mrgs: A Multi-Robot SLAM Framework for ROS with Efficient Information Sharing

Gonçalo S. Martins, David Portugal, and Rui P. Rocha

Abstract The exploration of unknown environments is a particularly and intuitively detachable problem, allowing the division of robotic teams into smaller groups, or even into individuals, which explore different areas in the environment. While exploring, the team can discretely reassemble and build 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 presents `mrgs`, an open source framework for Multi-Robot SLAM. We propose a 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. The solution is validated through experiments conducted over real-world data and we analyze its performance in terms of scalability and communication efficiency.

Keywords SLAM · Multi-Robot · ROS · Open source software · Efficient information sharing

1 Introduction

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. Unlike humans, robots can be designed to resist harsh conditions, can be extremely precise in their measurements, can be made

G. S. Martins (✉)
Ingeniarius, Ltd., R. Coronel Veiga Simão, Edifício B CTCV, 3025-307 Coimbra, Portugal
e-mail: gondsm@ingeniarius.pt

D. Portugal · R. P. Rocha
Institute of Systems and Robotics, University of Coimbra, Pólo II, Coimbra, Portugal
e-mail: davidbsp@isr.uc.pt

R. P. Rocha
e-mail: rprocha@isr.uc.pt

A. Koubaa (ed.), *Robot Operating System (ROS)*, Studies in Computational Intelligence 895, https://doi.org/10.1007/978-3-030-45956-7_3

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.

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, a problem usually known as SLAM: Simultaneous Localization And Mapping.

In the presence of a large, intricate environment, such as an abandoned factory, or a cave, 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. Intuitively, then, a team of robots could have tremendous advantages over a single robot at mapping certain locations, particularly vast ones. This also constitutes 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 are able to communicate amongst themselves, so that they can coordinate their exploratory efforts. While exchanging information, efficiency is the key to scalability. 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.

Given our knowledge of previous work, our prime objective is to develop a 2D Multi-Robot SLAM approach. 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, for instance due to a link failure or robot withdrawal. 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. Hence, the main contribution of this work is a 2D Multi-Robot SLAM framework for ROS, referred to as `mrgs`, which provides an easy-to-use collection of packages, and an open source implementation that allows the inclusion, swapping and testing of different modules for Multi-Robot SLAM.

This chapter is organized as follows: we start by overviewing the state of the art on Multi-Robot SLAM and Information sharing, in Sect. 2; in Sect. 3, we present and describe the system developed in ROS for performing Multi-Robot SLAM, and provide instruction to run the system on ROS, in Sect. 4. We then validate the system experimentally in Sect. 5. Finally, in Sect. 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 Background

2.1 Single Robot SLAM

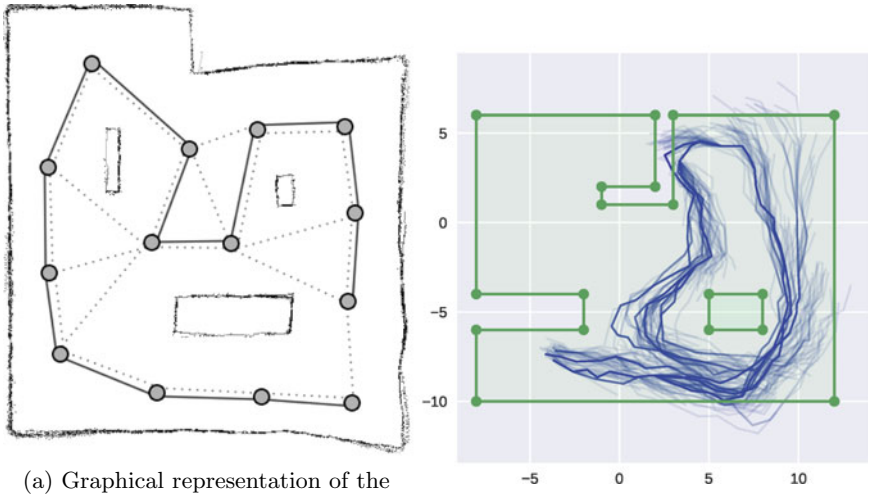
SLAM stands for Simultaneous Localization and Mapping. Essentially, this means that a robot performing SLAM is tasked with both creating a map of the environment and localizing itself in it.

Using its sensors, which usually include range and odometric sensors, the robot gathers data as it explores the world. Since all data gathered through sensors is plagued by noise and uncertainty, probabilistic solutions to the SLAM problem prevail over mathematically simpler approaches [1]. Odometry, in particular, produces errors that accumulate over time. In fact, with no assistance from other sensors, odometric errors can grow arbitrarily large [2].

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*, which 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, 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.

Classical solutions to the SLAM problem include Filtering techniques based on Bayes Filters [1], such as the Kalman Filter (KF), presented for the first time in [3], which uses Gaussian distributions for the posterior probability distribution; the Extended Kalman Filter, which is based on the linearization of the inherent non-linearities of both the motion and the sensor models [1], and Particle Filters (PFs), which 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 [4].

Recently, Graph-based SLAM techniques became very popular [5]. Unlike filtering techniques, in which data is discarded after being processed and incorporated into the filter's state, Graph-Based SLAM techniques save all gathered data, and a full notion of uncertainty, in the form of a graph, keeping a complete history of past poses



(a) Graphical representation of the constraint network used by Graph-SLAM. There can be various links from each node, and multiple links connecting any two nodes.

(b) Trajectories for all particles involved in a RBPF SLAM approach [11].

Fig. 1 Graph-based SLAM and Particle Filter SLAM concept

(see Fig. 1a). Graph-based SLAM performs a global optimization over all poses, thus growth in computation power, and techniques such as variable elimination [6] are key to allow the use of the technique in real time.

