POLITECNICO DI TORINO

SCUOLA DI DOTTORATO

Dottorato in Meccatronica – XXVI ciclo

Tesi di Dottorato

Localization and Mapping for Service Robotics Applications

Stefano Rosa 178800

Tutore
prof. Basilio Bona

Coordinatore del corso di dottorato
prof. Giancarlo Genta

Aprile 2014

"I love deadlines. I like the whooshing sound they make as they fly by."

Douglas Adams

Contents

List of Tables

List of Figures

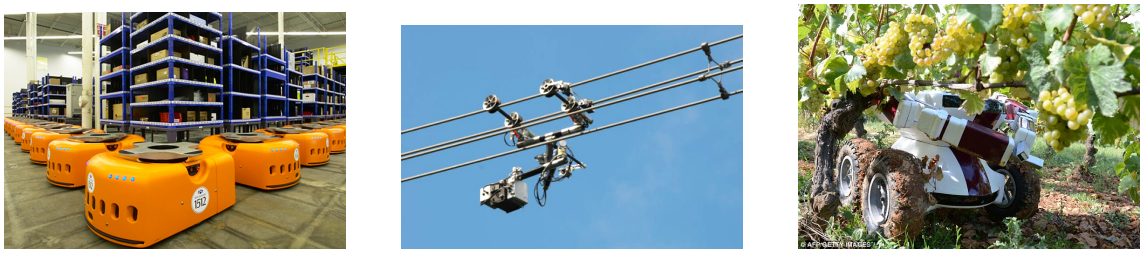
Chapter 1

Introduction

1.1 Motivation

Service robotics include all the robotic systems that are built to perform tasks in place of human users or in collaboration with them. Service robotics is defined as those robotics systems that assist people in their daily lives at work, in their houses, for leisure, and as part of assistance to the handicapped and elderly. In industrial robotics, the task is typically to automate tasks to achieve a homogenous quality of production or a high speed of execution. In contrast, service robotics' tasks are performed in spaces occupied by humans and typically in direct collaboration with people. Service robotics is normally divided into professional and personal services.

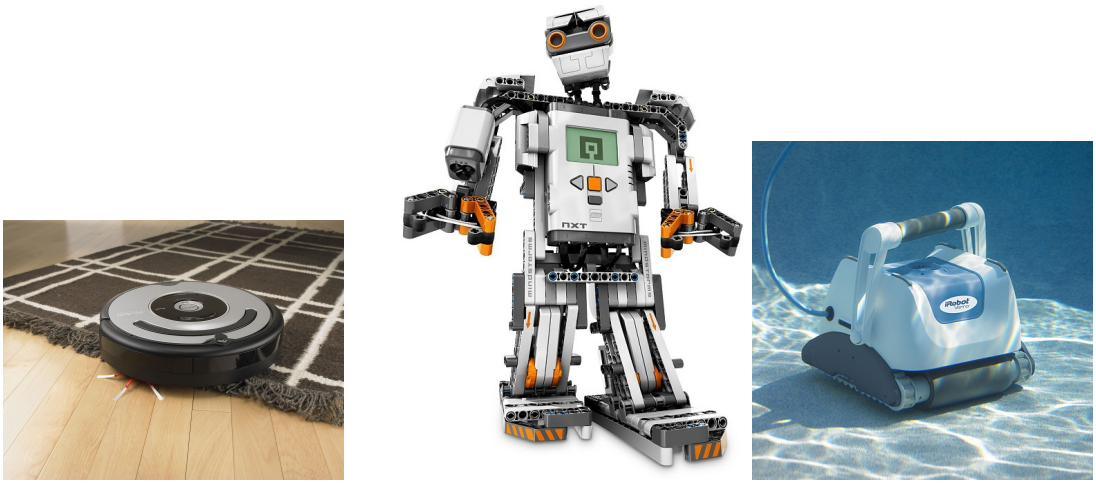
Professional service robotics includes agriculture, emergency response, pipelines, and the national infrastructure, forestry, transportation, professional cleaning, and various other disciplines. These systems typically augment people for execution of tasks in the workplace. According to the IFR/VDMA World Robotics, more than 110,000 professional robots are in use today and the market is growing rapidly every year [9].



(a) Kiva Systems (b) Expliner (c) Wall-Ye

Figure 1.1. Examples of professional service robots.

Personal service robots, on the other hand, are deployed for assistance to people in their daily lives in their homes, or as assistants to them for compensation for mental and physical limitations. By far, the largest group of personal service robots consists of domestic vacuum cleaners; over 6 million iRobot Roombas alone have been sold worldwide, and the market is growing 60% each year. In addition, a large number of robots have been deployed for leisure applications such as artificial pets (AIBO), dolls, etc. With more than 4 million units sold over the last 5 years, the market for such leisure robots is experiencing exponential growth and is expected to remain one of the most promising in robotics [9]. Some examples of popular commercial service robots are shown in Figure 1.2.



(a) Roomba™ (b) LEGO Mindstorms™ (c) Verro™

Figure 1.2. Examples of personal service robots.

Today, autonomous service robots are still far from being part of our everyday life. This is hard to understand if one considers the remarkable success of robots in automation industries. In controlled environments the robots' speed, accuracy and reliability by far exceed human capabilities.

There has been progress on challenging problems in highly unstructured environments, and several successful applications in more structured environment and tasks. These applications demonstrate that service robotics has value in solving real-world problems. Nonetheless, a number of important problems remain open. Finding solutions to these problems in the area of mobility will be necessary to achieve the level of autonomy and versatility required for the identified application areas.

Enabling safe autonomous navigation in unstructured environments with static obstacles, human-driven vehicles, pedestrians, and animals will require significant investments in component technologies.

Under the definition "autonomous navigation" we group all the techniques and algorithms which are necessary for a mobile robotic platform in order to accomplish these fundamental tasks:

- localization inside a known environment
- map building and simultaneous localization inside an unknown environment (SLAM)
- obstacle avoidance and path planning

Robot localization denotes the problem of estimating the position of a robot inside a known map of the environment. Map building is the process of estimating a map of an unknown environment given the positions of the robot and a set of measurements. Initially these two problems were studied independently; later they have been studied as two related issues. The resulting problem has been called simultaneous localization and mapping (SLAM). The third task is related to the ability of a robot to autonomously move from one position to another, while avoiding static and moving obstacles.

One of the key fields of development that has been indicated for Robotics in [9] is Logistics. In this case Logistics is mainly intended as the autonomous transport of goods and people. However there are other possible of developments of Robotics in Logistics and one of them can be found in the three-years (2008-2011) project *MACP4Log* - Mobile, autonomous and cooperating robotic platforms for supervision and monitoring of large logistic surfaces (co-funded by Regione Piemonte), which provided the foundation for the first part of this Phd work. MACP4Log is aimed at the study and development of a prototype of a mobile robotic platform, with on-board vision systems and sensors, integrating a flexible wireless communication solution, able to move autonomously in large logistic spaces, and to communicate with a supervisor and other similar platforms to achieve a coordinated action to carry out specific tasks. Logistic spaces are large areas where logistic societies or other transport enterprises receive, store and distribute large quantities of goods, mainly bulky ones, as containers, cars and other similar items. Other logistic spaces may include car-rental parkings, intermodal rail nodes, etc.

The main tasks to be performed by the robot team address the principal issues of potential logistic users and are given by: building and updating maps of both indoor and outdoor logistic spaces, performing programmed and pro-active surveillance, locating on the map specific items, as container, cars, etc., that can be marked by proper tags (e.g. license plates, container numbers, RFID) or unmarked, and should be distinguished by color, shape and/or other physical characteristics, achieving a full coordination of the team, always securing the wireless connectivity.

