It is also required that communication be as efficient as possible. That goal is achieved by employing the LZ4 compression algorithm to all occupancy grids meant to travel through the network, following the results presented in Chapter 3.

The insertion of a new local map into the network is triggered by the insertion of a new map into the system, which means that all new local maps that pass through the *Map Dam* node are sent into the network. This methodology is meant to facilitate the generation of an interpretable map as soon as possible, by quickly relaying new information to all robotic agents. Furthermore, maps are transmitted regardless of any measured link quality; we believe the usage of the LZ4 data compression technique makes the maps small enough to go through low-quality links.

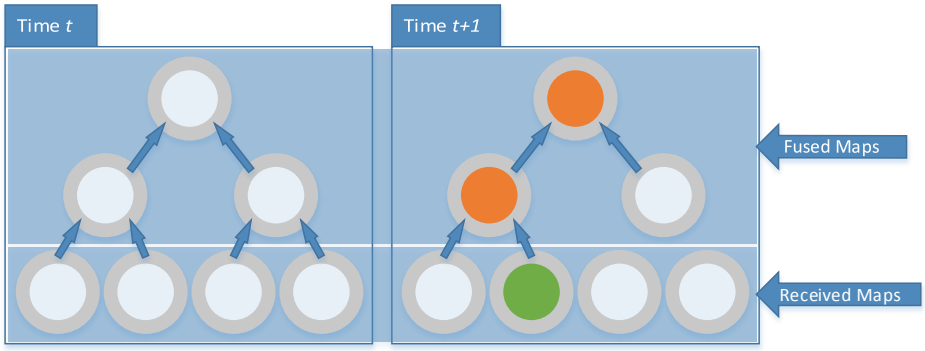


Figure 4.5: A pictorial description of the tree-like structure for storing and updating maps. At a given instant t, the system receives a vector of maps, and iteratively merges them until it obtains a single representation. When a new map is received (green), only the maps that depend on it must be re-merged (orange). Each pair of arrows represents a merging operation.

4.3 The *Map Fusion* Node

This ROS node, also inhabiting its own package, is responsible for fusing (*i.e.* aligning and merging) a pair of maps and for calculating the geometric transformations between the original occupancy grids' reference frames and the fused reference frame.

This module features the reference implementation of the alignment algorithm presented in [6] (and in Sec. 2.2.3) at its core. The aforementioned implementation is only capable of dealing with its own data representations, and only able to determine transformations and transform the maps, not to actually merge them. This node extends that implementation to be able to perform the remaining operations needed, while simultaneously *wrapping* it in ROS's standards, *i.e.* embedding it into a ROS node. This node also features a self-tuning ability, *i.e.* it gauges the computer's performance on startup, and adjust the amount of CPU dedicated to the alignment effort accordingly.

Since there is always the possibility of building a better, more robust algorithm for map fusion, special care has been taken to ensure that this node is as decoupled from the rest of the system as possible, so as to ensure that it can be easily swapped for another that communicates in the same way, much like one swaps a worn part of a car for a new one.

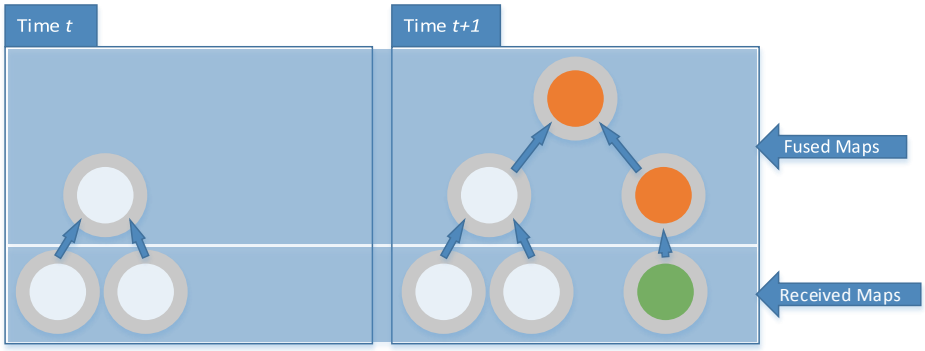


Figure 4.6: When a new robot joins the mission, the tree-like structure is grown sideways to accommodate the new data. Iterative merging then proceeds as usual, regardless of whether the current number of maps is even or odd.

4.4 The Complete-Map Generation Node

This node is responsible for building a representation of the environment using all the available information. It does so by controlling the *Map Fusion* node, which only fuses grids two at a time. This node receives a vector of occupancy grids and iteratively builds a tree-like data structure for storing occupancy grids, as represented in Fig. 4.5.

When a new occupancy grid is received, be it from the local or from a remote robot, this node starts building a new representation of the environment, incorporating into it the new information. Doing this in an incremental way, *i.e.* by trying to integrate the new map in the previously merged one, could lead to disastrous results: once a merging step fails, which is likely at the beginning of the mission due to the fact that little information is available, all future complete maps will include that error.

The tree-like representation of map storage allows us to reuse this information when new maps are received. When a new occupancy grid is received, we only re-merge the intermediate grids that depend on the newly-received grid, thus avoiding costly operation of completely rebuilding the final occupancy grid. This behavior is illustrated in Fig. 4.5. This method allows us to recover from an erroneous merge in a natural way: when new information is received, all grids that depend on it are rebuilt, thus giving the algorithm the opportunity to correct its previous error.

This structure is initialized with the first two maps received from different sources, *i.e.* the iterative map fusion process only starts when there are at least two different maps available. The structure then grows laterally as new teammates join the mission (or as their maps are

received for the first time) to accommodate more information, as illustrated in Fig. 4.6.

This node also keeps a similar structure for calculating the transformations between the various occupancy grids, so that the relative poses between robots can be determined. These are also updated as needed when new information arrives, and are fed into the *Remote Navigation* node every time a new Complete-Map is computed.

4.5 Auxiliary Nodes

A fourth package which contains two additional ROS nodes is included. While not difficult to implement or intricate in their operation, they are vitally important for the performance of the system.

The *Map Dam* node, as the name implies, intends to add control and intelligence to the way occupancy grids flow through our system, as well as further decouple our system's functionality from the behavior of the SLAM approach. This node receives intercepts all the occupancy grids output by the SLAM technique, and introduces them into our system. Before the introduction into the system, the grids are trimmed, *i.e.* any excess free space is removed from the grid.

The *Remote Navigation* node was designed to cope with the requirements of the *tf* ROS package. *tf* is responsible for managing the geometrical transformations between the various frames that compose the robot, and is an extremely useful and widespread tool. However, it requires that the information it receives be very carefully formatted and timed. It was also designed to work within a single robot, which creates a need to re-tag and re-format much of the information that is passed to it.

This last node receives transformation information from the *Complete-Map Generation* node and from the *Data Interface* node, and periodically propagates it into the *tf* topic, correctly tagged and formatted.

4.6 Summary

In this chapter, we have presented a novel solution for the Multi-Robot SLAM problem. We have described its structure, its behavior and the rationale behind most of the design decisions.

In the following chapter, we will validate the technique experimentally, testing both its general performance and its efficiency in communicating data.

5 Experimental Validation

5.1 Experimental Method

In order to evaluate and validate the performance of our system, experimental tests were performed, both in simulated and real-world environments. Since this is a Multi-Robot SLAM solution, the most appropriate way of validating its performance is using several robots in an appropriate environment.

A test comprising multiple robots must be a well-orchestrated and executed operation. Many real-world issues must be taken into account: batteries must be charged, wireless networks must be configured, several computers must be prepared, support personnel and hardware must be obtained, *etc.*

In order to test our system, we have segmented the experimental process into two steps: the data-gathering step, and the processing step. The data-gathering step consisted of collecting several *runs* of data from a single location, emulating an exploration performed by a team. We took advantage of ROS's facilities for data recording[1], ensuring that this data would later be played back as if it had just been acquired. This has enabled us to test the system with real-world data from the very beginning, by substituting the first blocks illustrated in Fig. 4.2 with a single ROS node tasked with playing back the recorded data.

The processing step took place in a controlled environment. Unit tests were performed on modules as they were constructed, and system tests were then run using a mixture of Virtualbox[2] virtual machines and real, physical computers, as illustrated in Fig. 5.1, each acting as a robot. This has allowed us to, while using real-world data, repeatedly test our system as it was improved.

This approach had the very significant advantage of providing a lifelike environment for testing the solution, down to the transmission of data through a network interface, that was

[1]Namely the *rosbag* tool, described at `http://wiki.ros.org/rosbag`.

[2]Virtualbox is available at `http://www.virtualbox.org`.

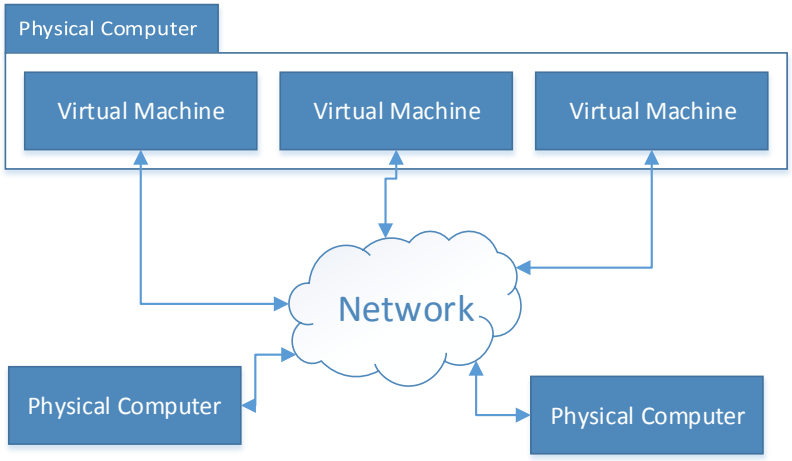


Figure 5.1: An illustration of the setup used to process data. Virtual machines and real computers can be used as equals, in any combination, in a network routed by OLSR.

not only repeatable but also moderately easy to set up, thanks to the ability of cloning and creating snapshots of virtual machines. The only significant drawback of this approach is the large amount of resources needed for running these virtual machines on a single host computer. However, this testing solution is lean enough be run using three simultaneous machines on a consumer-grade laptop with only 8GiB of RAM and a dual-core Intel i7-620M processor. Without the use of virtual machines, at least three computers would be needed for each test, which would have to be correctly configured and simultaneously operated, severely harming the repeatability of the experiment.