Graph Optimizers receive as input all the nodes and edges of the graph [7, 8], and return the optimized poses as output. Implementations of graph optimizers include the earlier TORO [7, 9]; SPA [10], an improved version of the algorithm originally proposed by Lu and Milios; and g2o [8], an open-source C++ framework for general graph optimization.

2.2 SLAM with Multiple Robots

Multi-Robot Systems (MRS) 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 very useful for monotonous, repetitive, complex, dangerous, large-scale and dividable problems [12]. Multi-Robot SLAM is 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 regards to SLAM, one of the biggest challenges is that of combining the information gathered by multiple robots.

2.2.1 Communicating Data

Communication is an essential part of cooperation, as robots have the need to share information in order to perform their cooperative tasks. Robots can either communicate using an existing communication infrastructure, e.g. a WiFi network, or a mobile, self-configuring, infrastructure-less dynamic network, namely Mobile Ad Hoc Networks (MANETs).

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 [13], 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.

2.2.2 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. In this section, we review and discuss some 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. [14–17], or by merging the rendered occupancy grids themselves [18].

In [14], 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 [19], in which virtual landmarks mounted on the robots are detected by cameras also mounted on the robots. These measurements are then used to estimate the rotation and translation needed to transform one robot's local coordinates into the other's. Once this 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 [15] 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 [16], a complete method for Graph-Based SLAM with multiple robots, which merges maps on the graph-level. As robots explore the environment, Graph-Based SLAM dictates that constraints be created between each pose at which a range scan is taken. 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). 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.

In [18], 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. The Hough Transform is an established method for detecting lines and other parametric-natured forms, such as circles or ellipses. The algorithm detailed in [18] uses the Hough Spectrum, as described in [20] 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, a metric named *acceptance index* is proposed to provide the solution which better fits the available data.

2.2.3 Multi-Robot SLAM

In [21], 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 [19], where the mutual detection and relative pose estimation problems are further explored.

The authors of [22] describe an interesting technique, also based on Rao-Blackwellized Particle Filters, which assumes each robot as an isolated entity, i.e. unlike in [21], 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 their respective filters.

In [23], the authors present an 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 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. 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 [16]. 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.

Recently, cloud-based approaches have also become popular for scalable and real-time Multi-Robot SLAM. In [24], the authors distribute the SLAM process to multiple computing hosts in a cluster, which enables 3D map building in parallel. They promote switchable messaging patterns to meet different transmission scenarios and eliminate the bottleneck from data sharing. Another cloud-based strategy is described in [25], where an efficient architecture is proposed to parallelize complex steps in Multi-Robot SLAM, via cloud computing nodes to free the robots from all the computational efforts. The system also decides between two alignment methods (data association and inter-robot observations), which is more appropriate for computing the coordinate transformation between the robot reference frames.

In this work we propose a technique-agnostic multi-robot SLAM package, `mrgs`. Our technique differs from other multi-robot SLAM packages by providing a framework that does not depend on the underlying SLAM technique, thus outperforming the state of the art in versatility. Unlike state-of-the-art techniques, `mrgs` relies on image-based techniques [18] to fuse maps, thus being able to seamlessly operate above single-robot techniques.

2.3 ROS: Robot Operating System

Algorithms and approaches 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 [26], 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.

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, 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 become a de facto standard in Robotics.

2.4 ROS Packages for SLAM

ROS software is distributed in packages, which generally contain code, messages types and other support files. We can find several ROS packages that implement SLAM algorithms.

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. Popular SLAM packages include GMapping[1] [27], Karto[2] and Hector[3] [28], which are compared in [29]; as well as the recent Cartographer graph-based SLAM approach implemented by a Google research team[4] [30].

The roscore,[5] 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. However, we envisage 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 systems always assume the existence of a single core that manages all communication between nodes. There is, however, interest in multimaster systems[6] [31].

In the context of this work, a workaround based on the *wifi_comm*[7] package is used. This package propagates 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

[1] http://wiki.ros.org/gmapping.
[2] http://wiki.ros.org/karto.
[3] http://wiki.ros.org/hector_mapping.
[4] http://wiki.ros.org/cartographer.
[5] http://wiki.ros.org/roscore.
[6] http://wiki.ros.org/sig/Multimaster.
[7] http://wiki.ros.org/wifi_comm.

of routing algorithms. By default, it uses the Optimized Link State Routing (OLSR) protocol [32].

2.5 *Efficient Communication in Multi-Robot Systems*

Cooperation among mobile robots usually 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 exploration through the environment.

However, in several specific scenarios, such as search and rescue operations, constrained connectivity can become an issue, and caution must be taken to avoid overloading the network. Additionally, in real-world applications, the exploration effort can be but a small part of the tasks that must be dealt with by a complete robotic system [33]. 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.

Previous works, such as [23, 34, 35], have worked on a solution for efficient inter-robot communication 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 [36], which the robot can use to avoid transmitting information with a utility measure below a certain threshold. 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 degree of 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, and their main objective is to represent data in as few bytes as possible, regardless of its contents. 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.

In our particular case, we intend to use a fast technique that efficiently utilizes resources, as long as it demonstrates an acceptable compression ratio. In a previous study, we have conducted a comparison of general-purpose FOSS compression techniques for the optimization of inter-robot communication [37], and we concluded that the best technique in terms of temporal efficiency is LZ4,[8] making it very suitable

[8] Available at https://github.com/lz4/lz4.

for real-time missions. Additionally, the compression ratio it achieves is appropriate for our purposes; as it leads to a minimum of 10x reduction in necessary bandwidth for map transmissions. Thus, we selected LZ4 as the most suitable technique for use in our Multi-Robot SLAM approach.

In the next section, we describe in detail the proposed `mrgs` system for Multi-Robot 2D SLAM.