The mobile platform must be able to: autonomously move within substantially flat and uniform indoor and outdoor spaces, partially structured, with or without

a given initial approximated map describing the environment; self-locate within the given/constructed map, by means of on-board sensors and cameras (passive or active markers may be present in the logistic surface) connect to the supervisor and upload heterogeneous data (video stream from cameras, data about positioning, logs, alarms, etc.); connect to other similar mobile platforms operating in the logistic scenario to build a cooperative approach to the supervision goal; autonomously contribute to a cooperative task in a supervised team management.

The second part of the work presented in this thesis was motivated and driven by a joint collaboration between Politecnico di Torino and Telecom Italia on the topic of service cloud robotic applications for industrial applications, in particular for data center monitoring and surveillance. This work follows the main guidelines of the previous one, but puts the focus on single-robot algorithms and low-cost robotic platforms.

Cloud robotics is an emerging field of robotics rooted in cloud computing, cloud storage, and other Internet technologies centered around the benefits of converged infrastructure and shared services. It allows robots to benefit from the powerful computational, storage, and communications resources of modern data centers. In addition, it removes overheads for maintenance and updates, and reduces dependence on custom middleware.

Cloud robotics allows robots to take advantage of the rapid increase in data transfer rates to offload tasks without hard real time requirements. This is of particular interest for mobile robots, where on-board computation entails additional power requirements which may reduce operating duration and constrain robot mobility as well as increase costs.

Broader definitions of Cloud Robotics may also include other Internet-related aspects of robotics, such as trends towards the online sharing of open source hardware and software, crowd-sourcing of robotics funding, telepresence, and human-based computation. Other definitions stress the links between robotics and related emerging fields such as the Internet of Things, Web of Things, robot app stores, sensor networks, Big data, and others.

In this context, the work presented in this part of the thesis is devoted to two different problems: first, we work on the enhanchement of robustness for single robot localization in indoor environments. This need stems from two facts: first, it is important for a real application to be reliable and be able to work in different scenarios; second, since one of the objectives of cloud robotics is to help the diffusion of affordable domestic robots, we have to expect that these robots will contain low-cost and low-accuracy sensors, compared to usual research or industrial platforms.

The third part of this thesis was motivated by the work done with Istituto Italiano di Tecnologia on the topic of space robotics applications. The aim was the development of the basic components for human robotics and related technologies for space applications and to foster all related spin-offs for terrestrial applications. In

particular the Phd work reported in this thesis was done in collaboration with Thales Alenia Space on astronaut detection and tracking for extravehicular activities where semi-autonomous robots need to follow and assist the astronauts. The work was partially done under the *STEPS* (Systems and Technologies for Space Exploration) Space Project. This Space project, which lasted for three years and was co-financed by Regione Piemonte, aimed to study new technologies that may be used in the robotic or human space exploration, with particular attention to Moon and Mars exploration.

1.2 Thesis outline

The first problem that we address in Chapter 2 is the problem of robot localization. We first show how a multi-robot localization approach can speed up the process as well as introducing robustness to failures. At the same time we show the possibility to introduce some sort of insight into the algorithm.

We then focus on increasing the robustness of single robot localization, by taking advantage of sensor fusion and by exploiting visual markers embedded in the environment. We then show how the approach can be extended from ad-hoc markers to an arbitrary set of pre-existing objects already present in the environment.

In Chapter 3 we study the more complicated problem of Simultaneous Localization and Mapping. In particular we focus on a promising technique called GraphSLAM and we show the implementation of a front-end for this optimization technique in order to create a real online SLAM approach.

The last part of this thesis is devoted to another part of service robotics which is not directly related to navigation: Human Robot Interaction (HRI). While this field of robotics is not strictly related to autonomous navigation, some part of it can be, such as the case of user following and object tracking. In Chapter 4 we discuss the implementation of an adaptive people tracking algorithm taylored for user tracking and following. We also show how this algorithm can easily be extended to the case of generic object tracking for human-robot cooperation.

Chapter 2

Localization

2.1 Introduction

The problem of robot localization consists of answering the question *Where am I*? from a robot's point of view. This means the robot has to estimate its location relative to the environment. When we talk about location, pose, or position we mean the x and y coordinates and heading direction of a robot in a global coordinate system. The localization problem is an important problem. It is a key component in many successful autonomous robot systems. If a robot does not know where it is relative to the environment, it is difficult for it to decide what to do. The robot will most likely need to have at least some idea of where it is to be able to operate and act successfully [26].

The general localization problem is composed by two increasingly difficult sub-problems. In the *position tracking* problem the robot knows its initial location in the environment. The goal of the localization is to keep track of the position while the robot is navigating through the environment. Methods that address this problem are called local techniques [40].

In the *global localization*, instead, problem the robot does not know its initial position and it has to localize itself from scratch. It hereby possibly needs to be able to deal with multiple hypotheses about its location. Methods that solve this problem are called global techniques [40]. An even harder problem is the kidnapped robot problem. The robot is already correctly localized, but it is transferred (or "kidnapped"), to another location without the robot being aware of this. This can happen for example in the case of failures or if the robot is physically moved by an operator. The problem for the robot is to detect that it has been kidnapped and to find out what its new location is [40]. A factor that complicates each of these problems is the dynamics of the environment in which the robot is moving. Most research on localization has been focused on static environments. This means that

the robot is the only moving object in the environment. Obviously this is not the case in the real world. Dynamic environments contain other moving objects and in these environments localization is significantly more difficult, since these other objects might confuse the robot about its location by corrupting the information used for localization.
In determining its location, a robot has access to two kinds of information. First, it has a-priori information gathered by the robot itself or supplied by an external source in an initialization phase. Second, the robot gets information about the environment through every observation and action it makes during navigation. In general, the a-priori information supplied to the robot describes the environment where the robot is driving around. It specifies certain features that are time-invariant and thus can be used to determine a location. The a-priori information can come in different flavors.

In most localization approaches the robot has access to a map representation of the environment. Such map can be geometric or topological [81]. Geometric maps describe the environment in metric terms, much like normal road maps. Topological maps describe the environment in terms of characteristic features at specific locations and ways to get from one location to another. A map can be given by an external source in an initialization phase. Alternatively, the robot can learn a map of the environment while it is navigating through it, which is known as *Simultaneous Localization and Mapping* (Chapter 3).

The information which is usually available to the robot in order to perform localization is divided into two categories.

2.1.1 Relative measurements

Acquiring relative measurements is also referred to as dead reckoning, which has been used for a long time, ever since people started traveling around. Originally, this is the process of estimating the position of an airplane or a ship, only based on the speed and direction of travel and the time that passed since the last known position. Since the position estimates are based on earlier positions, the error in the estimates increases over time. In mobile robotic applications, relative position measurements are either acquired by wheel odometry or by inertial measurement units (IMU).

2.1.2 Navigational information

The second type of information to which a robot has access is navigational information, i.e., information that the robot gathers from its sensors while navigating through the environment. A robot typically performs two alternating types of actions when navigating: it drives around or acts in the environment on one hand,

and it senses the environment on the other. These two types of actions give rise to two different kinds of position information.
To be able to move around in an environment a robotic vehicle has a guidance or driving system [27]. A guidance system can consist of wheels, tracks or legs, in principle anything that makes the vehicle move around. These components are called actuators. Obviously, the guidance system plays an important role in the physical position of a robot. The guidance system directly changes the location of the vehicle. Without a guidance system the robot is not driving around, which makes localizing itself a lot easier.