With the purpose of illustrating the system's performance, two main offline tests took place: one in the MRL Arena, illustrated in Figs. 5.2; and on ISR's corridors, which are illustrated on Fig. 5.3. For these experiments, two sets of data were gathered at the MRL Arena, and three were gathered at the corridors of the Institute, simulating a mission composed of two and three robots, respectively.

Tests with actual multiple robots also took place, in a setup similar to Fig. 5.2. These online tests were used to determine if the results we were obtaining in the test setup were correct, and to demonstrate the system's ability to operate in real-time. These tests also took place mainly in the MRL arena and the corridors of the Institute.

5.1.1 Hardware

Real-world data was gathered using one or several Pioneer 3-DX mobile platforms, shown in Fig. 5.4, running the software stack described in figure 4.2. These are equipped with

Figure 5.2: A small experiment taking place in the MRL Arena. In this experiment, a Traxbot and a Stingbot platforms were used. In this experiment, the centralized mode was used, to cope with the low processing power of the smaller laptops.

Figure 5.3: An illustration of the Institute's arrow-straight corridors, where experiments took place. In this instance, the distributed mode can be used, since the Pioneer 3-DX is able to carry larger laptops that are able to run the full system.

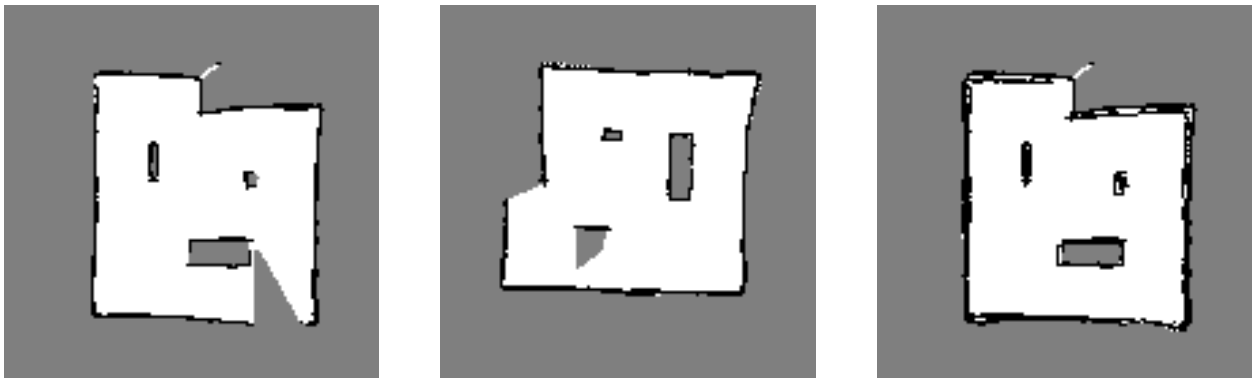
Figure 5.4: The Pioneer 3-DX platform used for data gathering. We can see, mounted on the small white platform at the front, the main sensor used in our experiments: the Hokuyo URG-04LX-UG01 laser range finder. Also visible are the Asus Eee laptop used for controlling the platform and recording data, as well as the Wiimote used for teleoperation.

a Hokuyo URG-04LX-UG01 Laser Range Finder, and the Pioneer's build-in encoders were used for odometry. The platforms were teleoperated using a Wiimote or a remote machine. For teleoperation and recording purposes, the platforms each had an Asus laptop mounted on it. The simulation and virtual machine-related operations were performed on a more powerful, if slightly dated, Toshiba Qosmio F60 laptop.

5.2 Results and Discussion

5.2.1 System Performance and Immunity to SLAM Technique Variation

In order to test the system's immunity to the variation of SLAM technique, *i.e.* its ability to perform its functionality regardless of the SLAM technique used in the mission, data was gathered in the MRL Arena, which was then processed using the three different SLAM techniques we have mentioned in Chapter 2: *slam_gmapping*, *slam_karto* and *hector_slam*, using their default parameters, namely regarding the rate at which they output maps, and their size. We have configured them all to use a resolution of 5cm/cell.



(a) (b) (c)

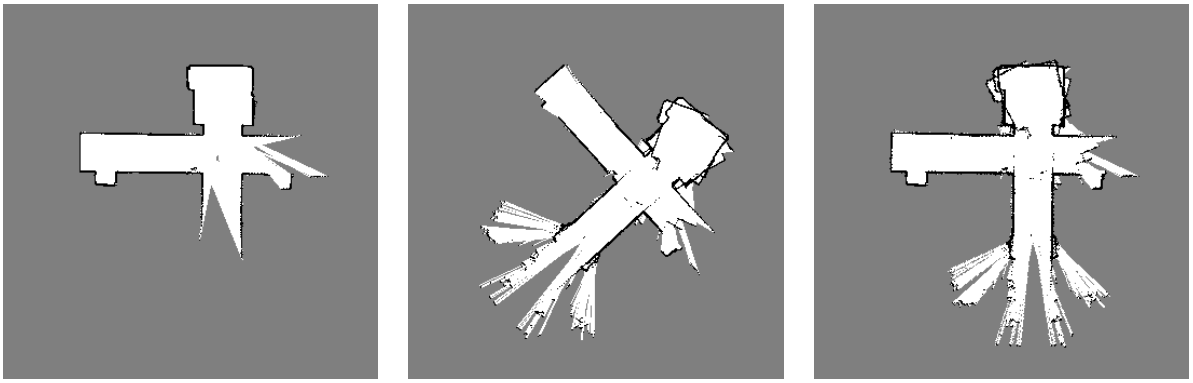
Figure 5.5: An example of the results obtained using *GMapping*. (a) and (b) are the local maps of each of the robots, (c) is the result of the map fusion process.

(a) (b) (c)

Figure 5.6: An example of the results obtained using *Karto*. (a) and (b) are the local maps of each of the robots, (c) is the result of the map fusion process.

(a) (b) (c)

Figure 5.7: An example of the results obtained using *Hector*. (a) and (b) are the local maps of each of the robots, (c) is the result of the map fusion process.



(a) (b) (c)

Figure 5.8: An example of a failed three-way map fusion. (a) is one of the robot's local map, and (b) is the result of the fusion of two other local maps. We can see that, while the maps were roughly rotated and translated correctly, there is a noticeable misalignment on one of the maps.

These three techniques, while equivalent in their results, have different operational requirements. For instance, *Hector* does not use odometric data. Their results are also different, both in format and quality. *GMapping* tends to output maps that have significant empty areas surrounding a region of interest, as does *Hector*; *Karto* outputs maps that are very closely trimmed to the region of interest. Our system was designed to deal with this issue by cropping all the input maps to include only the region that has useful information. The *Map Fusion* node then applies padding around the maps to guarantee that no information is lost during rotation.

Figs. 5.5, 5.6 and 5.7 show the results of the MRL Arena experiment for each of the SLAM techniques, which were run over the same recorded data. These results show our system's ability to handle and adapt to the SLAM technique in use, being able to successfully attain a global representation of the environment. We can see that while every SLAM technique builds a different slightly representation of the environment, our system is able to process the data regardless. It is also important to note that the map fusion process fuses the *second* map *into* the *first*, which is why there is a disparity in the orientation of the final representations.

The experiment that took place in the corridors involved a team of three robots. In this scenario, the *Map Fusion* node was expected to be able to deal with maps it had already fused before, as described and illustrated in Chapter 4. However, its performance revealed room for improvement, as illustrated in Fig. 5.8.

As illustrated, an unsuccessful fusion step can be catastrophic to the remaining effort,

Table 5.1: Network statistics for outgoing data obtained during in the MRL Arena and the ISR corridors. These are respective to one of the robots in the mission; the results obtained by its teammate are equivalent. N is the number of processed maps (output by the SLAM technique into our system), $\bar{R}$ is the average compression ratio achieved during the mission, L_t is the total size of the maps received by our system (before compression, after cropping), L_s is the total amount of data sent into the network by this robot and, D_s is the amount of data we have saved, *i.e.* the difference between the total size of the maps and the data actually transmitted, and, finally, T_p is the total time spent processing maps, in milliseconds. All sizes are in bytes.

(a) Results obtained during the MRL Arena mission.

	N	$\bar{R}$	L_t	L_s	D_s	T_p
GMapping	21	8.78	169062	19253	149809	2.77
Karto	6	8.03	48357	6015	42342	0.82
Hector	75	8.61	606667	70472	536195	9.61

(b) Results obtained during the ISR corridors mission.

	N	$\bar{R}$	L_t	L_s	D_s	T_p
GMapping	21	13.92	930050	66787	863263	7.68
Karto	6	12.06	209799	17402	192397	2.64
Hector	76	12.03	3198376	265883	2932493	14.99

utterly invalidating the final result. This unsuccessful fusion can be attributed to several factors, such as the fact that the corridors are almost featureless, which hinders the efforts of both the SLAM technique and the map fusion process; the presence of glass panes, visible in the lower part of the image, that interfere with the laser range finder; disparities in the way different maps represent the same real-world area...

5.2.2 Communication Efficiency

Table 5.1 illustrates the results obtained during the MRL Arena mission. Essentially, these show that the technique adopted to ensure efficiency in communication, compression using the LZ4 technique, is a viable option.