3 `mrgs`: A ROS-Based Multi-Robot SLAM Solution

The Multi-Robot SLAM system implemented, `mrgs`,[9] is completely modular and its main functionality is divided into five components, distributed into four ROS packages, forming the main contribution of our work, as summarized in Fig. 2.

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. 3. In other words, this system can quickly enable any system running SLAM to run Multi-Robot SLAM.

The underlying individual SLAM technique benefits from laser scan readings acquired by robots, and possibly odometry estimation and/or other multimodal sources to assist the robot's local positioning (see for instance [29] for a list of

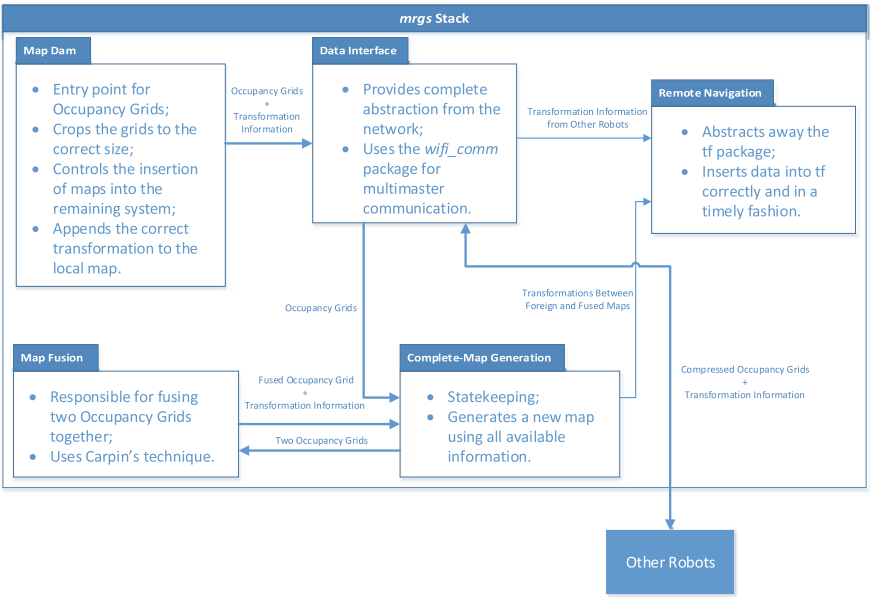


Fig. 2 An overview of our system's design and data flow. This system is replicated in each of the team members

[9]The `mrgs` framework is openly available at https://github.com/gondsm/mrgs.

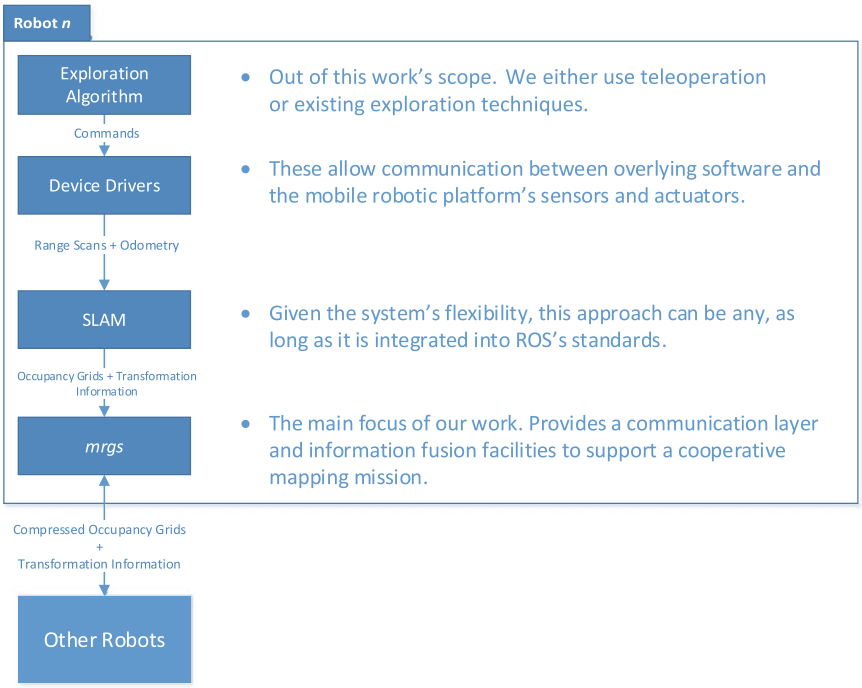


Fig. 3 A general overview of the complete system that runs on each robot. The focus of our work is on the `mrgs` block, a ROS metapackage containing our framework for Multi-Robot SLAM. `mrgs` enables robots to communicate with its peers and to build a global representation of the environment

popular SLAM techniques available in ROS), and provides as output an estimation of the robot's pose, as well as a local occupancy grid.

The local occupancy grids 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 priori 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 framework 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.

3.1 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.

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 framework, as 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.

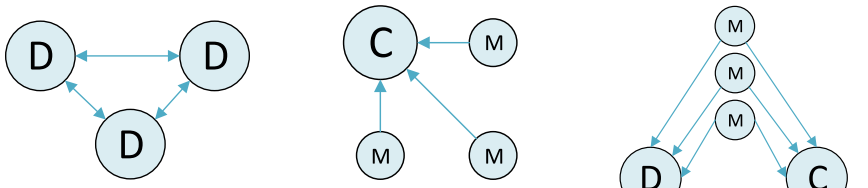
Mapping robots are the simplest of the three, they run a SLAM technique and use the Map Dam and Data Interface nodes to propagate the maps they have build. Central nodes, on the other hand, are assumed to be computationally powerful; they run the full framework, except for the `Map Dam` node, do not 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, e.g. a powerful desktop computer.

Finally, a third mode of operation can be achieved by combining the use of robots running 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.