Assuming that a robot does have a guidance system, the way the guidance system changes the location contains valuable information in estimating the location. Knowing the effects of actions executed by the driving system gives a direct indication of the location of the vehicle after execution of these actions. By monitoring what the driving system actually does using sensors, the displacement of the robot vehicle can be estimated. This results in relative position measurements, or also sometimes referred to as proprioceptive measurements. Relative position measurements are measurements that are made by looking at the robot itself only. No external information is used and these measurements can therefore only supply information relative to the point where the measurements were started.
The robot senses the environment by means of its sensors. These sensors give momentary situation information, called observations or measurements. This information describes things about the environment of the robot at a certain moment. Observations made from the environment provide information about the location of the robot that is independent of any previous location estimates. They provide absolute position measurements, also sometimes called exteroceptive measurements to emphasize that the information of these measurements comes from looking at the environment instead of at the robot itself.

2.1.3 Landmark Based Methods

One group of localization methods relies on the detection of landmarks. Landmarks are features in the environment that a robot can detect. Sensor readings from a robot are analyzed for the existence of landmarks in it. Once landmarks are detected, they are matched with a-priori known information of the environment to determine the position of the robot. Landmarks can be divided into active and passive landmarks.

Active landmarks, also called beacons, are landmarks that actively send out location information. Active landmarks can take on the form of satellites or other radio transmitting objects. A robot senses the signals sent out by the landmark to determine its position. Two closely related methods are commonly used to determine the absolute position of the robot using active landmarks: triangulation and trilateration [81]. Triangulation techniques use distances and angles to three or more

active landmarks; trilateration techniques only use distances. The angles and/or distances are then used to calculate the position and orientation of the robot. The GPS, or Global Positioning System, uses trilateration techniques to determine latitude, longitude and elevation. It uses time of flight information from uniquely coded radio signals sent from satellites. Twenty-four satellites orbit the earth in 12 hours in six different orbital planes. Ground stations track the satellites and send them information about their position. On their turn, the satellites broadcast information back to the earth. The result gives a position accuracy between 100 and 150 meters. To be able to use the mentioned methods, the robot needs to know the location of the landmarks in advance. Besides this, no a-priori information is required. There are some problems with these techniques though. The transmitting of the active signals can be disturbed by atmospheric and geographic influences while going from sender to receiver [81]. These disturbances can be refractions and reflections, and will result in incorrect measurements. Another problem is that active landmarks in practice often cannot send out their signals in all directions, and thus cannot be seen from all places. Furthermore, active landmarks may be expensive to construct and maintain.

If the landmarks do not actively transmit signals, the landmarks are called passive landmarks. The robot has to actively look for these landmarks to acquire position measurements. Techniques using passive landmarks in determining the position of the robot rely on detection of those landmarks from sensor readings. The detection of landmarks depends on the type of sensor used. For example, in detecting landmarks in images from a vision system, image processing techniques are used. When three or more landmarks are detected by the robot, it can use the triangulation or trilateration techniques to compute its location. Passive landmarks can be either artificial or natural and the choice of which kind of landmarks to use can play a significant role in the performance of the localization system.

Artificial landmarks are landmarks designed to be recognized by robots. They are placed at locations in the environment that are known in advance and that are well visible to the robot's sensors.

Natural landmarks are landmarks that are not specifically engineered to be used as localization means for robots. Natural landmarks are already part of the environment of the robot. In indoor environments, examples of passive natural landmarks are doors, windows, and ceiling lights, whereas in outdoor environments roads, trees, and traffic signs are candidates.

2.1.4 Map Based Methods

Another group of localization techniques are map based positioning or model matching techniques. These approaches use geometric features of the environment to compute the location of the robot. Examples of geometric features are the lines that

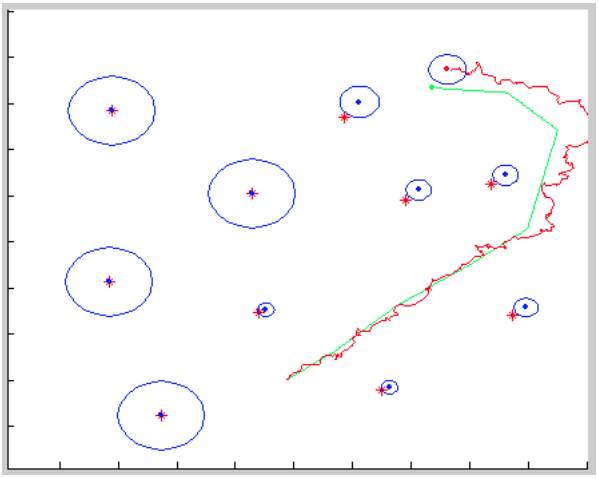


Figure 2.1. Example of feature-based map.

describe walls in hallways or offices. Sensor output, for example from sonars, is then matched with these features. Model matching can be used to update a global map in a dynamic environment, or to create a global map from different local maps [81]. The representation of maps can differ. It can either be geometric or topological. Geometric maps contain the environment in a global coordinate system. Topological maps contain the environment in the form of a network where nodes represent places in the environment and arcs between the nodes represent actions relating one location to another.

In this Phd thesis grid representations of the environment, called Occupancy grid maps, are employed. Occupancy grid maps, which were introduced in the 1980s by Moravec and Elfes [65], are a popular probabilistic approach to represent the environment. They are an approximative technique in which for each cell of a discrete grid the posterior probability that the corresponding area in the environment is occupied by an obstacle is is calculated. The advantage of occupancy grid maps lies in the fact that they do not rely on *any* predefined features. Additionally they offer a constant-time access to grid cells and provide the ability to represent unknown (unobserved) areas, which is important in many applications (e.g., exploration tasks). Their disadvantages lie in potential discretization errors and the high memory requirements.

2.2 Monte Carlo Localization

Monte Carlo Localization (MCL) is a well established localization algorithm which is based on particle filters. It has already become one of the most popular localization algorithms in robotics. It is easy to implement, and tends to work well across a

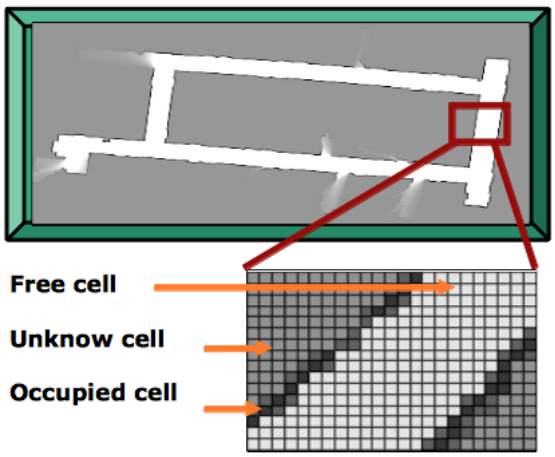


Figure 2.2. Example of grid-based map.

broad range of localization problems.

We recall here briefly some basic aspects of particle filters while using laser range finders. Particle filters are a sample-based implementation of a Bayes filters, which recursively estimate the pose of the robot p^t by representing the belief $Bel(p^t)$ using a set χ^t of n samples (called particles) distributed according to $Bel(p^t)$. The pose estimated by each particle of the set is indicated by $\hat{p}_i^t$. Basically particle filters realize the recursive Bayes filter using sampling procedures, usually referred as sequential importance sampling with resampling (SISR).