As postulated in Chapter 3, using compression on occupancy grids yields important data savings. In this case, using real occupancy grids (as opposed to the simplified ones used on Chapter 3), we have saved at least about 7/8 of all data meant to be sent, which equates to about 88% savings in data sent. These savings come at a very reduced cost, as is visible on the last column of Table 5.1. At the most, we spent a total of about 15 milliseconds

processing maps during a mission, which given that during that mission we saved 11/12, or 91.6%, on transmitted data, is a very positive result.

5.3 Summary

In this chapter, we have tested the solution presented in Chapter 4.

We have found its performance to be acceptable. The technique is able to successfully map an environment using the information gathered by multiple robots, and does so employing a very efficient communication method.

In the next chapter, we will reflect upon our work, analyzing these results in light of the requirements that were defined in Section 1.1. We will also briefly explore several lines of possible future work.

6 Conclusion

In Section 1.1, we have defined a set of goals for this work. We shall now reflect upon them and on our system's performance.

First and foremost, we needed our system to be completely distributed. By designing the *Data Interface* node to abstract away all the network-related details, and the *Complete-Map Generation* node to be able to fuse an unknown number of maps, we were able to build a system that, by design, does not limit the number of robots we can use; the bottleneck in our system's performance is the machine it is running on.

However, operating in a fully distributed way creates an overly redundant system, where resources are wasted in calculating the same results on various machines. While desirable in a high-risk scenario, a more modest solution was required for situations in which the risk of losing a robot was not as high. To solve this issue, we have designed our system to support three different modes of operation, which allow us to fully adapt to and exploit the risk-level of the situation.

Failures in communication were also taken into account, and to deal with them we make use of a combination of TCP with the usage of hardware-bound addresses for robot identification, so that a failure in the network does not compromise our ability to correctly identify the team members.

Scalability was also an important topic of this work, and was assured in a twofold way. Firstly, by employing compression in our communication, we have been able to save approximately 90% in required bandwidth, which allows us to greatly enlarge the team. Secondly, by designing the system so as to not depend *a priori* on the number of robots available, *i.e.* by making it able to dynamically embrace new team members, we have achieved a solution that does not have a theoretical limitation to the team's size.

The last requirement stated that the system be able to deal with any SLAM technique that was integrated in ROS's standards. To satisfy this requirement, we have designed our system to depend on those same standards as much as possible, and to retrieve information

in a standardized way. As shown in Chapter 5, we were successful.

6.1 Future Work

The system presented in this work constitutes a solid basis for the performance of Multi-Robot SLAM tasks. However, mainly due to calendar-related constraints, not all the features that were initially desired are included in the system. In this section, we will list and reflect upon the various improvements that the authors imagined and designed, starting with the simplest.

As of this writing, the system completely relies on ROS's facilities to display data to the user. As such, gauging the real-time performance of the system is limited to observing the evolution of the complete-map and the textual output of each of the nodes. It would be very useful to have a tool that graphically displayed the network's status, the bandwidth used by each connection, the current gains obtained by using compression, *etc.* A tool like this would greatly simplify the control of the mission, and would make information available "at a glance".

The *Data Interface* node is already efficient enough at transmitting data for our purposes. However, it would be interesting to explore the concept of *delta encoding* in an attempt to further increase its efficiency. Delta encoding is a broad concept that essentially consists of having a backlog of previous messages, and constructing the following message as a set of differences from on or more of the previous. In this case, delta encoding could be applied in one of two levels: at the map level, where each new map would be sent as a set of differences from the last; or at the buffer level, by sending each compressed buffer as a set of differences from the last. It would be interesting to test both hypotheses.

The *Map Fusion* node could also be vastly improved. As it stands, it is a mere example of what it could have been, since it was the recipient of the least amount of attention. Being very well decoupled from the rest of the system, it would be very interesting to see it replaced with a more robust approach, which could dramatically increase the success of the mapping effort itself.

The *Complete-Map Generation* node could also be improved to detect and deal with failed fusion attempts, for instance by discarding the newest fusion result and keeping the last successful one. Currently, a failed map fusion attempt has catastrophic results for the mapping mission.

This work does not address the issue of inter-robot interference, *i.e.* the fact that the

presence of a robot within another's scanning range violates the *static world* assumption, generating inconsistencies in the final map. It would be very interesting to look into the possibility of creating a way of filtering robots out of each others' laser scans without using additional sensors.

The current model of distributed operation is highly redundant: every single node that is operating has to create its own complete-map. Even though this model was a decided upon to maximize the tolerance to failure, it can represent a large waste of computational resources in scenarios where the network is trustworthy. An improvement over the current system would be to create a new mode of operation, where it would capable of performing the processing of maps in a distributed fashion, in a manner inspired by computer clusters. Robots would have to periodically sync, and trade maps as well as control messages to determine how the complete-map generation process would take place. Each robot would be responsible for building a part of the map tree, effectively distributing the computational load across the robotic network. For instance, for four robots, two of them would fuse two pairs of maps, and then send the fused maps to another robot, who would fuse them to create a complete-map, which would then be propagated across the network.

Finally, as a major improvement, the system could be upgraded to be able to deal with any kind of data. At its core, this system creates a framework that enables robotic peers to efficiently propagate their respective local representations of the environment, and to methodically fuse them with the information received from other robots. However, as we know, a representation of an environment is not limited to an occupancy grid, *i.e.* there are several other ways of representing our surroundings. In this sense, the system would be converted to not rely on a specific type of representation, but instead to transmit, store and organize raw data. This process is, conceptually, much simpler than may be perceived at first sight. In fact, the only node that explicitly relies on occupancy grids is the *Map Fusion* node. If every other node were converted to deal with arbitrary data, only the *Map Fusion* node would have to be deeply rebuilt in order to cope with the change. For instance, we could perform Multi-Robot SLAM based not on occupancy grids, but on Pose Graphs, as described in chapter 2. Given the system's architecture, we could retain the communication's efficiency, as well as the map tree method of computing a global representation, simply by replacing occupancy grids in the data structures with content-agnostic structures.

Bibliography

[1] Lars AA Andersson and Jonas Nygards. On multi-robot map fusion by inter-robot observations. In *FUSION'09. 12th International Conference on Information Fusion*, pages 1712–1721. IEEE, 2009.

[2] J-C Bermond, Luisa Gargano, Stephane Perennes, Adele A Rescigno, and Ugo Vaccaro. Efficient collective communication in optical networks. In *Automata, Languages and Programming*, pages 574–585. Springer, 1996.

[3] Rainer Kuemmerle; Giorgio Grisetti; Hauke Strasdat; Kurt Konolige; Wolfram Burgard;. g2o: A general framework for graph optimization, . URL `http://www.openslam.org/g2o.html`.

[4] Rainer Kuemmerle; Giorgio Grisetti; Hauke Strasdat; Kurt Konolige; Wolfram Burgard;. TORO - Tree-based netwORk Optimizer, . URL `http://www.openslam.org/toro.html`.

[5] Luca Carlone, M Kaouk Ng, Jingjing Du, Basilio Bona, and Marina Indri. Rao-Blackwellized particle filters multi robot SLAM with unknown initial correspondences and limited communication. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 243–249. IEEE, 2010.

[6] Stefano Carpin. Fast and accurate map merging for multi-robot systems. *Autonomous Robots*, 25(3):305–316, 2008.

[7] Andrea Censi, Luca Iocchi, and Giorgio Grisetti. Scan matching in the Hough domain. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2739–2744. IEEE, 2005.

[8] Alexander Cunningham, Manohar Paluri, and Frank Dellaert. DDF-SAM: Fully distributed SLAM using constrained factor graphs. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3025–3030. IEEE, 2010.

[9] P. Deutsch. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951 (Informational), May 1996. URL `http://www.ietf.org/rfc/rfc1951.txt`.

[10] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[11] J. F. Ferreira and J. Dias. *Probabilistic Approaches to Robotic Perception*. Springer Tracts in Advanced Robotics (STAR), 2014. ISBN 978-3-319-02006-8. doi: 10.1007/978-3-319-02006-8.

[12] Dieter Fox, Jonathan Ko, Kurt Konolige, Benson Limketkai, Dirk Schulz, and Benjamin Stewart. Distributed multirobot exploration and mapping. *Proceedings of the IEEE*, 94 (7):1325–1339, 2006.

[13] Udo Frese. A discussion of simultaneous localization and mapping. *Autonomous Robots*, 20(1):25–42, 2006.

[14] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with Rao-Blackwellized particle filters. *Robotics, IEEE Transactions on*, 23 (1):34–46, 2007.

[15] Giorgio Grisetti, Cyrill Stachniss, Slawomir Grzonka, and Wolfram Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Robotics: Science and Systems*, 2007.

[16] Andrew Howard. Multi-robot simultaneous localization and mapping using particle filters. *The International Journal of Robotics Research*, 25(12):1243–1256, 2006.

[17] David A Huffman. A method for the construction of minimum redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[18] Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[19] Kurt Konolige, Giorgio Grisetti, R Kummerle, Wolfram Burgard, Benson Limketkai, and Regis Vincent. Efficient sparse pose adjustment for 2D mapping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 22–29. IEEE, 2010.

[20] Rainer Kuemmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g2o: A general framework for graph optimization. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011.

[21] Rainer Kümmerle, Bastian Steder, Christian Dornhege, Michael Ruhnke, Giorgio Grisetti, Cyrill Stachniss, and Alexander Kleiner. On measuring the accuracy of SLAM algorithms. *Autonomous Robots*, 27(4):387–407, 2009.