3.2 Design and Implementation Principles

This framework was designed and built with efficiency, scalability and maintainability in mind. As such, all software strictly follows ROS's C++ guidelines,[10] and

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



(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.

Fig. 4 An illustration of the three modes of operation supported. "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

Software Engineering good practices. The system was purposefully segmented into concurrently-executing modules so as to explore the flexibility of multimaster 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, 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. In our implementation, code reuse is restricted to the usage of the reference implementation of Stefano Carpin's `mapmerge` [18], all remaining code is completely original.

3.3 Multi-Robot SLAM System Components

In this section, we provide details about the system components developed in this work, namely the `Data Interface`, `Map Fusion`, `Complete-Map Generation` and other auxiliary ROS nodes.

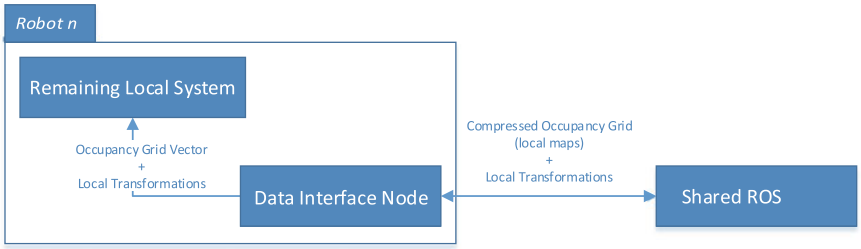


Fig. 5 The Data Interface node publishes into and subscribes to a topic that is shared among all robots. All information published into this topic is broadcast to all known robots in range in order to propagate the latest local map and transformation information

3.3.1 The Data Interface Node

The `Data Interface` node for ROS, 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. 5.

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 [13]. 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 multimaster communication approach of *wifi_comm* 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 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. As such, the system is expected to be robust to single node failure, as well as link failure.

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 [32] 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 [37].

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.

3.3.2 The Map Fusion Node

The `Map Fusion` 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 [18] at its core. As referred in Sect. 2.2.2, this algorithm fuses occupancy grids using an image processing method based on the Hough Spectrum to determine the candidate transformation that better fits the existing occupancy data. The aforementioned implementation is only capable of dealing with its own data representations, and only able to determine the transformations of the maps, but 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.

3.3.3 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

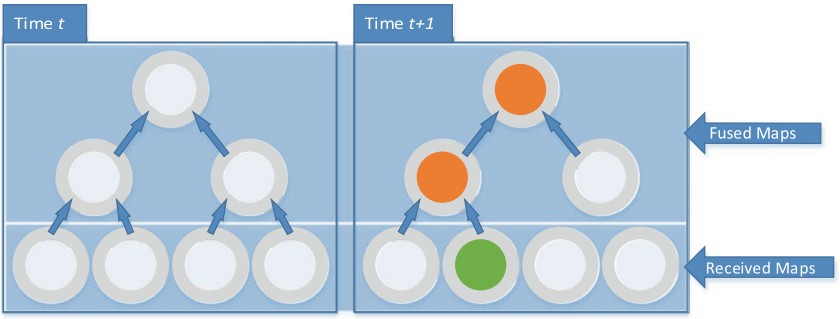


Fig. 6 Illustration of the tree-like structure for storing and updating maps. At a given instant *t*, the system receives a vector of maps, and iteratively merges them until it obtains a single representation. When a new map is received (green), only the maps that depend on it must be re-merged (orange). Each pair of arrows represents a merging operation

iteratively builds a tree-like data structure for storing occupancy grids, as represented in Fig. 6.

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 attempting 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. 6. 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. 7.

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.

Note, however, that the fused map, also known as global map, is not shared between the robots. It is only accessible to the robots that run the `Complete-Map`

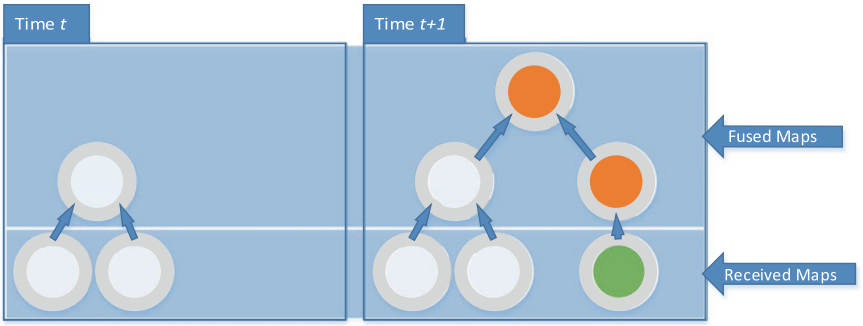


Fig. 7 When a new robot joins the mission, the tree-like structure is grown sideways to accommodate the new data. Iterative merging then proceeds as usual, regardless of whether the current number of maps is even or odd

`Generation` Node, i.e. robots that have interest on the data. Sharing the global map can easily be achieved by transmission of the map to other robots using the exact same process of sharing the individual partial maps. Yet, this is out of the scope of this work.

3.3.4 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 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 Installation and Usage

4.1 Installation

The `mrgs` system is meant to be installed as any cutting-edge ROS package: by cloning its repository into the current source space, installing its dependencies and building the workspace. Thus, we assume that the user has a working installation of ROS Melodic.[11] Earlier distributions should also be compatible, provided they are post-Fuerte, i.e. provided they use `catkin` and support the necessary dependencies.

Clone the package into your workspace by running:

```
git clone https://github.com/gondsm/mrgs
```

in your catkin source space (the `src` folder in your workspace).