A time iteration of the basic particle filter localization algorithm is outlined in Algorithm 1. At each iteration, the algorithm receives a sample set χ^{t-1} representing the previous belief of the rover, a control input u^{t-1}, and an observation z^t, which usually includes range measurements and camera images. The control information usually consists of the rover's odometry readings. The posterior belief consists of n samples with weights normalized to one. The posterior belief generation process is made of several steps. First decide which sample to draw from the previous set. In this case, the sample is chosen with probability proportional to its weight. Then, the next state p^t is predicted using the generated sample and the control information u^{t-1}.
The update process is aimed at computing the posterior $p(p_t|z_t)$ and this is performed in steps 4-18. In particular step 10 computes $p(z^t|p^t)$, the importance weight.
In this approach the measurements z^t come only from the laser rangefinder, which provides high precision distance estimates, with errors of approximately 0.001 m in the range [0-32m]. Step 11 keeps track of the normalization factor, and Step 12 inserts the new sample into the sample set. After generating n samples, the weights are normalized to one.

This algorithm implements the Bayes filter, using an (approximate) sample-based representation, and the sample-based posterior converges to the true posterior at a rate of $\frac{1}{n}$ as the number n of samples goes to infinity.
Finally, a technique that reduces the number of needed samples is employed. This technique is called **KLD** sampling [38].

Require: $\chi^{t-1} = \{p_i^{t-1}, w_i^{t-1} \mid i = 1, ..., n\}$, representing belief $Bel(x^{t-1})$, N_{min}, N_{max}, control measurement u^{t-1} , observation z^t
Ensure: χ^t
1: **Initialization**
2: $\chi(t) = 0$
3: $\eta_\chi = 0$
4: **while** $\eta_\chi \leq i$ **do**
5: **Resampling**
6: Sample an index j from the discrete distribution given by the weights in S^{t-1}
7: **Next State Prediction**
8: Sample p_i^t from $p(x^t|x^{t-1}, u^{t-1})$ conditioned on p_j^{t-1} and u^{t-1}
9: **Update**
10: $w_i^t = p(z^t|p_i^t)$; // Compute importance weight
11: $\alpha = \alpha + w_i^t$; // Update normalization factor
12: $S^t = S^t \bigcup \langle p_i^t, w_i^t \rangle$; // Insert sample into sample set
13: $i = i + 1$;
14: **KLD criterion**
15: $\eta_\chi =$ **KLD**$(\chi^t, N_{min}, N_{max})$;
16: **if** $\eta_\chi > i$ **then**
17: break
18: $[\hat{p}^t] =$ **DT_clustering**(χ^t)

Algorithm 1: The particle filter algorithm in pseudocode

2.3 Multi-robot Localization in Highly Symmetrical Environments

In this Section the global localization problem in highly symmetrical environments is considered. In these environments it is supposed that absolute sensor data (such as GPS), is not available, since it could be unavailable in some indoor areas. It has to be noticed that "symmetrical environment" is not a so precise expression; in fact an environment can be completely symmetric, or can have some minor asymmetries. If the environment is completely symmetric, it is not possible to solve the ambiguity

without employing some sensor giving an absolute information. A possibility in this case is to use absolute heading measurements coming from a compass sensor. In the first part of this Section it is investigated multi-robot global localization in a completely symmetrical environment, where sometimes absolute heading measurements are available to the rovers and an algorithm for multirobot localization called **SMCL** is proposed and validated through simulations. In the second part of this Section, it is considered an environment with little asymmetries, where heading measurements are no more available. A new algorithm for multirobot localization called **3SMCL** is proposed, validated through simulations and real experiments. One of the points of interest is to see how the propagation of the information of the asymmetry is spread across the multirobot system, and how much this speeds up the localization of the team of robots, with respect to the single robot case.

2.3.1 Related Work

Multirobot collaboration is becoming one of the most challenging and promising research areas in mobile robotics. A team of rovers, suitably coordinated, can be used to execute complex tasks, as in surveillance, monitoring, and mapping, to cite only a few.

In these tasks the correct and reliable localization with respect to a known map is of capital importance, and represents one of the most fundamental problems in mobile robotics: a comprehensive study is reported in [86]. Potentially, the multirobot case gives some interesting advantages, since the accuracy of the rovers pose estimates can be improved by a cooperative localization, even if wireless communication and data sharing problems must be considered. Extended Kalman Filters (EKF) and Monte Carlo Localization (MCL) methods are the most common approaches to rover localization. The data association problem is generally solved in the EKF approaches by multi modal distributions that approximate the position probability distribution, sometimes including iterations that propagate also an estimate of the posterior marginal densities of the unknown variances (see e.g., [48], [60], [66], [71], [74], [79]). The MCL methods approximate an arbitrary posterior probability distribution by using particle filters (see e.g., [39], [43], [72], [77]). Cooperative robust multirobot localization has also been proposed, in which unknown but bounded error models are employed for the sensor measurements (see e.g., [62], [85]).

Localization includes two distinct sub-problems: *position tracking* and *global localization*. In the first one, the rover pose is iteratively estimated while the robot moves starting from an initial condition, known with a given uncertainty, while the second one determines the absolute rover position with respect to a given environment map; this problem is the most challenging, since no information of initial pose – or a completely wrong estimation of the actual pose, as in the so-called *kidnapped*

robot – is usually available.

Many of the papers cited above use multirobot and/or mutual localization to improve the quality of self-localization estimates that single rovers could achieve on the basis of their own sensors only, implicitly assuming that the measurements provided by such sensors would be sufficient to obtain a sufficiently correct, even if not precise, global localization. Unfortunately, without some external absolute information, a correct global self-localization cannot be performed by a single rover when the environment is highly symmetrical.

Highly symmetrical environments are commonly encountered in large logistic spaces, like the ones considered here, which present a team of rovers performing surveillance and monitoring tasks. A logistic space is similar to an indoor or outdoor warehouse, i.e., an area where logistic or transport companies receive, store and distribute large quantities of goods, as containers, cars, crates and other similar items. In order to achieve an efficient occupancy of the area and facilitate the handling operations, free corridors among the stored goods form a regular grid, as in Figure 2.3.

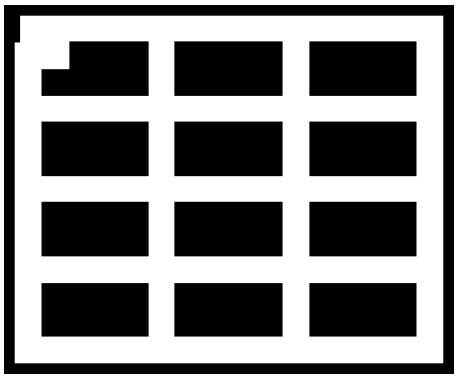


Figure 2.3. The map of the environment.

The symmetry of the environment map prevents a reliable global self-localization of each rover when its initial position is unknown, no specific landmarks are introduced to discriminate each corridor, and no absolute information (e.g., from GPS) is available or exchanged with the other rovers. By using only its own sensors (odometry, laser scanner, sonar, etc.) a rover could estimate its position within the corridors, but it may not determine in which corridor is actually moving.

2.3.2 The Three-State Multirobot Collaborative Localization (3SMCL)

Preliminaries

The *Three-State Multirobot Collaborative Localization* algorithm (3SMCL) ([12], [15]) allows each member of a group of rovers moving in a highly symmetrical area to accurately localize itself and to correctly track its position over time. The **3SMCL** algorithm correctly operates for rovers endowed with a basic set of sensors, like laser range sensors and a monocular camera. The camera is used to detect the positions of other rovers when they are in the field of view; the measurement accuracy is improved using the laser range finder. An occupancy grid map of the environment is assumed to be available.