[22] Maria Teresa Lazaro, Lina María Paz, Pedro Pinies, Jose A. Castellanos, and Giorgio Grisetti. Multi-Robot SLAM Using Condensed Measurements. In *Proc. of 2013 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2013)*. IEEE, 2013.

[23] Zhao Li and Ryad Chellali. Visual place recognition for multi-robots maps merging. In *International Symposium on Safety, Security, and Rescue Robotics*, pages 1–6. IEEE, 2012.

[24] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4(4):333–349, 1997.

[25] Gonçalo S. Martins, David Portugal, and Rui P. Rocha. A Comparison of General-Purpose FOSS Compression Techniques for Efficient Communication in Cooperative Multi-Robot Tasks. In *ICINCO*, 2014.

[26] João Alexandre Simões Martins. MRSLAM - Multi-Robot Simultaneous Localizations and Mapping. Master's thesis, Faculdade de Ciências e Tecnologia, Universidade de Coimbra, Portugal, 2013.

[27] Malika Meghjani and Gregory Dudek. Multi-robot exploration and rendezvous on graphs. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 5270–5276. IEEE, 2012.

[28] José Santos Martins Pereira. Implementation of a Mobile Ad Hoc Network communication protocol for Human-Robot Search and Rescue Teams. Master's thesis, Faculdade de Ciências e Tecnologia, Universidade de Coimbra, Portugal, 2013.

[29] David Portugal and Rui P Rocha. Distributed multi-robot patrol: A scalable and fault-tolerant framework. *Robotics and Autonomous Systems*, 61(12):1572–1587, 2013.

[30] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, 2009.

[31] R. P. Rocha. *Building Volumetric Maps with Cooperative Mobile Robots and Useful Information Sharing: a Distributed Control Approach Based on Entropy.* PhD thesis, University of Porto, Portugal, 2006.

[32] Rui P. Rocha, David Portugal, Micael S. Couceiro, Filipe Araújo, Paulo Menezes, and Jorge Lobo. The CHOPIN project: Cooperation between Human and rObotic teams in catastroPhic INcidents. In *Proc. of 11th IEEE Int. Symp. on Safety, Security, and Rescue Robotics (SSRR 2013)*. IEEE, 2013.

[33] Ohad Rodeh and Avi Teperman. zFS-a scalable distributed file system using object disksf. In *Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference on*, pages 207–218. IEEE, 2003.

[34] David Salomon. *A concise introduction to data compression.* Springer, 2007.

[35] João Machado Santos, David Portugal, and Rui P. Rocha. An Evaluation of 2D SLAM Techniques Available in Robot Operating System. In *Proc. of 11th IEEE Int. Symp. on Safety, Security, and Rescue Robotics (SSRR 2013)*. IEEE, 2013.

[36] OLSRD Team. olsrd: an adhoc wireless mesh routing algorithm daemon. URL `http://www.olsr.org/`.

[37] Sebastian Thrun. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, pages 1–35, 2002.

[38] Sebastian Thrun and Arno Bücken. Learning maps for indoor mobile robot navigation. Technical report, DTIC Document, 1996.

[39] Sebastian Thrun and Michael Montemerlo. The graph SLAM algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, 25(5-6):403–429, 2006.

[40] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial intelligence*, 128(1):99–141, 2001.

[41] Xun S Zhou and Stergios I Roumeliotis. Multi-robot SLAM with unknown initial correspondence: The robot rendezvous case. In *International Conference on Intelligent Robots and Systems*, pages 1785–1792. IEEE, 2006.

[42] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.

[43] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *Information Theory, IEEE Transactions on*, 24(5):530–536, 1978.

Appendix A

Paper Accepted for Presentation at the ICINCO 2014 Conference

A Comparison of General-Purpose FOSS Compression Techniques for Efficient Communication in Cooperative Multi-Robot Tasks

Gonçalo S. Martins, David Portugal and Rui P. Rocha
Institute of Systems and Robotics
University of Coimbra
3030-290, Coimbra, Portugal
{*gmartins, davidbsp, rprocha*}*@isr.uc.pt*

Keywords: Compression Methods, Multi-Robot Systems, Efficient Information Sharing.

Abstract: The efficient sharing of information is a commonly overlooked problem in methods proposed for cooperative multi-robot tasks. However, in multi-robot scenarios, especially when the communication network's quality of service is less than desirable, either in bandwidth or reliability, efficient information exchange is a key aspect for the successful deployment of coordinated robotic teams with proper exchange of information.
Compression is a popular, well-studied solution for transmitting data through constrained communications channels, and many general-purpose solutions are available as free and open-source software (FOSS) projects. There are various benchmarking tools capable of comparing the performance of these techniques, but none that differentiate between them in the compression of the typical data exchanged among robots in a cooperative task. Thus, choosing a compression technique to be used in this context is still a challenge.
In this paper, the issue of efficiently communicating data among robots is addressed by comparing the performance of various compression techniques in a case study of multi-robot simultaneous localization and mapping (SLAM) scenarios using occupancy grids, a cooperative task usually requiring the exchange of large amounts of data.

1 INTRODUCTION

Cooperation among mobile robots almost always involves interaction via explicit communication, usually through the use of a wireless network. Commonly, this network is taken for granted and little care is taken in minimizing the amount of data that flows through it, namely to assist the robot's navigation though the environment.

However, in real-world applications, the navigational effort can be but a small part of the tasks that must be dealt with by a complete robotic system (Rocha et al., 2013). Therefore, it should operate as efficiently as possible. Additionally, in harsher scenarios, such as search and rescue operations, constrained connectivity can become an issue, and caution must be taken to avoid overloading the network. An efficient model of communication is also a key element of a scalable implementation: as the number of robots sharing the network increases, the amount of data that needs to be communicated does as well. Thus, greater care in preparing data for transmission is needed, so as to avoid burdening the network by transmitting redundant or unnecessary data.

In this paper, we analyze the data transmitted by a team of robots on a cooperative mission that includes mapping and navigation. With this purpose, we use a multi-robot simultaneous localization and mapping (SLAM) task (Lazaro et al., 2013) as a case study of the exchange of information among robots, though the ideas proposed herein can be generalized to other cooperative tasks, at different abstraction levels. In our case study, mobile robots are required to communicate occupancy grids (Elfes, 1989) among themselves, in order to obtain a global representation of the environment based on partial maps obtained locally by individual robots.

Occupancy grids are metric representations of the environment, being repetitive by nature (Elfes, 1989). In their simplest form, they consist of a matrix of cells that are commonly in one of three states: free, occupied or unknown. These can be seen as the result of a "thresholding" operation applied to a more complex occupancy grid, which is composed of cells that, instead of one of three values, contain a probability value or distribution (Rocha, 2006) of the occupancy

of the space they represent.

In larger environments, or at greater resolutions, these simpler grids are composed of large matrices filled with only three different values, often containing very long chains of repeated cells. Keeping this data in memory in this form is a sensible approach. The data is very easily accessible, with no computational overhead. However, transmitting it in this form is most likely a wasteful use of bandwidth.

Compression methods are widely used in the transmission and storage of bulky data, such as large numbers of small files, logs, sound and video. Compression is even being used by default in specific file systems, offering a possible solution for this problem. These exploit the data's inherent *compressibility* in order to represent it using fewer bits of data than originally.

In the following pages, various general-purpose, lossless compression techniques are analyzed and compared, in an effort to determine which, if any, is more suitable as a solution to the large bandwidth requirements of multi-robot systems. We will start by presenting a review of previous work in efficient communication between coordinated robots, followed by a short presentation of the various techniques being compared. We then present and discuss our results, and summarily conclude by taking an outlook into future work.

1.1 Related Work

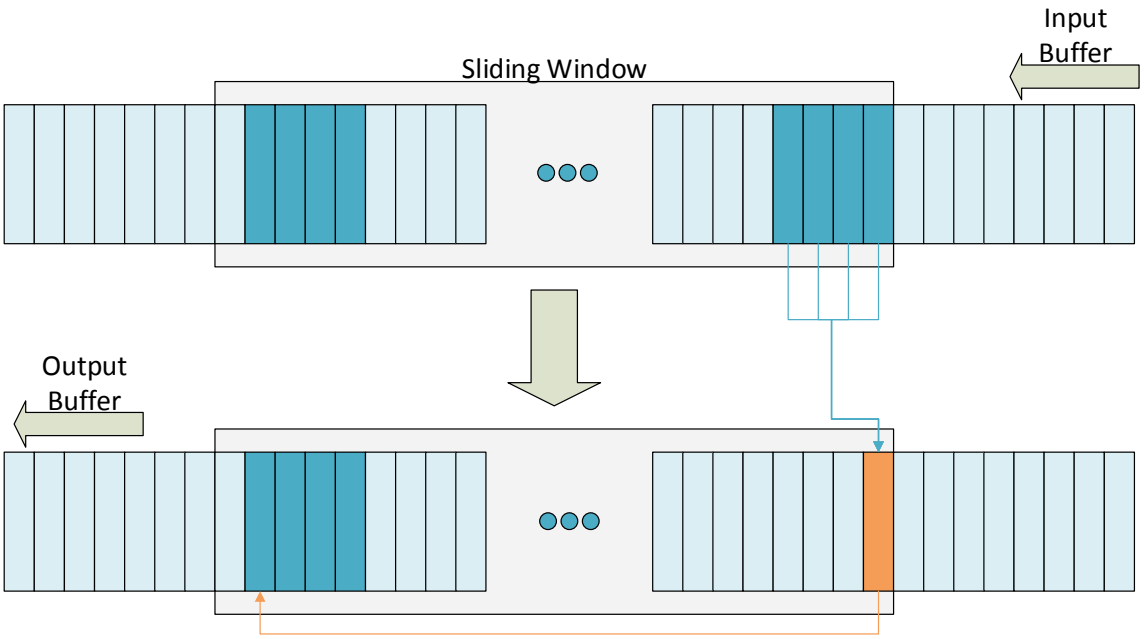
Data compression is a process through which we aim to represent a given piece of digital data using fewer bytes than the original data, and can be seen as a way of trading excess CPU time for reduced transmission and storage requirements. Compression methods are divided into two main groups: *lossless* methods, which make it possible to reconstruct the original data without error; and *lossy* methods, which make use of the way humans perceive signals to discard irrelevant data.