The `mrgs` metapackage, included in the repository, lists all of the necessary dependencies that must be fulfilled for the system to work. Thus, installing the system's dependencies should be achieved by simply running:

```
rosdep install mrgs
```

which should be followed by ROS installing a number of packages through `apt`, as necessary. Once that procedure ends, you'll need to compile your workspace by changing the directory into its root and running:

```
catkin_make
```

4.2 Running the System

The `mrgs` package contains the necessary launch files to run the system.[12] As seen in Sect. 3, the system can be run in two main modes: centralized mode, which runs the full stack, and distributed mode, in which not all robots fuse information. These modes are achieved by mixing and matching two kinds of nodes: the central nodes, which receive information from all of the robots in the team and fuse it to obtain a global map; and mapper nodes, which run only the bare minimum system necessary to input information into the remainder of the team.

To run the system in centralized mode, only one robot in the team should run a central node. To achieve this, this main robot should use the launch file:

```
roslaunch mrgs central_node.launch
```

[11] http://wiki.ros.org/melodic/Installation/Ubuntu.

[12] It is important to note that, since `mrgs` is still under heavy development, it is possible that the general operating guidelines for the system change over time. As such, this text includes the most generic instructions necessary, with all detail being included in the project's repository. In case of conflict, the instructions on the repository should take precedence, as they will be significantly more up to date.

and, thus, run the full system. The remaining robots need to be launched using the launch file:

```
roslaunch mrgs mapper_node.launch
```

Which will ensure that they do not run the relatively CPU-heavy map merging operations. To run the system in distributed mode, all robots should run as central nodes. Thus, all robots will share and receive information from all other nodes, which results in a highly redundant system, better suited to hazardous situations.

5 Experiments

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 `mrgs` is a Multi-Robot SLAM solution, the most appropriate way of validating its performance is using several robots in an appropriate environment. 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,[13] 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 replacing the first blocks illustrated in Fig. 3 with a single ROS node tasked with playing back the recorded data. We designate the tests with data-gathering via recording without the `mrgs` system running in real time, as *offline* tests.

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 virtual machines and real, physical computers, as illustrated in Fig. 8, 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 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 to be run using three simultaneous machines on a consumer-grade laptop with only 8 GB 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

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

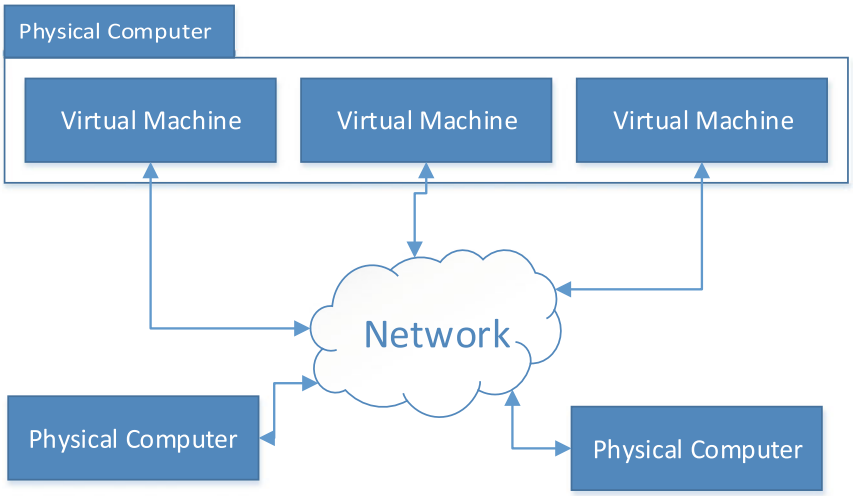


Fig. 8 An illustration of the setup used to process data. Virtual machines and real computers can be used as equals, in any combination, in a network routed by OLSR

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 Fig. 9; and on ISR's corridors, which is illustrated in Fig. 10. 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. 9. These online tests were used to determine if the results we were obtaining in the offline setup were correct, and to demonstrate the system's ability to operate in real-time. These tests also took place in the MRL arena and the corridors of the Institute, and achieved results that are identical to those of the offline methodology. Thus, we discuss only the results of the offline tests.

To further mimic the real-world usage of the package, the communication between the machines was compressed by the `data_interface` node.

5.1.1 Hardware

Real-world data was gathered using one or several Pioneer 3-DX mobile platforms, shown in Fig. 10, running the software framework described in Fig. 3. These are equipped with 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, each plat-

Fig. 9 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

Fig. 10 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

form had an Asus EEE 1005 mini-laptop mounted on it. The simulation and virtual machine-related operations were performed on a more powerful 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 mentioned in Sect. 2.4: *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 5 cm/cell.

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.

Figures 11, 12, 13 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,

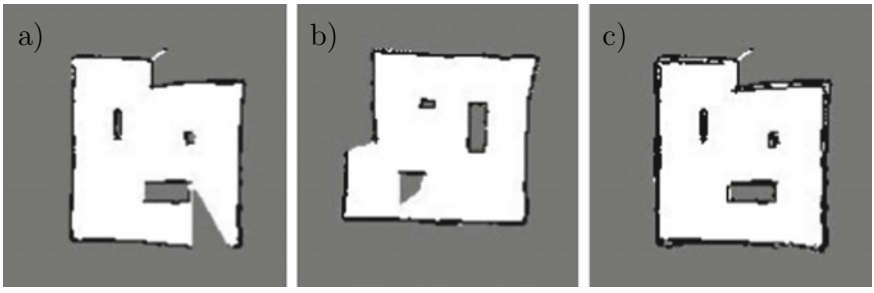


Fig. 11 An example of the results obtained using *GMapping* in the MRL Arena. **a** and **b** are the local maps of each of the robots, **c** is the result of the map fusion process