The algorithm first performs *global localization* over the map in a decentralized way, exploiting the position estimates of the other rovers of the group, then realizes when the localization error estimate is lower than a given threshold, causing to switch to a pure *position tracking* algorithm. Finally, the algorithm allows rovers to detect a sudden increase in the localization error, due for instance to kidnapping or failures in proprioceptive sensors (e.g., wheel encoders rupture). In this case the algorithm switches again to *global localization*.

Let $\mathcal{R} = \{r_i : i = 1, \ldots, N_R\}$ be the set of rovers deployed in the area; with t we indicate the time variable that clocks the whole localization algorithm. With $d_i(t)$ we indicate data coming form the i-th robot proprioceptive and exteroceptive sensors at time t. In particular we have that

$$d_i(t) \quad = \quad \begin{cases} o_i(t) & \text{if proprioceptive measurement} \\ z_i(t) & \text{if exteroceptive measurement} \end{cases}$$

The proprioceptive measurement $o_i(t)$ is used to perform dead-reckoning, while the exteroceptive measurement $z_i(t)$ contains the range measurements given by the range sensors.

Each rover is able (a) to measure the positions of the other rovers in the field of view of its vision sensor in its local reference frame, (b) to transform the measurements in a global reference frame common to all rovers, and (c) to finally send these values to the detected rovers via a wireless link.

Let k denote a time instant at which the position of the i-th rover is detected by a set of rovers $R_i(k) \subseteq R$, ($|R_i(k)|$ being its cardinality). The rovers belonging to $R_i(k)$ send their measurements to the i-th rover, which collects them in the following vector:

$$h_i(k) \quad = \quad \begin{bmatrix} \hat{x}_i^{\,1}(k), \hat{y}_i^{\,1}(k) \\ \vdots \\ \hat{x}_i^{|R_i(k)|}(k), \hat{y}_i^{|R_i(k)|}(k) \end{bmatrix}. \tag{2.1}$$

Each row of (2.1) contains an hypothesis on the position of the i-th rover expressed in Cartesian global coordinates.

The set of all the measurements received by the i-th rover up to time k is then defined as $H_i^k = \{h_i(1), \ldots, h_i(k)\}$.

The algorithm has been conceived as a finite state machine, with three states: 1) GL = *global localization*, 2) UN = *undecided*, and 3) PT = *position tracking*. It represents an upgraded version of the multi-robot localization algorithm originally proposed in [11], called **3SMCL**, which was based on two states only, but applied to a *completely* symmetric environment, where absolute heading measurements are only occasionally available.

At the beginning of the execution of the algorithm, each rover is in the first state, as no information is available about its initial position, and hence it has to be *globally* localized with respect to the map. When the number of feasible hypotheses about its actual position becomes small (as detailed in the next Subsection), the rover enters the UN state. Finally, it switches to the PT state, when it is supposed to be correctly localized with a high degree of confidence. If a sudden increase in the localization error is detected, due for instance to kidnapping or failures in proprioceptive sensors (e.g., wheel encoders rupture) or changes in the map, the algorithm may switch again to the UN state.

The algorithm

We now describe the core of the **3SMCL** algorithm, which runs onboard each rover.

The algorithm outlined in Figure 2 is basically organized as a Finite State Machine (FMS) with three states. The first state is the default initial state and it is active when the rover is in GL state (lines 17-60). Then, when the accordance function defined in 2.5 is under a certain threshold, the algorithm enters the UN state. This state indicates that there are at most two relevant position hypotheses, but the algorithm is still not able to definitely choose which hypothesis has the highest likelihood.

When one hypothesis markedly prevails, i.e., when the localization performances are sufficiently accurate, the algorithms changes its state to PT.

Ideally, once reached the PT state, the rovers should never switch back to the UN state. However, the algorithm continues to monitor the localization performances. In case of localization performance degradation, the algorithm switches again to the UN state.

The algorithm is based on particle filters [86]; observing the pseudo-code in Algorithm 2 relative to the GL and UN states,

it can be noticed the typical prediction phase at line 19 and the update phase at lines 21-22. The prediction phase computes the vector $p_i(t)$ containing the predicted pose (in terms of global coordinates $(x,\ y,\ \theta)$) for each particle, while the purpose of the update phase is twofold. It gives the vector $w_i(t)$ containing the importance factors for each particle, and it verifies whether matrix $h_i(k)$ contains position estimates outside the map. If this is the case, such estimates are weighted using a Bivariate Normal Distribution. Then, at line 23, the algorithm verifies if it has received a vector of measurements $h_i(k)$ from other rovers of the set $R_i(k)$ at time k. If $R_i(k)$ is empty, a classic *Kullaback-Leibler Divergence* (*KLD*) resampling occurs (see [86]); N_{min} and N_{max} are respectively the lower and upper bound of the number of particles N_{kld} employed in the resampling algorithm. If instead $R_i(k)$ is not empty, a further improved version of the *KLD* resampling is implemented (line 29). The idea is to exploit the relative Cartesian position measurements (contained in vector $h_i(k)$) that the i-th rover receives from the other rovers of $R_i(k)$ to propagate the information about the few asymmetries of the environment. Instead of employing all the available particles $(N_{max} - N_{kld})$ to approximate each belief contained in matrix $h_i(k)$, in the solution here proposed we adopt the *Kullback-Leibler Divergence* to compute the number of particles needed to approximate each position hypothesis contained in matrix $h_i(k)$, and we propose an original resampling algorithm called *Mutual KLD* outlined in Algorithm 3.

In line 1 the number n_{hyp} of hypotheses received is calculated as the cardinality of the elements contained in $h_i(k)$. N_{kld} is the number of particles used to approximate the belief without taking into account the position hypotheses coming from the other rovers. This number is computed in lines 2-3. If this number is greater than $N_{max} - N_{hyp}$, we reduce it to $N_{max} - N_{hyp}$, where N_{hyp} is the maximum number of particles that in this case can be used to approximate the probability distribution of the n_{hyp} position hypotheses.
Then, the number of particles N^i_{mkld} is computed for the i-th hypothesis according to

$$N^i_{mkld} = \frac{1}{2\epsilon}\chi^2_{k-1,1-\delta}$$

which is the equation (13) in [38], where $\chi^2_{k-1,1-\delta}$ is a chi-square distribution with $1-k$ degrees of freedom. This number is the required number of particles to guarantee that with probability $1-\delta$ the Kullback-Leibler distance between the MLE of the position hypothesis and the true distribution is less than ϵ.
Our claim is that we do not need to know the exact probability distribution of the incoming hypotheses, because they are only used to add inaccurate information on the position of the i-th rover. Therefore we can accept an approximation of the probabiliy distribution of these hypotheses. By increasing or decreasing the value of ϵ and δ better or worse approximations of the distribution of each position hypothesis are set. This has indeed an impact on the number of particles used to approximate