Lossy compression algorithms are commonly used in the compression of signals intended for human perception, such as image and sound. These techniques usually make use of the way we perceive signals to reduce their size (Salomon, 2007). For example, given that the human hearing's capability ranges from about 20Hz to about 20kHz, sound compression techniques can remove any signal components outside that frequency range. Although the compressed data should be significantly smaller than the original, humans hearing sound reconstructed from lossy compressed data should experience much the same. However, the original signal cannot be recovered.

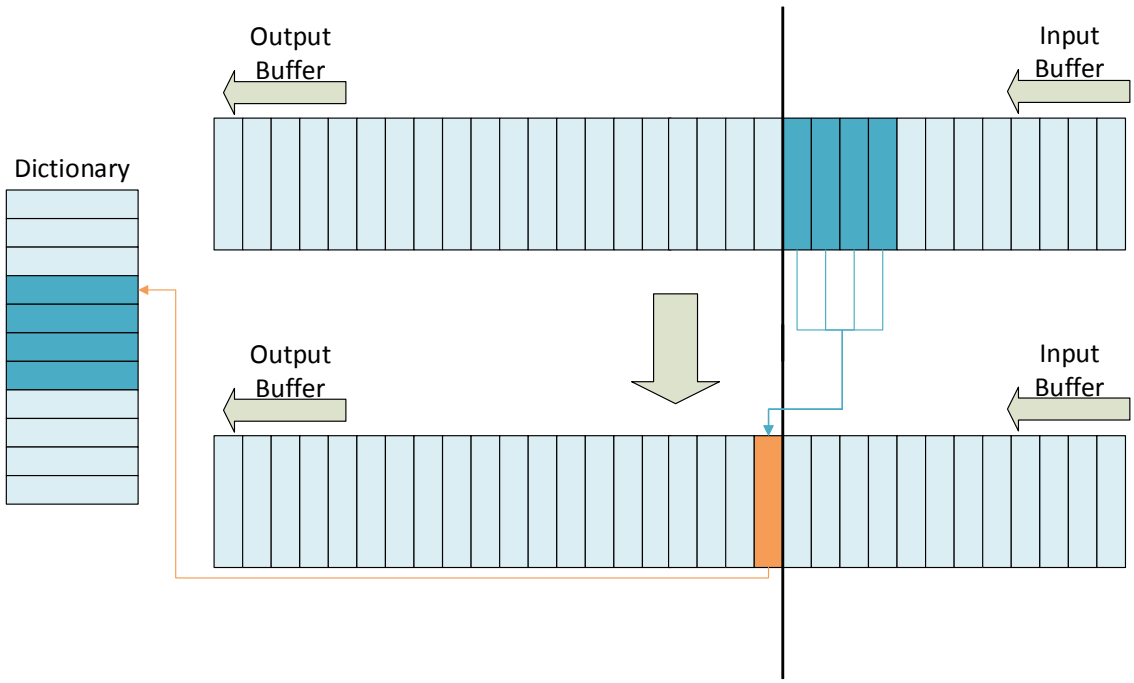
Lossless compression, on the other hand, compresses data in a way that it is later fully recoverable. In 1977 (Ziv and Lempel, 1977) and 1978 (Ziv and Lempel, 1978), Abraham Lempel and Jacob Ziv developed two closely related algorithms which were to become the basis for most of the lossless, general-purpose compression algorithms currently in use. LZ77 and LZ78, as their works were to become known, are methods of dictionary-based lossless compression. Summarily, the LZ77 and LZ78 algorithms keep a *dictionary* of byte chains encountered throughout the uncompressed data, and replace repetitions of those chains with *links* to entries in the dictionary, thus reducing the size of the data.

LZ77 compresses data by running a sliding window of a given fixed length over the input data, which is composed of variable-length sequences of bytes. For each input sequence, the algorithm looks for matches between the current sequence and a previous occurrence inside the sliding window. When a match is found, the repeated sequence is replaced by an offset and a length, which represent location of the previous occurrence in the sliding window, and the length of the repetition. For example, if the string "abc" existed twice in the window, the second occurrence would be replaced by an offset that pointed to the beginning of the string, and a length of three characters. This simple concept is the basis of dictionary coding. Furthermore, LZ77 has a way of dealing with very long repetitions, by specifying a length that is longer than the source string. This way, when decoding, the source string is copied multiple times into the output buffer, correctly rebuilding the repetition. For example, if the string "abc" exists somewhere in the sliding window, and the string "abcabc" exists somewhere after it, the second string would be replaced by an offset that pointed to the letter 'a' in the first string, and a length of six characters, instead of the length of three characters one might have expected, thus encoding the whole six-letter string into a single offset-length pair. Once all the data is encoded, decoding it consists of reversing the process, by replacing every offset-length pair in the coded data by their corresponding byte chains.

Despite technically being a dictionary coder, LZ77 does not explicitly build a dictionary. Instead, it relies on offset-length pairs to elliminate repetition. LZ78, on the other hand, does create an explicit dictionary. The algorithm attempts to find a match in the dictionary for every sequence that is taken from the input buffer. If a match is not found, it is added to the dictionary. Every match that is found is replaced with a structure analogous to the offset-length pair de-



(a) LZ77 operates by running a sliding window over the data. When a sequence in the input data is matched to data that is still inside the window, it is replaced with an offset-length pair that points to the previous instance of that data. In this figure, the dark blue segments were matched, and the second one is replaced with the orange, smaller segment, that points to the first copy of the matched segment.



(b) LZ78 operates by building an explicit dictionary. As the input data is consumed, the algorithm attempts to match each input sequence with an existing sequence in the dictionary. If the matching operation fails, the new data is added to the dictionary. This illustration shows the case where a match is found. In that case, the dark blue segments are matched to an entry in the dictionary, and replaced in the output buffer with the orange, shorter segment that points to the correct entry in the dictionary.

Figure 1: A simplified pictorial explanation of LZ77 and LZ78's operation.

scribed above, differing in the fact that now the offset represents an entry in the dictionary. The LZ78 dictionary is allowed to grow up to a given size, after which no additional entries are added, and input data that cannot be matched with any dictionary entries is output unmodified. Decoding LZ78-encoded data also consists of simply reversing the process, substituting each offset-length pair with the appropriate entry from the dictionary. The operation of these algorithms is illustrated in Fig. 1.

We have restricted our choice of algorithms to those based on Lempel and Ziv's work, for their focus on reducing redundancy by exploiting repetition, and for their *lossless* nature. It is important that the algorithms we are employing be fully lossless, *i.e.* that the compressed data can be used to reconstruct the original data, since we intend to generalize this technique to other types of data which may not tolerate any errors. For example, lossy image-based compression techniques, such as *JPEG*, could be used to reduce the size of an occupancy grid, processing it as an image. However, compression artifacts and other inaccuracies could lead to an erroneous representation of the environment, either by distorting its features or by hindering other aspects of the multi-robot mapping effort, such as occupancy grid image-based alignment and merging (Carpin, 2008).

Efficient inter-robot communication is not an area devoid of research. Other works, such as (Bermond et al., 1996), (Lazaro et al., 2013) and (Cunningham et al., 2010), have worked on a solution for this issue by creating new models of communication for robotic teams, *i.e.* by developing new ways of representing the data needed to accomplish the mission. Other research efforts focused on developing information utility metrics, *e.g.* by using information theory (Rocha, 2006), which the robot can use to avoid transmitting information with a utility measure below a certain threshold. We could find none, however, that applied compression to further increase their optimization gains. These techniques, while successful in their intended purpose, rely on modifications to the inner workings of their respective approaches. In our case, we intend to create an optimization solution that is more general, and that does not depend on modifications to the intricacies of the underlying techniques.

Finally, there are several examples[1] of compression benchmarks. However, we found none that focus on the algorithms' ability to optimize inter-robot communication. Their main focus is on comparing the techniques' performance on the compression and decompression of standard datasets, such as long sections of text, random numbers, *etc*. The need to test these techniques in the compression of specific, Robotics-related datasets, as well as the need to do so in a methodical, unbiased way, compelled us to create our own solution.

[1]Such as Squeeze Chart (`http://www.squeezechart.com/`) and Compression Ratings (`http://compressionratings.com/`).

1.2 Contributions

In this paper, we present a novel compression benchmarking tool and metric, as well as results and discussion of a series of experiments on the compression and decompression of occupancy grids, as a case study for the application of compression techniques in multi-robot coordinated tasks.

2 FOSS DATA COMPRESSION ALGORITHMS

As stated previously, occupancy grids, while a practical way of keeping an environment's representation in memory, are cumbersome as transmission objects. At the typical size of 1 byte per cell, an 800-by-800 cell grid (*e.g.* a representation of a somewhat small 8-by-8 meter environment at 100 cells per meter) occupies 640 kilobytes of memory. Depending on how fast an updated representation is generated, and how many robots take part in the mapping effort, this can lead to the transmission of prohibitively large amounts of data. If we update that same grid once every three seconds on each robot, each robot will generate an average of about 213KB/s. For a relatively small team of three robots, that equates to generating 640KB per second of data that needs to be transmitted. This simple calculation does not take into account the possibility of one of the robots exploring the environment further away from the others, causing the grids to expand, which would further enlarge the amount of repetitive data generated.