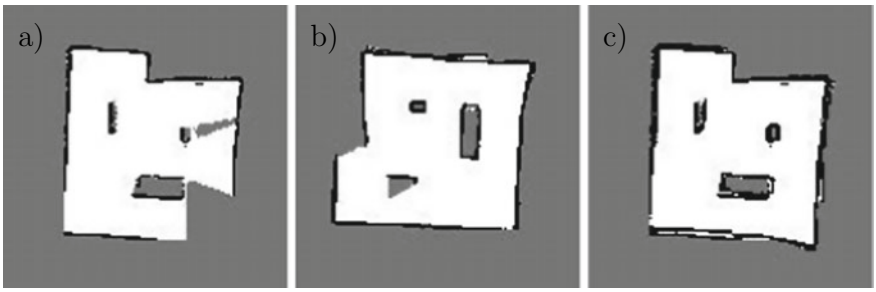


Fig. 12 An example of the results obtained using *Karto* in the MRL Arena. **a** and **b** are the local maps of each of the robots, **c** is the result of the map fusion process

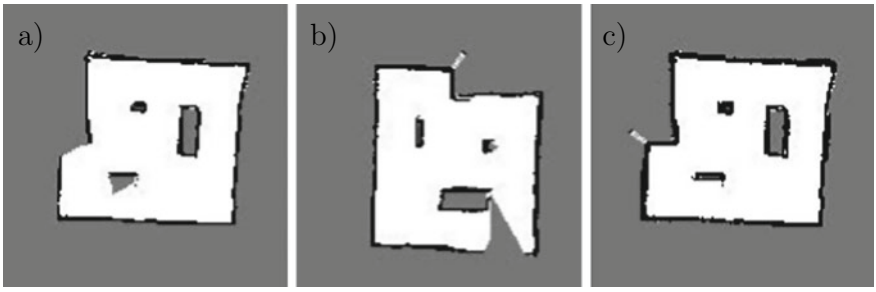


Fig. 13 An example of the results obtained using *Hector* in the MRL Arena. **a** and **b** are the local maps of each of the robots, **c** is the result of the map fusion process

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 Sect. 3.3.2. However, its performance revealed room for improvement, as illustrated in Fig. 14.

An unsuccessful fusion step can be catastrophic to the remaining effort, 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, and other factors.

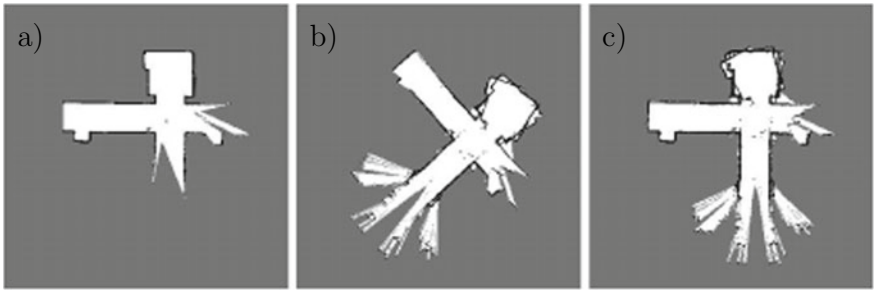


Fig. 14 An example of a failed three-way map fusion in the Institute's corridors. **a** is one of the robot's local map, and **b** is the result of the fusion of two other local maps. We can see that, while the maps were roughly rotated and translated correctly, there is a noticeable misalignment on one of the maps

5.2.2 Communication Efficiency

Table 1 displays the results obtained during the MRL Arena mission. N represents 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. Essentially, these show that the technique adopted to ensure efficiency in communication, compression using the LZ4 technique, is a viable option.

As postulated in Sect. 2.5, using compression on occupancy grids yields important data savings. In this case, using real occupancy grids, 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 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. It is important to note that these results are limited to occupancy grids, as mentioned before; it is not expected that the sole usage of LZ4 yield such dramatic improvements in the efficiency of the communication. For our use case, however, compression has been shown to be a very competent solution, which we will look to extend in the future.

In general, results show that the performance of our Multi-Robot SLAM solution is 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, and allowing smooth real-time operation of the system. A video of the system running with three robots in the MRL Arena has been made available.[14]

[14] https://drive.google.com/open?id=18jy5uftf5n-CqTDRJKNVOdtcd13BGPkw.

Table 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. All sizes are in bytes

	N	$\bar{R}$	L_t	L_s	D_s	T_p
(a) Results obtained during the MRL Arena mission						
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						
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

N Number of Processed Maps
$\bar{R}$ Average Compression Ratio
L_t Total Size of the Maps Received
L_s Total Amount of Data Sent into the Network by the Robot
D_s Amount of saved data
T_p Total map processing time (ms)

We do not assess the accuracy of the obtained maps in detail, as this heavily relies on the specific modules used in the `mrgs` framework, especially the `Map Fusion` node, which has been shown to have room for improvement. The interested reader may refer to [38] for an accuracy analysis conducted by a different research group, that made use of our system. Instead, we have focused on demonstrating that the proposed approach is able to correctly handle all the decoupled functionalities developed, namely: collection, compression, transmission and merging of the individual robots' maps, regardless of the underlying techniques used.

6 Conclusions and Future Work

In Sect. 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 on the number of robots available a priori, 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 should be able to deal with any 2D 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. This has allowed the use of the framework by other researchers in the community (see for example [38]).

As for future work, it would be useful to include a tool that graphically displays 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 improved. Being very well decoupled from the rest of the system, it would be very interesting to see it replaced with a more robust approach in the future, such as [39, 40], which could increase the success of the mapping effort itself. We could also test merging maps for robots running different individual 2D SLAM techniques, which would be possible, assuming that the output maps of the different techniques follow the same standard data principles and have the same map resolution.

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.

Moreover, 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 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 be capable of performing the processing of maps in a distributed fashion, in a manner inspired by computer clusters (e.g. see [41]).

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, a major improvement would be to deal with any kind of data. As we know, a representation of an environment is not limited to an occupancy grid. In this sense, the system could be upgraded to transmit, store and organize raw data. 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 Sect. 2.1 (see also [42]). 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.