Require: χ^{t-1}, $d_i(t)$, $R_i(k)$, H_i^k, N_{min}, N_{max}, N_{hyp}, m
Ensure: $\Phi_i(t)$, $\phi_i^{best}(t)$

if flag = 1 **then**
  state = 'PT'
  $[\chi^t, \mu_k, l]$ = **position_tracking**($d_i(t)$, χ^{t-1}, $h_i(k)$, l)
  $[\Phi_i(t), \phi_i^{best}(t)]$= **DT_clustering**($\chi_t$)
  if $l > n_{p2u}$ **then**
    $[\mu_k]$ = **loc_perf**($\phi_i^{best}(t)$, $h_i(k)$)
    if $\mu_k \geq \mu_{p2u}$ **then**
      $flag = 0$; $l = 0$
else
  if flag = 2 **then**
    state = 'UN'
  else if flag = 0 **then**
    state = 'GL'
  initialize χ_t
  if $d_i(t) = o_i(t)$ **then**
    $p_i(t)$ = **sample_motion_model**($d_i(t)$, $p_i(t-1)$)
  else if $d_i(t) = z_i(t)$ **then**
    $w_i(t)$ = **measurement_model**($d_i(t)$, $p_i(t)$, m)
    $\bar{\chi}^t = \bar{\chi}^t + \langle p_i(t), w_i(t) \rangle$
    if $R_i(k) = \emptyset$ **then**
      χ^t = **KLD_1**($\bar{\chi}^t$, N_{min}, N_{max})
      $[\Phi_i(t), \phi_i^{best}(t)]$ = **DT_clustering**(χ^t)
    else
      $l = l + 1$
      if state == 'GL' **then**
        χ^t = **Mutual_KLD**($\bar{\chi}^t$, N_{min}, N'_{max}, N_{hyp},$h_i(k)$)
        $[\Phi_i(t), \phi_i^{best}(t)]$ = **DT_clustering**(χ^t)
      Φ_{best} = best_hyp_extraction($\Phi(t)$)
      if state = 'UN' **then**
        if $l > n_{u2p}$ **then**
          $[\mu_k]$ = **loc_perf**($\phi_i^{best}(t)$,H_i^k)
          if $\mu_k \leq \mu_{u2p}$ **then**
            $flag = 1$; $l = 0$
        if $l > n_{u2g}$ **then**
          $[\mu_k]$ = **loc_perf**($\phi_i^{best}(t)$, H_i^k)
          if $\mu_k \leq \mu_{u2g}$ **then**
            $flag = 0$; $l = 0$
      if state = 'GL' **then**
        if $\bar{p}_i(t) > \mu_{g2u}$ **then**
          $flag = 2$

Algorithm 2: The 3SMCL algorithm in pseudocode

Require: $\bar{S}^t$, N_{min}, N_{max}, N_{hyp},N_{kld}, $R_i(k)$, ϵ
Ensure: χ^t

$n_{hyp} = |h_i(k)|$
if $N_{kld} > N_{max} - N_{hyp}$ **then**
  $N_{kld} = N_{max} - N_{hyp}$
if $N_{limitHyp} = \frac{N_{max} - N_{kld}}{n_{hyp}}$ **then**
  for $i = 1 : n_{hyp}$ **do**
    $N^i_{mkld} = \frac{1}{2\epsilon}\chi^2_{k-1,1-\delta}$
    $N^i_{curr} = min\{N^i_{mkld}, N_{limitHyp}\}$
    add N^i_{curr} to S^t_i
    $N_{kld} = \sum_i N^i_{curr}$, $i = 1, \ldots, n_{hyp}$
    $S^t = \sum_i S^t_i$, $i = 1, \ldots, n_{hyp}$

Algorithm 3: The Mutual *KLD* resampling

the final belief on the position of the rovers, as it will be discussed later in Remark 1, with reference to the results of the simulation test 4.
Higher values of ϵ allow to potentially use less than $N_{max} - N_{kld}$ particles to approximate the belief of the rovers, hence the value of ϵ in the case of our *Mutual KLD* is higher than the value adopted in the standard KLD resampling.
The value of ϵ can indeed be seen as a tunable parameter that changes the importance of the position hypotheses contained in $h_i(k)$ on the final belief of the i-th rover. We do not focus here on how to find general methodologies to calculate ϵ, but we simply set this value to five times the value of ϵ in the standard *KLD* sampling.
In line 8, for each hypothesis we constrain the number of particles N^i_{mkld} to be at most equal to $N_{limitHyp}$, which is the maximum number of particles that can be reserved for each hypothesis. The new N_{kld} number of particles which approximates the belief of the rover is computed in line 10 of the algorithm in Figure 3 as the sum over i of all the N^i_{curr}.

The Algorithm in Figure 3 can be seen as an extension of the adaptive resampling proposed in [38], since it adapts the sample set to represent the belief of a rover using also information coming from the other rovers of the multirobot team.

After this resampling phase, a classic *Density-Tree* clustering [39] (lines 4, 25, 30 of Figure 2) is always performed, providing a set of N_h hypotheses $\Phi_i(t) = \{\phi^j_i(t)\}$, $j = 1, \ldots, N_h$, on the position of the i-th rover, among which the best hypothesis $\phi_{best}(t)$ is selected. Each hypothesis $\phi^j_i(t)$ consists of the predicted pose $p^j_i(t)$, its covariance matrix $\Sigma^j_i(t)$, and the associated weight $W^j_i(t)$, representing its confidence level:

$$\phi^j_i(t) = \{p^j_i(t), \Sigma^j_i(t), W^j_i(t)\}. \tag{2.2}$$

The best hypothesis at time t is defined as

$$\phi^{best}_i(t) = \arg\max_{W^j_i}(\Phi_i(t)) = \{p^{best}_i(t), \Sigma^{best}_i(t), W^{best}_i(t)\}. \tag{2.3}$$

The mean distance among the hypotheses is defined as

$$\bar{p}_i(t) = \sum_{j=1}^{N_h} \frac{p^j_i(t)}{N_h}. \tag{2.4}$$

Switching rules among the three states are based on the following *accordance* function:

$$\mu_k = \sum_{q=k-n}^{k} \sum_{j=1}^{|R_i(q)|} \frac{\sqrt{(\hat{x}^j_i(q) - \hat{x}^{best}_i(t))^2 + (\hat{y}^j_i(q) - \hat{y}^{best}_i(t))^2}}{n|R_i(q)|}, \tag{2.5}$$

where n is the length of the sliding window used to compute the average in (2.5).

If the rover is in the UN state and it is verifying whether it can switch to PT, n is set equal to n_{u2p}. If the rover is in the UN state and it is verifying whether it has to switch back to GL, n is set equal to n_{u2g}. The inner summation in (2.5) averages the distances among the elements of $h_i(k)$ and the best position hypotheses of the i-th rover. The outer summation in (2.5) performs a moving average of length n on the results of the inner summation. Therefore μ_k measures the accordance between the actual belief on the position of the i-th rover and the average of the beliefs that the other rovers have on its position at time k.

When μ_k is lower than a certain threshold μ_{u2p} (empirically determined) the algorithm switches to PT. This phase is aimed at tracking the position of the rover over time, and it is implemented in a classic way (see [86]). μ_k is computed also during the position tracking phase: if μ_k becomes greater than a given threshold μ_{p2u}, the algorithm switches again to UN.

When the rover is in GL, it switches to UN if and only if the distance among the hypothesis defined in (2.4) is smaller than a certain threshold μ_{g2u} (see Figure 2, lines 50-52 and Figure 2.4).

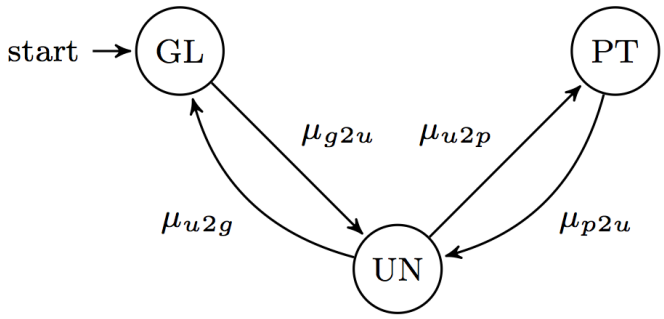


Figure 2.4. The states and the thresholds.