If we assume that each robot has to transmit its map to each of the team members, in a client-server networking model, each map update carries a bandwidth cost of $C = S \times (n-1)$, where C is the total cost, in bytes, S is the size of the map, in bytes, and n is the number of robots in the team. We can easily determine then that a regular 802.11g access point, operating at the typical average throughput of 22Mb (or 2.75MB) per second could support a team of 14 robots.

Given the redundancy that is naturally occurring in the data, there is great potential for optimization in the team's usage of bandwidth. Since data compression methods aim to remove redundancy from data,

and can be applied to any type of data, they seem adequate candidates for network optimization.

LZ77 and LZ78 inspired multiple general-purpose lossless compression algorithms, widely used today as Free and Open Source Software (FOSS) implementations. We have collected the ones that we believe are the most suitable as solutions to our problem, given their availability, use and features. We will summarily discuss them next.

DEFLATE[2], presented in (Deutsch, 1996), is the algorithm behind many widely used compressed file formats such as *zip* and *gzip*, compressed image formats such as *PNG*, and lossless compression libraries such as *zlib*, which will be the implementation through which DEFLATE will be tested. This algorithm combines the LZ77 algorithm with Huffman Coding (Huffman et al., 1952). The data is first compressed using LZ77, and later encoded into a Huffman tree. Being widely used, this technique was one of the very first to be considered as a possible solution to this problem.

LZMA[3], which stands for Lempel-Ziv-Markov Chain Algorithm, is used by the open-source compression tool *7-zip*. To test this algorithm, we have used the reference implementation distributed as the *LZMA SDK*. No extensive specification for this compressed format seems to exist, other than its reference implementation. LZMA combines the sliding dictionary approach of LZ77 with range encoding.

LZ4[4] is an LZ77-based algorithm focused on compression and decompression speed. It has been integrated into the Linux kernel and is used on the BSD-licensed implementation of ZFS (Rodeh and Teperman, 2003), OpenZFS, as well as other projects.

QuickLZ[5] is claimed to be "the world's fastest compression library". However, the benchmark results provided by its authors do not compare this technique to either LZ4 or LZMA, warranting it a place in our comparison.

Finally, Snappy[6], created by Google, is a lightweight compression library that aims at maximizing compression and decompression speed. As such, and unlike other techniques, it does not employ an entropy encoder like the Huffman Coding technique used in DEFLATE.

[2] *zlib* is available at `http://www.zlib.net/`

[3] The LZMA SDK used is available at `http://www.7-zip.org/sdk.html`

[4] LZ4 is available at `http://code.google.com/p/lz4/`

[5] QuickLZ is freely available for non-commercial purposes at `http://quicklz.com/`

[6] Snappy is available at `https://code.google.com/p/snappy/`.

3 BENCHMARKING METHODOLOGY

Part of the motivation behind this work consists of the fact that compression benchmarking tools usually focus on either looking for the fastest technique, or for the one that achieves the highest compression ratio, as defined by:

$$R = \frac{L_U}{L_C}, \tag{1}$$

where R is the compression ratio, L_U is the size of the uncompressed data, and L_C is the size of the compressed data, both usually measured in bytes.

When choosing among a collection of compression techniques, compression ratio is a metric of capital importance, since the better the ratio, the less information the robots have to send and receive to complete their goal. However, the techniques' compression and decompression speeds are also important; an extremely slow, frequent compression may jeopardize mission-critical computations. Thus, we cannot simply find the technique that maximizes one of these measures; there is a need to define a new, more suitable performance metric, in order to find an acceptable trade-off.

Therefore, we define:

$$E = \frac{R}{T_c + T_d}, \tag{2}$$

in which E is the technique's *temporal efficiency*. It is determined by dividing the compression ratio achieved by the technique, R, by the total time needed to compress and decompress the data, T_c and T_d, respectively. The purpose of this quantity is to provide an indication of how efficiently the technique at hand uses its computational time. The algorithm that achieves the highest temporal efficiency, while at the same time achieving acceptable compression ratio, is a strong candidate for integration in work that requires an efficient communication solution, provided that its absolute compression ratio is acceptable.

In order to test these techniques, the authors developed a benchmarking tool[7] that, given a number of compression techniques, runs them over occupancy grids generated by SLAM algorithms, outputting all the necessary data to a file. This tool allows us to both apply the techniques to the very specific type of data we wish to compress, as well as test them all in the same controlled environment. It was designed to be simple and easily extensible. As such, the addition

[7] The tool is publicly available under the BSD license at `https://github.com/gondsm/mrgs_compression_benchmark`.

of a new technique to the benchmark should be trivial for any programmer with basic experience.

To account for the randomness in program execution and interprocess interference inherent to modern computer operating systems, each algorithm was run over the data 100 times, so that we could extract results that were as isolated as possible from momentary phenomena, such as a processor usage peak, but that reflected the performance we could expect to obtain in real-world usage. Interprocess interference could have been eliminated by running test process in the highest priority. However, that does not constitute a real-world use case, and that methodology would provide results that could not be expected to occur during normal usage of the techniques. Results include the average and standard deviation of the compression and decompression times for each technique and dataset, as well as the compression ratio achieved for each case. These results can be seen textually in Table 1, or graphically in Figs. 3 and 4. Each technique was tested using their default, slowest and fastest modes, except for QuickLZ and Snappy, which only provide one mode of operation, and LZ4, which only provides a fast (default) and a slow, high compression mode.

All tests were run on an Intel Core i7 M620 CPU, with 8 GB of RAM, under Ubuntu Linux 12.04.

3.1 Datasets

In order to test the effectiveness of compression algorithms in treating typical occupancy grids, and given the intention of studying, at least to some degree, how each algorithm behaves depending on the dataset's size, five grids of different environments were chosen: Intel's Research Lab in Seattle; the ACES building, in Austin; MIT's CSAIL building and, finally, MIT's Killian Court, rendered in two different resolutions, so that differing sizes were obtained. The datasets are illustrated in Fig. 2. The occupancy grids we present were obtained from raw sensor logs using the *GMapping*[8] (Grisetti et al., 2007) SLAM algorithm, running on the ROS (Quigley et al., 2009) framework. The logs themselves have been collected using real hardware by teams working at the aforementioned environments, used for benchmarking SLAM techniques (Kümmerle et al., 2009), and later made publicly available.[9]

[8] A description of the version of GMapping can be found at `http://wiki.ros.org/slam_gmapping`.

[9] The raw log data used to create these maps is available at `http://kaspar.informatik.uni-freiburg.de/~slamEvaluation/datasets.php`.

4 RESULTS AND DISCUSSION

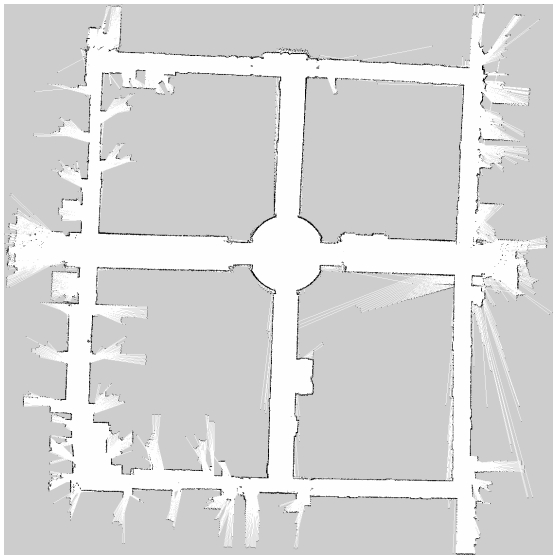
Fig. 3 and Table 1 illustrate the obtained results. In Fig. 3(a), we show the general trend in temporal efficiency for each technique as the size of the map grows. The general tendency is for efficiency to decrease as the data increases in size. However, in Fig. 3(b), we can observe that the compression ratio achieved tends to grow with the data's size. This effect can be attributed to the fact that, as the map grows, there are longer sequences of repetitive data, such as large open or unknown areas. It can also be explained, to a much smaller degree, by the fact that every compression technique adds control information to the compressed data, and that the size of this control data tends to be less significant as the uncompressed data grows. These figures lack error bars or other uncertainty representations due to the small dispersion of results, illustrated in Table 1 by the small values of standard deviation.

As expected, slower techniques generally achieve higher compression ratios. However, our results show that some techniques are indeed superior to others, in both temporal efficiency and compression ratio. LZ4 has shown both a higher temporal efficiency and compression ratio than that of QuickLZ and Snappy, making it a clearly superior technique, in this case. However, LZ4 HC, LZ4's slower mode of operation, is an inferior technique, both in temporal efficiency and ratio, when compared to LZMA and DEFLATE in the compression of larger datasets. Its temporal performance diminishes significantly with the growth in map dimensions, with an insufficient increase in compression ratio.

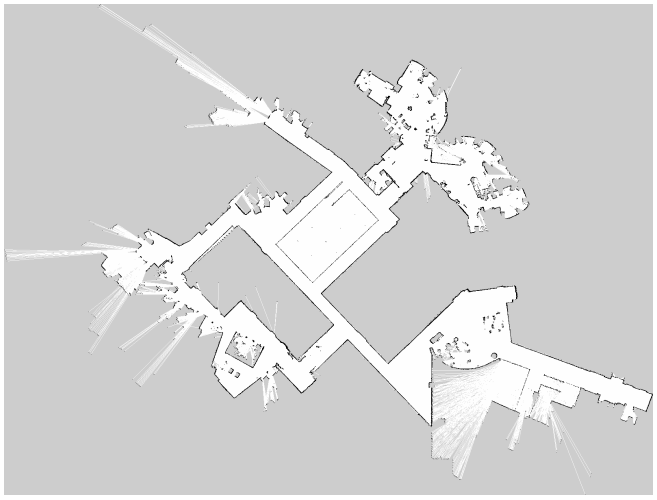
In applications where compression ratio is secondary relatively to speed, LZ4 is a strong candidate, and clearly the best among the techniques that were tested. It strongly leans towards speed and away from compression ratio, but offers acceptable ratios (around 15 for smaller maps, reaching 50 in larger ones) given its extremely fast operation. In other words, for applications which rely on transmitting occupancy grids, a very significant reduction of data flow can be achieved by employing this relatively low-footprint technique, which makes it suitable for use in real-time missions. As Fig. 3(a) shows, this technique is, by far, the most efficient at utilizing resources, achieving the best results in terms of temporal efficiency among the techniques that we have tested.