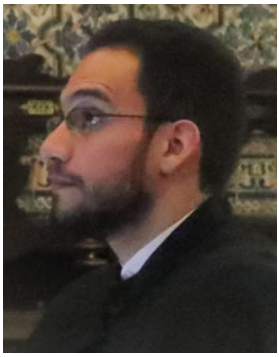
Acknowledgements We are sincerely thankful for the contributions on the free and open source frameworks adopted in this work, particularly: Giorgio Grisetti for his work on *RBPF SLAM*, Brian Gerkey and Vincent Rabaud for porting and maintaining *GMapping* for ROS, Stefan Kohlbrecher for the design and development of *Hector Mapping* for ROS, SRI International for *Karto SLAM*, and its maintainers for ROS over the years (Bhaskara Marthi, Michael Ferguson, Luc Bettaieb and Russell Toris), and to the overall ROS community. This work was supported by the **S**eguranças robó**T**icos co**OP**erativos (STOP) research project (ref. CENTRO-01-0247-FEDER-017562), co-funded by the "Agência Nacional de Inovação" within the Portugal2020 programme.

References

1. S. Thrun, Robotic mapping: a survey, in *Exploring Artificial Intelligence in the New Millennium*, vol. 1(1–35) (2002), p. 1
2. U. Frese, A discussion of simultaneous localization and mapping. Auton. Robots **20**(1), 25–42 (2006)
3. R.E. Kalman, A new approach to linear filtering and prediction problems. J. Basic Eng. **82**(1), 35–45 (1960)

4. S. Thrun, D. Fox, W. Burgard, F. Dellaert, Robust Monte Carlo localization for mobile robots. Artif. Intell. **128**(1–2), 99–141 (2001)
5. F. Lu, E. Milios, Globally consistent range scan alignment for environment mapping. Auton. Robots **4**(4), 333–349 (1997)
6. S. Thrun, M. Montemerlo, The graph SLAM algorithm with applications to large-scale mapping of urban structures. Int. J. Robot. Res. **25**(5–6), 403–429 (2006)
7. R. Kuemmerle, G. Grisetti, H. Strasdat, K. Konolige, W. Burgard, TORO - Tree-Based netwORk Optimizer (2008). http://www.openslam.org/toro.html
8. R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, W. Burgard, g2o: a general framework for graph optimization, in *2011 IEEE International Conference on Robotics and Automation* (IEEE, 2011), pp. 3607–3613. http://www.openslam.org/g2o.html
9. G. Grisetti, C. Stachniss, S. Grzonka, W. Burgard, A tree parameterization for efficiently computing maximum likelihood maps using gradient descent, in *Robotics: Science and Systems*, vol. 3 (2007), p. 9
10. K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, R. Vincent, Efficient sparse pose adjustment for 2D mapping, in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems* (IEEE, 2010), pp. 22–29
11. G. Vallicrosa, P. Ridao, H-slam: Rao-blackwellized particle filter SLAM using Hilbert maps. Sensors **18**(5), 1386 (2018)
12. D. Portugal, R.P. Rocha, Scalable, fault-tolerant and distributed multi-robot patrol in real world environments, in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems* (IEEE, 2013), pp. 4759–4764
13. M. Meghjani, G. Dudek, Multi-robot exploration and rendezvous on graphs, in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (IEEE, 2012), pp. 5270–5276
14. L.A. Andersson, J. Nygards, On multi-robot map fusion by inter-robot observations, in *2009 12th International Conference on Information Fusion* (IEEE, 2009), pp. 1712–1721
15. Z. Li, R. Chellali, Visual place recognition for multi-robots maps merging, in *2012 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)* (IEEE, 2012), pp. 1–6
16. D. Fox, J. Ko, K. Konolige, B. Limketkai, D. Schulz, B. Stewart, Distributed multirobot exploration and mapping. Proc. IEEE **94**(7), 1325–1339 (2006)
17. R. Natarajan, M.A. Gennert, Efficient factor graph fusion for multi-robot mapping and beyond, in *2018 21st International Conference on Information Fusion (FUSION)* (IEEE, 2018), pp. 1137–1145
18. S. Carpin, Fast and accurate map merging for multi-robot systems. Auton. Robots **25**(3), 305–316 (2008)
19. X.S. Zhou, S.I. Roumeliotis, Multi-robot SLAM with unknown initial correspondence: the robot rendezvous case, in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* (IEEE, 2006), pp. 1785–1792
20. A. Censi, L. Iocchi, G. Grisetti, Scan matching in the Hough domain, in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation* (IEEE, 2005), pp. 2739–2744
21. A. Howard, Multi-robot simultaneous localization and mapping using particle filters. Int. J. Robot. Res. **25**(12), 1243–1256 (2006)
22. L. Carlone, M.K. Ng, J. Du, B. Bona, M. Indri, Rao-Blackwellized particle filters multi robot SLAM with unknown initial correspondences and limited communication, in *2010 IEEE International Conference on Robotics and Automation* (IEEE, 2010), pp. 243–249
23. M.T. Lazaro, L.M. Paz, P. Pinies, J.A. Castellanos, G. Grisetti, Multi-robot SLAM using condensed measurements, in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems* (IEEE, 2013), pp. 1069–1076
24. P. Zhang, H. Wang, B. Ding, S. Shang, Cloud-based Framework for scalable and real-time multi-robot SLAM, in *2018 IEEE International Conference on Web Services (ICWS)* (IEEE, 2018), pp. 147–154
25. S.S. Ali, A. Hammad, A.S. Tag Eldien, Cloud-based map alignment strategies for multi-robot FastSLAM 2.0. Int. J. Distrib. Sens. Netw. **15**(3), 1550147719829329 (2019)