2.3.3 Simulation tests

In this Section we demonstrate the effectiveness of the proposed **3SMCL** algorithm, carrying out a series of online localization tests in simulation.

The Mobilesim simulator [64] provided with the Aria library was used to perform simulations of the rovers and their environment. It is based on the *Stage* library [42], and it simulates MobileRobots platforms. We performed experiments with a team of simulated Pioneer P3DX rovers, endowed with sonar sensors and laser range finders. The simulator embeds a model of the behavior of sonar and laser range finders, provides rover odometry pose estimation with cumulative error, and allows multiple

rovers simulation.
The simulator has also been improved by adding a simple simulated vision sensor and the support for communication among rovers.
Two environments were considered. The first environment simulates a large logistic area (see Figure 2.3). The occupied black areas can be thought to represent containers or similar bulky items stored by transport societies before distribution. The dimension of the whole environment is 80×65 m, the black areas are 20×10 m and the corridors are 5 m wide. A small asymmetry is present in the upper left corner.

The second environment is the hallway considered in [39] and reported here in Figure 2.5. We ran the **3SMCL** algorithm in this environment in order to perform some comparisons with the approach proposed in [39].

Simulation test 1

In this test we analyze the robustness of the **3SMCL** algorithm with respect to random variations in the initial position of the rovers. The localization error of the i-th rover is defined as

$$e_i^{\rho}(t) = \sqrt{(x_i(t) - \hat{x}_i^{best}(t))^2 + (y_i(t) - \hat{y}_i^{best}(t))^2} \tag{2.6}$$

$$e_i^{\theta}(t) = \theta_i(t) - \hat{\theta}_i^{best}(t) \tag{2.7}$$

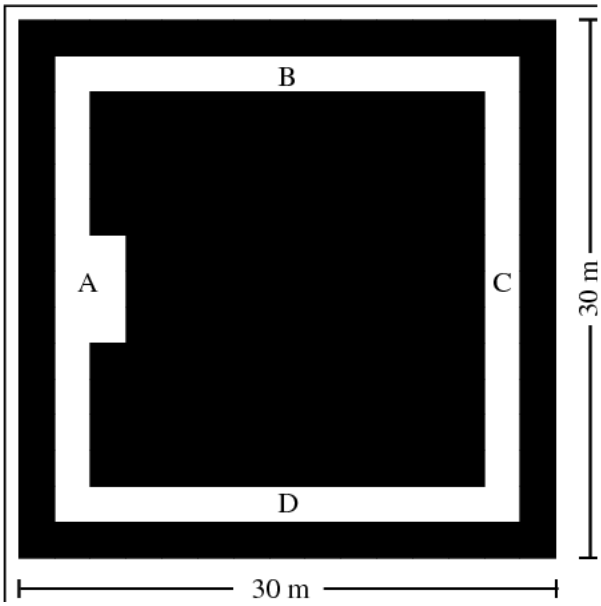


Figure 2.5. The hallway.

The pose informations of the best hypothesis are given by $p_i^{best}(t)$ and can be extracted by $\phi_i^{best}(t)$, defined in (2.3).
We randomly initialize the pose of $N_R = 6$ rovers in free areas of the map, let them move according to a simple obstacle avoidance behavior, and monitor the localization error $e_i^{\rho}(t)$ for $i = 1, \ldots, N_R$ up to $t = 2500$ s. We are interested in evaluating the average localization error among repetitions of the experiments, hence we repeat them several times, each time setting randomly the initial position of the rovers, and we define $\bar{e}_i^{r}(t)$ as the average of $e_i^{\rho}(t)$ for the i-th rover over n_e realizations. The results are shown in Figure 2.6 for $n_e = 100$.

The localization error $e_i^{\rho}(t)$, $i = 1, \ldots, N_R$ decreases approximately linearly for all the rovers, and the mean error among all the 6 rovers (dashed line in Figure 2.6) reaches a final value below 0.4 m. The **3SMCL** algorithm is thus not susceptible to variations in the initial positions of the rovers. This fact has an important impact on the application side, in particular when considering robotic applications in logistic spaces, since the algorithm does not require any particular initial formation of the rovers, avoiding any human intervention to initially place the rovers in a specific area of interest.

We now give the definition of the first and the last switching time from one state to another during the experiments. The first switching time is the moment when a rover changes its state for the first time, while the last switching time is the moment when a rover changes its state remaning in the final state. Figure 2.7 shows the

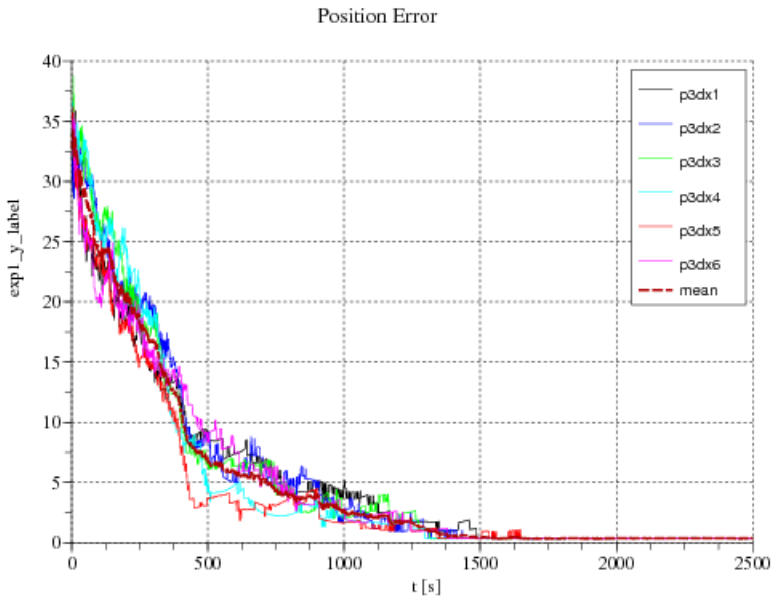


Figure 2.6. Simulation test 1: average localization errors.

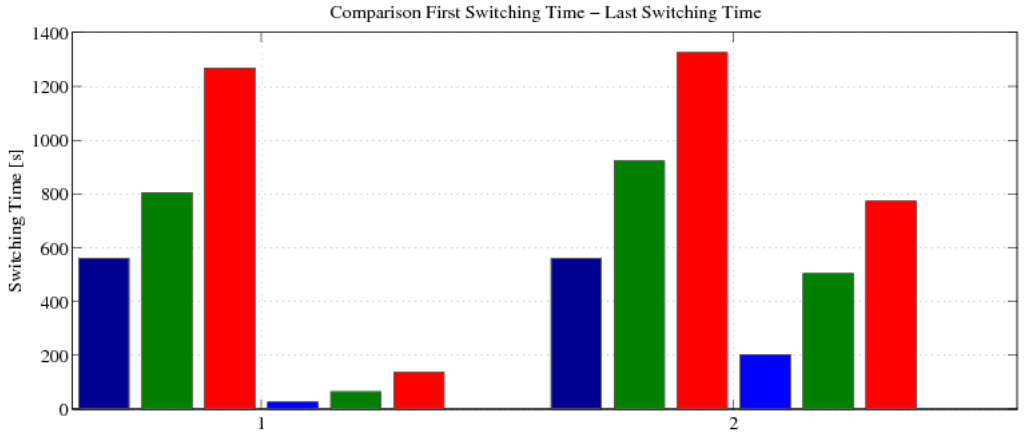


Figure 2.7. Simulation test 1: switching times to different states (see text for explanations)