If further reduction in bandwidth is required, other techniques offer better ratios, at the expense of computational time. LZMA's fast mode offers one of the best ratios that we have observed, while still being ac-

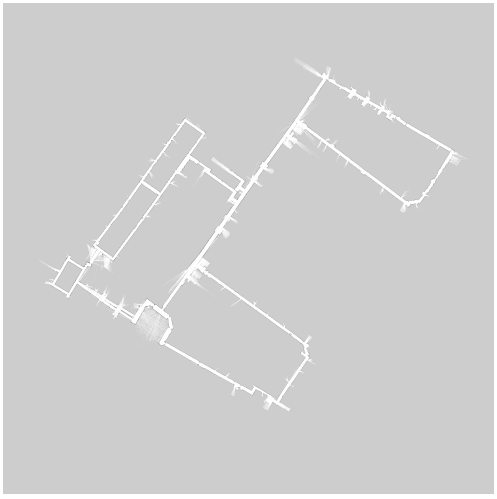
(a) Intel's Research Lab, measuring 753,078 bytes uncompressed.



(b) ACES Building, measuring 1,280,342 bytes uncompressed.

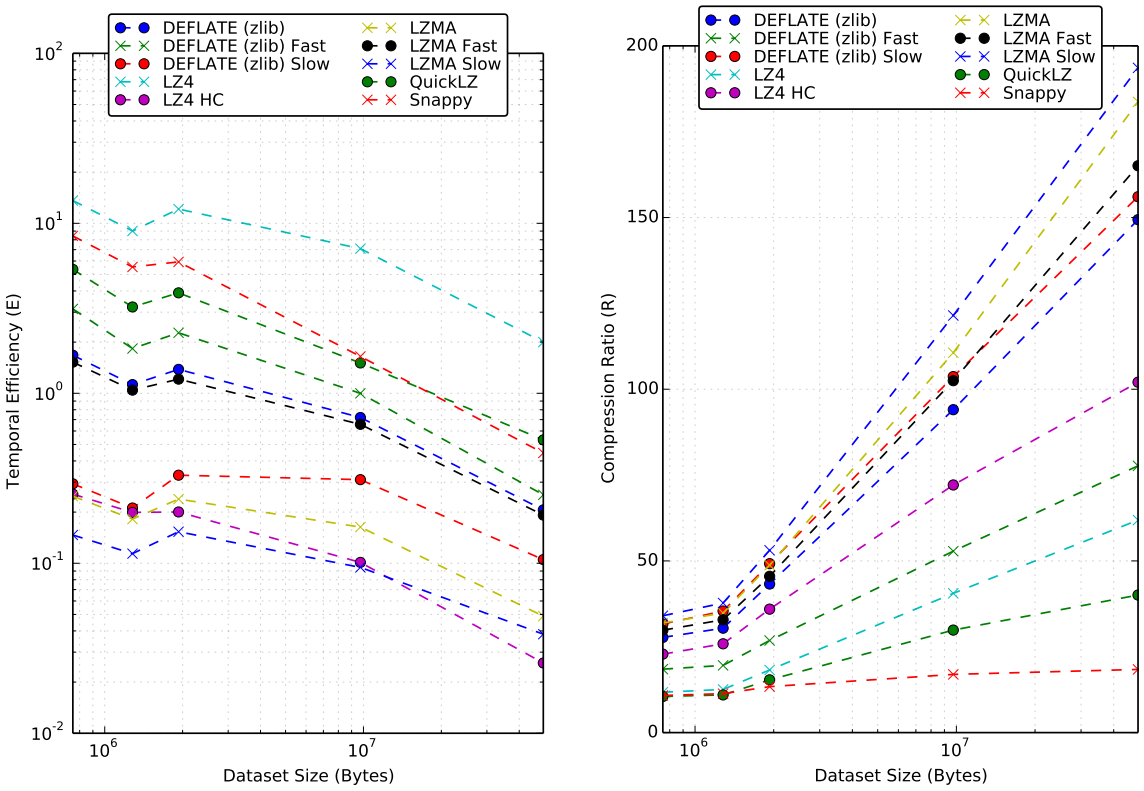


(c) MIT CSAIL Building, measuring 1,929,232 bytes uncompressed.



(d) MIT Killian Court, measuring 9,732,154 bytes (low resolution rendering) and 49,561,658 bytes (high resolution rendering) uncompressed.

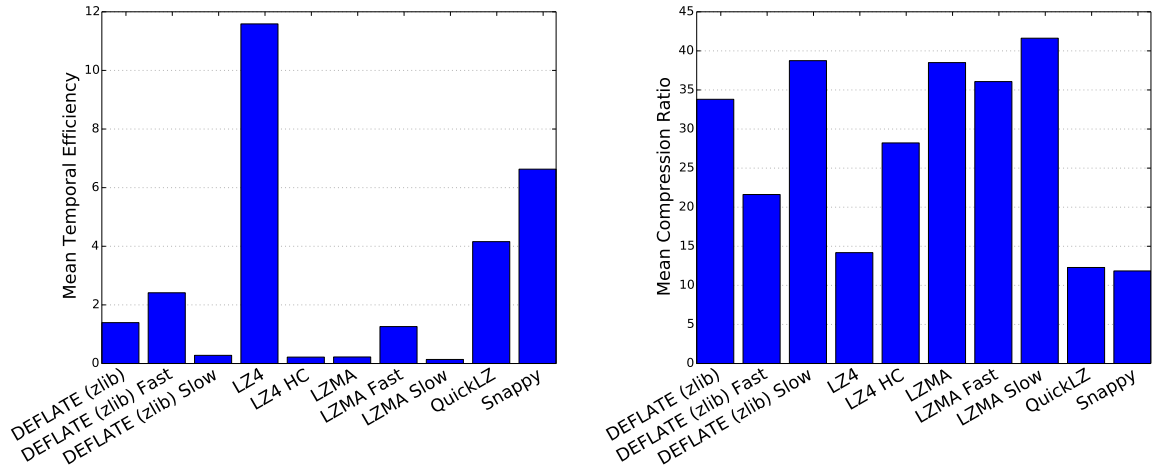
Figure 2: A rendering of each dataset used in our experiments. These were obtained by performing SLAM over logged sensor data.



(a) Temporal efficiency for each of the techniques and datasets.

(b) Compression ratio achieved by each technique for each dataset.

Figure 3: A graphical illustration of each technique's performance on all datasets. Each of the dotted lines connects data points for the same technique, so that trends become evident. Note the logarithmic scale.



(a) Mean temporal efficiency achieved by each technique for the three smaller datasets.

(b) Mean compression ratio achieved by each technique for the three smaller datasets.

Figure 4: A graphical illustration of each technique's performance on smaller datasets.

Table 1: Results obtained by processing the three smallest datasets 100 times with each technique. σ_c and σ_d correspond to the standard deviations of the compression and decompression times, respectively. $\bar{T}_c$ and $\bar{T}_d$ correspond to the average compression and decompression times, respectively.

(a) Raw results obtained for the Intel Research Lab dataset.

	Ratio	$\bar{T}_c$ (ms)	σ_c	$\bar{T}_d$ (ms)	σ_d
DEFLATE (zlib)	27.727	15.130	1.179	1.423	0.140
DEFLATE (zlib) Fast	18.474	4.503	0.736	1.388	0.241
DEFLATE (zlib) Slow	31.633	106.519	4.167	1.306	0.195
LZ4	11.741	0.452	0.064	0.410	0.064
LZ4 HC	22.850	89.312	3.721	0.241	0.028
LZMA	31.920	126.282	7.315	2.364	0.287
LZMA Fast	29.825	17.080	1.156	2.487	0.181
LZMA Slow	34.029	229.789	13.086	2.290	0.242
QuickLZ	10.519	1.222	0.153	0.742	0.069
Snappy	10.807	0.753	0.128	0.529	0.100

(b) Raw results obtained for the ACES Building dataset.

	Ratio	$\bar{T}_c$ (ms)	σ_c	$\bar{T}_d$ (ms)	σ_d
LZ4	12.5734	0.737898	0.118167	0.656754	0.0954742
LZ4 HC	25.8623	129.498	9.9717	0.381131	0.0711197
DEFLATE (zlib)	30.4135	24.7584	1.49278	2.27353	0.329637
DEFLATE (zlib) Fast	19.573	8.26037	1.6616	2.41425	0.444267
DEFLATE (zlib) Slow	35.4023	165.532	5.24992	1.91064	0.341901
LZMA	34.815	187.78	10.3723	3.60015	0.352538
LZMA Fast	32.8633	27.4526	1.42182	4.04572	0.499422
LZMA Slow	37.7465	327.663	11.5554	3.62876	0.431443
QuickLZ	10.9759	2.11142	0.243054	1.29769	0.127622
Snappy	11.3352	1.20599	0.12735	0.841902	0.108266

(c) Raw results obtained for the MIT CSAIL Building dataset.