26. M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A.Y. Ng, ROS: an open-source robot operating system, in *ICRA Workshop on Open Source Software*, vol. 3, No. 3.2 (2009), p. 5
27. G. Grisetti, C. Stachniss, W. Burgard, Improved techniques for grid mapping with Rao-Blackwellized particle filters. IEEE Trans. Robot. **23**(1), 34 (2007)
28. S. Kohlbrecher, O. Von Stryk, J. Meyer, U. Klingauf, A flexible and scalable SLAM system with full 3D motion estimation, in *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics* (IEEE, 2011), pp. 155–160
29. J.M. Santos, D. Portugal, R.P. Rocha, An evaluation of 2D SLAM techniques available in robot operating system, in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics* (SSRR). Linköping, Sweden, Oct 21–26, (IEEE, 2013), pp. 1–6
30. W. Hess, D. Kohler, H. Rapp, D. Andor, Real-time loop closure in 2D LIDAR SLAM, in *2016 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2016), pp. 1271–1278
31. A. Tiderko, F. Hoeller, T. Röhling, The ROS multimaster extension for simplified deployment of multi-robot systems, in *Robot Operating System (ROS)* (Springer, Cham, 2016), pp. 629–650
32. A. Tonnesen, T. Lopatic, H. Gredler, B. Petrovitsch, A. Kaplan, S.O. Turke, *OLSRD: An Ad Hoc Wireless Mesh Routing Daemon* (2008). http://www.olsr.org/
33. R.P. Rocha, D. Portugal, M. Couceiro, F. Araújo, P. Menezes, J. Lobo, The CHOPIN project: cooperation between Human and rObotic teams in catastroPhic INcidents, in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)* (IEEE, 2013), pp. 1–4
34. J.C. Bermond, L. Gargano, S. Perennes, A.A. Rescigno, U. Vaccaro, Efficient collective communication in optical networks, in *International Colloquium on Automata, Languages, and Programming* (Springer, Berlin, Heidelberg, 1996), pp. 574–585
35. A. Cunningham, M. Paluri, F. Dellaert, DDF-SAM: fully distributed SLAM using constrained factor graphs, in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems* (IEEE, 2010), pp. 3025–3030
36. R. Rocha, J. Dias, A. Carvalho, Cooperative multi-robot systems: a study of vision-based 3-D mapping using information theory. Robot. Auton. Syst. **53**(3–4), 282–311 (2005)
37. G.S. Martins, D. Portugal, R.P. Rocha, A comparison of general-purpose FOSS compression techniques for efficient communication in cooperative multi-robot tasks, in *2014 11th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, vol. 2 (IEEE, 2014), pp. 136–147
38. M.A. Abdulgalil, M.M. Nasr, M.H. Elalfy, A. Khamis, F. Karray, Multi-robot SLAM: an overview and quantitative evaluation of MRGS ROS framework for MR-SLAM, in *International Conference on Robot Intelligence Technology and Applications* (Springer, Cham, 2018), pp. 165–183
39. N. Shaik, T. Liebig, C. Kirsch, H. Müller, Dynamic map update of non-static facility logistics environment with a multi-robot system, in *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)* (Springer, Cham, 2017), pp. 249–261
40. J.G. Mangelson, D. Dominic, R.M. Eustice, R. Vasudevan, Pairwise consistent measurement set maximization for robust multi-robot map merging, in *2018 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2018), pp. 2916–2923
41. D. Portugal, B.D. Gouveia, L. Marques, A distributed and multithreaded SLAM architecture for robotic clusters and wireless sensor networks, in *Cooperative Robots and Sensor Networks 2015* (Springer, Cham, 2015), pp. 121–141
42. I. Deutsch, M. Liu, R. Siegwart, A framework for multi-robot pose graph SLAM, in *2016 IEEE International Conference on Real-time Computing and Robotics (RCAR)* (IEEE, 2016), pp. 567–572



Gonçalo S. Martins received the M.Sc. degree in Electrical and Computer Engineering from the University of Coimbra (UC) in 2014, and a Ph.D. degree on Automation and Robotics also at the University of Coimbra in 2019, focusing on user-adaptive systems, machine learning and social robotics. Until recently, he was a researcher at the Institute of Systems and Robotics, where he worked on the H2020 GrowMeUp and Interreg EuroAGE projects. He is now working as a Senior Researcher at Ingeniarius, Ltd. His main research interests include Field and Service Robotics, Artificial Perception and Multi-Robot systems.

David Portugal completed his Ph.D. degree on Robotics and Multi-Agent Systems at the University of Coimbra in Portugal, in March 2014. His main areas of expertise are cooperative robotics, multi-agent systems, simultaneous localization and mapping, field robotics, human-robot interaction, sensor fusion, metaheuristics, and graph theory. After his Ph.D., he spent 4 years outside academia, and he is currently working as an Assistant Researcher at the University of Coimbra since 2019. He has been involved in several local and EU-funded research projects in Robotics and Ambient Assisted Living, such as CHOPIN, TIRAMISU, Social Robot, CogniWin, GrowMeUp, STOP, CORE and SEMFIRE. He has co-authored over 65 research articles included in international journals, conferences and scientific books.

Rui P. Rocha is an Assistant Professor (tenure position) in the Department of Electrical and Computer Engineering and a permanent researcher at the Institute of Systems and Robotics, both at the University of Coimbra, Portugal. His research interests include cooperative mobile robotics, cooperative perception, multi-robot systems, human-robot team cooperation, distributed control, ambient assisted living, social robots, and autonomous robots. Rocha received a Ph.D. in electrical and computer engineering from the University of Porto in May 2006. He has been involved in several FP6 and FP7 European funded projects developed in consortium for the past few years, and he is a senior member of IEEE and of the IEEE RAS Technical Committee on Multi-Robot Systems.