	Ratio	$\bar{T}_c$ (ms)	σ_c	$\bar{T}_d$ (ms)	σ_d
DEFLATE (zlib)	43.274	27.927	1.203	3.370	0.172
DEFLATE (zlib) Fast	26.818	9.100	0.382	2.717	0.178
DEFLATE (zlib) Slow	49.205	146.207	1.760	3.027	0.069
LZ4	18.236	0.779	0.052	0.725	0.090
LZ4 HC	35.953	179.027	2.698	0.432	0.087
LZMA	48.763	200.306	11.911	4.142	0.302
LZMA Fast	45.522	33.280	0.448	4.304	0.105
LZMA Slow	53.088	342.213	8.815	4.019	0.261
QuickLZ	15.359	2.533	0.117	1.407	0.088
Snappy	13.387	1.250	0.059	1.008	0.048

Table 2: Results obtained by processing the two largest datasets 100 times with each technique. σ_c and σ_d correspond to the standard deviations of the compression and decompression times, respectively. $\bar{T}_c$ and $\bar{T}_d$ correspond to the average compression and decompression times, respectively.

(a) Raw results obtained for the smallest MIT Killian Court dataset.

	Ratio	$\bar{T}_c$ (ms)	σ_c	$\bar{T}_d$ (ms)	σ_d
LZ4	61.8855	15.0167	2.04569	15.9073	2.94617
LZ4 HC	102.05	3928.3	86.8325	12.4447	1.46376
DEFLATE (zlib)	149.383	614.592	24.0875	110.101	4.45021
DEFLATE (zlib) Fast	77.6953	242.236	21.732	65.1652	7.47955
DEFLATE (zlib) Slow	156.064	1375.26	50.3694	109.791	5.90444
LZMA	183.704	3685.39	150.48	75.4362	6.84588
LZMA Fast	165.082	776.456	20.6407	83.2567	6.06081
LZMA Slow	193.595	4995.91	386.814	63.4425	5.07394
QuickLZ	40.063	53.632	2.382	21.701	1.365
Snappy	18.400	17.986	0.799	23.335	1.080

(b) Raw results obtained for the largest MIT Killian Court dataset.

	Ratio	$\bar{T}_c$ (ms)	σ_c	$\bar{T}_d$ (ms)	σ_d
DEFLATE (zlib)	94.044	111.906	1.738	18.610	0.492
DEFLATE (zlib) Fast	52.831	41.207	3.083	11.647	0.846
DEFLATE (zlib) Slow	103.676	316.500	5.208	17.499	0.717
LZ4	40.553	2.920	0.198	2.797	0.406
LZ4 HC	72.116	710.753	32.165	1.992	0.147
LZMA	110.622	663.896	15.645	13.595	0.527
LZMA Fast	102.493	141.536	1.216	14.580	0.316
LZMA Slow	121.472	1269.680	158.155	14.937	1.938
QuickLZ	29.856	14.027	2.274	5.774	0.612
Snappy	16.951	5.192	0.751	5.101	0.492

ceptably fast. For the smallest dataset, this technique took, on average, about 15ms for compression, and achieved a ratio of 29.8. Depending on the application, 15ms of processor time per compression may be acceptable, given that this technique achieves a ratio that is almost three times as large as LZ4's, which achieved a ratio of 11.7, as is visible on Table 1(a).

In Fig. 4, we explore the case of the exchange of smaller maps, by averaging the temporal efficiency and ratio for each technique when operating over the smaller datasets. Smaller maps are commonly transmitted between robots at the beginning of the mission, when there is still little information about the environment. In these conditions, we note, as mentioned before, a generalized decrease in total compression ratio, and a narrowing of the gap between slow and fast techniques in terms of compression ratio: all techniques produce results within the same order of magnitude. However, the relationships between approaches in terms of temporal efficiency remain much the same. Thus, for smaller data, faster techniques appear to be a better option, since they achieve results that are comparable to those of their slower counterparts, at a much smaller cost in computational resources.

Larger maps, such as our largest examples, are very uncommonly transmitted during multi-robot missions, and hence unworthy of a closer analysis. Additionally, for these larger datasets, the multi-robot SLAM technique employed may make use of delta encoding techniques for transmission, transmitting only, for example, the updated sections of the map. In this case, we expect that the compression techniques applied to the map sections have the same performance as those applied to the smaller datasets in this test, since they will effectively be compressing smaller maps.

It is important to note that even the worse-performing techniques have achieved significant compression ratios, with a minimum ratio of about 10. Consequently, by using compression, we can reduce the total data communicated between robots during a mapping mission by at least a factor of 10, which shows the viability of compression as a solution for the problem of exchanging occupancy grids in a multi-robot system. In the context of the example we presented at the beginning of section 2, this equates to cutting our bandwidth requirements from 213KB/s per robot, to a much more affordable 21.3KB/s per robot, boosting our access point's theoretical capacity from 14 to 140 robots.

5 CONCLUSION

In this paper, we have explored the issue of communication optimization in the context of cooperative robotics, specifically the application of general-purpose lossless compression techniques to reduce the volume of data transmitted in cooperative robotic mapping missions. We have shown that compression is a viable option for the reduction of required network bandwidth in these scenarios, by defining and employing a new metric for the comparison of compression techniques, as well as the implementation of a new benchmarking tool. Moreover, important results about the performance of different lossless compression techniques in the context of multi-robot tasks were obtained, which can support an informed decision on which technique should be used in this context.

In the future, we plan to include and test one or various of these techniques in a real-world SLAM experiment, in order to gauge the impact of its use in the bandwidth needed to complete the mission. It would also be of interest to rerun these tests using datasets closer in size, so that we can more closely predict how the techniques' performance evolve with the size of the dataset. This may be a greater challenge than it appears since datasets differ in more ways than their size. A plausible way of working around this problem would be to expand the datasets using image processing techniques, such as nearest-neighbor interpolation, to isolate the dataset's size as the only variable characteristic between datasets.

It would also be interesting to investigate the influence of the application these techniques in the operation of Ad-Hoc networks, such as MANETs (Mobile Ad Hoc Networks), since they can be used in Search and Rescue operations (Rocha et al., 2013), a type of operation that requires great communication efficiency.

Additionally, the occupancy grids tested in this work, as stated before, correspond to the simplest form of occupancy grid: a simple matrix composed of only three different values. Given this, it would be very interesting to repeat these tests using the more complex form of the occupancy grids, as it would give us better insight into what we can expect from the application of these techniques in real-world scenarios.

Finally, given that occupancy grids are not, by any means, the only form of data exchanged during cooperative robotic missions, it would be interesting to explore the application of compression to other types of bandwidth-heavy data that robots need to exchange, such as the more complex occupancy grids described in (Ferreira et al., 2012), possibly culminating in the

creation of a compression technique mainly intended for the optimization of robotic communication.

ACKNOWLEDGEMENTS

This work was supported by the CHOPIN research project (PTDC/EEA-CRO/119000/2010) and by the ISR-Institute of Systems and Robotics (project PEst-C/EEI/UI0048/2011), funded by the Portuguese science agency "Fundação para a Ciência e a Tecnologia" (FCT).

The authors would like to acknowledge Eurico Pedrosa, Nuno Lau and Artur Pereira (Pedrosa et al., 2013) for providing us with a software tool intended to adapt the raw sensor log files into a format readable by *ROS*.

REFERENCES

Bermond, J.-C., Gargano, L., Perennes, S., Rescigno, A. A., and Vaccaro, U. (1996). Efficient collective communication in optical networks. In *Automata, Languages and Programming*, pages 574–585. Springer.

Carpin, S. (2008). Fast and accurate map merging for multi-robot systems. *Autonomous Robots*, 25(3):305–316.

Cunningham, A., Paluri, M., and Dellaert, F. (2010). DDF-SAM: Fully distributed SLAM using constrained factor graphs. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3025–3030. IEEE.

Deutsch, P. (1996). DEFLATE Compressed Data Format Specification version 1.3. RFC 1951 (Informational).

Elfes, A. (1989). Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57.

Ferreira, J. F., Castelo-Branco, M., and Dias, J. (2012). A hierarchical Bayesian framework for multimodal active perception. *Adaptive Behavior*, 20(3):172–190.

Grisetti, G., Stachniss, C., and Burgard, W. (2007). Improved techniques for grid mapping with Rao-Blackwellized particle filters. *Robotics, IEEE Transactions on*, 23(1):34–46.

Huffman, D. A. et al. (1952). A method for the construction of minimum redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101.

Kümmerle, R., Steder, B., Dornhege, C., Ruhnke, M., Grisetti, G., Stachniss, C., and Kleiner, A. (2009). On measuring the accuracy of SLAM algorithms. *Autonomous Robots*, 27(4):387–407.

Lazaro, M. T., Paz, L. M., Pinies, P., Castellanos, J. A., and Grisetti, G. (2013). Multi-robot SLAM using condensed measurements. In *Proc. of 2013 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2013)*. IEEE.

Pedrosa, E., Lau, N., and Pereira, A. (2013). Online SLAM Based on a Fast Scan-Matching Algorithm. In *Progress in Artificial Intelligence*, pages 295–306. Springer.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3.

Rocha, R. P. (2006). *Building Volumetric Maps with Cooperative Mobile Robots and Useful Information Sharing: a Distributed Control Approach Based on Entropy*. PhD thesis, University of Porto, Portugal.

Rocha, R. P., Portugal, D., Couceiro, M., Araujo, F., Menezes, P., and Lobo, J. (2013). The CHOPIN project: Cooperation between Human and rObotic teams in catastroPhic INcidents. In *Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on*, pages 1–4. IEEE.

Rodeh, O. and Teperman, A. (2003). zFS-a scalable distributed file system using object disks. In *Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference on*, pages 207–218. IEEE.

Salomon, D. (2007). *A concise introduction to data compression*. Springer.

Ziv, J. and Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343.

Ziv, J. and Lempel, A. (1978). Compression of individual sequences via variable-rate coding. *Information Theory, IEEE Transactions on*, 24(5):530–536.