comparison between the first switching time and the last switching time between the UN and the PT states and between the GL and UN states. The group of six bars on the left (group "1" on the x-axis) refers to the first switching time, while the group of six bars on the right (group "2" on the x-axis) refers to the last switching

time. The first three bars of each group are relative to the switching from UN to PT, while the second three bars are relative to the switching from GL to UN. The blue bars indicate the minimum among the switching times, the red bars the maximum and the green bars the average. Observing the second green bars of each group, which represent the average of respectively the first and the last switching time between the GL and the UN states, we notice a big difference, since the first switching time is around 70 s while the last switching time is around 500 s. This behavior is particularly positive, since rovers may switch to the UN state many times and in different moments during the localization process, thus avoiding false positive that may compromise the correct localization of the whole team. Observing then the first green bar of each group, it can be noticed that the last switching time occurs only two minutes after the first switching time. This means that the algorithm does not bounce for a long time between the UN and the PT states.

Simulation test 2

This test has been designed to understand how the localization performance of the **3SMCL** algorithm is affected by the number of rovers in the team, in terms of Cartesian position error.
We define the average position error among the N_R rovers of the team over the n_e realizations of the experiments as:

$$E^r_{N_R}(t) = \frac{1}{r}\sum_{j=1}^{r}\sum_{i=1}^{N_R}\frac{e^\rho_i(t)}{N_R} \tag{2.8}$$

where $e^\rho_i(t)$ is the localization error defined in equation (2.6). The results of the simulations for N_R = 1,3,6,9 are reported in Figure 2.8 and in Table 2.1.

Table 2.1. Correct Localization Percentage

# of rovers	Correct Localization Percentage
1	78%
3	98%
6,9	100%

It can be clearly seen that one rover is not sufficient to resolve the ambiguity in localization, and that also three rovers are not enough to assure reliable localization, since rovers are able to localize themselves correctly only the 98% of the trials. As soon as 6 rovers are employed, the localization error goes below 2.5 m after nearly

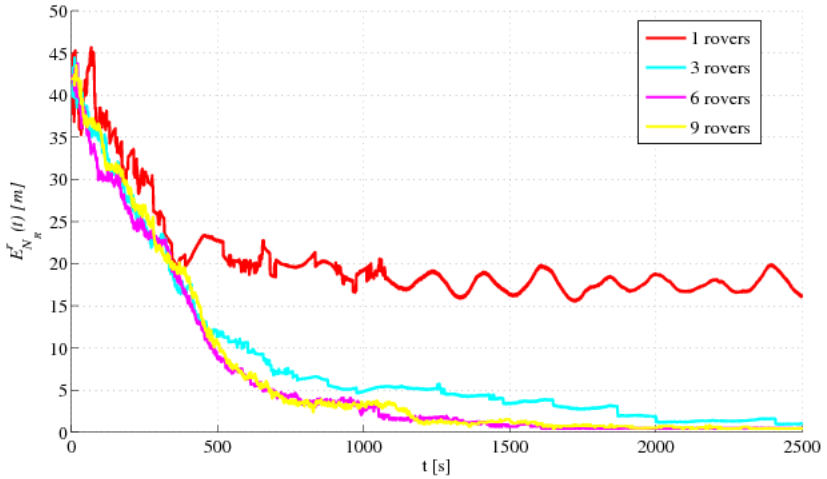


Figure 2.8. Simulation test 2: average localization errors.

1000 s, and all the trials are successful; the results with nine rovers are comparable with those obtained with six rovers.
Path planning algorithms become more effective for the rovers relying only on their position estimations, thus allowing rovers to accomplish in a more reliable way the assigned task (e.g., handling hazardous events collaboratively). Therefore the obtained results are particularly relevant in practical applications. Of course, the *exact* number of rovers that ensure correct localization of all the members of the team depends on the size of the area where the rovers move. Future investigations will be devoted to study the performance of the proposed algorithm with respect to variations of the ratio between the number of rovers and the area to be covered by the robot team.
Figure 2.9 finally shows the average first and last switching times from the UN to the PT states considering 3, 6 and 9 rovers. The switching times dramatically decrease passing from three rovers to six and remain nearly the same when passing from six to nine rovers. The performance of the **3SMCL** algorithm in terms of switching time improves increasing the number of rovers of the team, since there is a clear increase in the speed with which the rovers reach the PT state.

Simulation test 3

This test is aimed at demonstrating that, once the rovers are all in the PT state, the algorithm is robust even with respect to partial variations of the map. To show this robustness, we have set up a case study where $N_R = 6$ rovers are deployed in

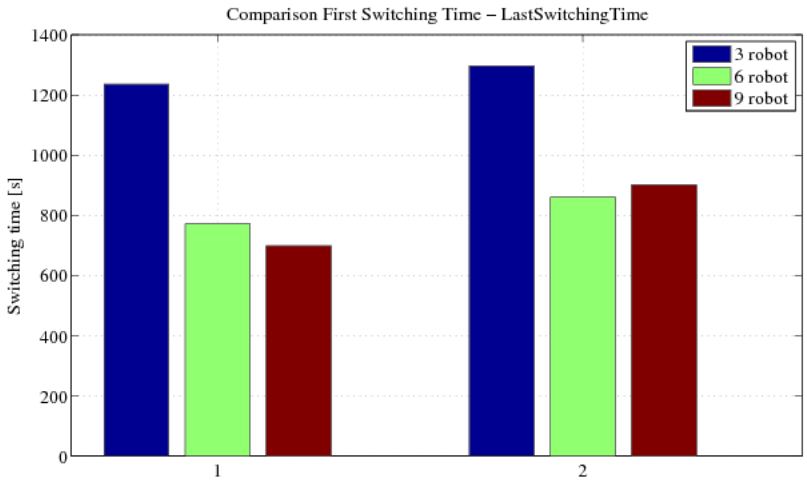


Figure 2.9. Simulation test 2: switching times for 3, 6 and 9 rovers

the same logistic area of the tests 1 and 2. After all the rovers have reached the PT state, a fork lift is supposed to enter the logistic area in order to remove and add pallets. The fork lift moves ideally at a constant speed of 1 m/s, and removes or adds randomly a pallet in the map, employing 3 seconds to perform these operations. Tests have been performed with a decreasing occupancy percentage, starting from 90% of occupancy in steady state condition, up to 50% occupancy with step of 10%. It is important to say that the informations about the map variations are not communicated to the rovers, therefore the challenge here for the **3SMCL** algorithm is to maintain the PT condition and to keep the localization error low for all the rovers. Figure 2.10 shows the localization error $e_i^{\rho}(t)$ for $i = 1, \ldots, 6$, considering only one realization of the experiment. The first plot shows the localization error reduction when the rovers, in each test, reach the PT condition. On average after approximately 700 s all the rovers in each test are in PT, and the algorithm that simulates the intervention of a fork lift begins to modify the map.
Observing the second plot, which is simply a zoom of the first plot, we see that the error increases, but only from 0.4 m to 0.6 m. Therefore the PT state of the proposed algorithm can be considered stable with respect to random variations in the map.

Simulation test 4

We have applied the **3SMCL** algorithm in localization experiments with the map shown in [39] and reported in Figure 2.5, in order to compare our results with those

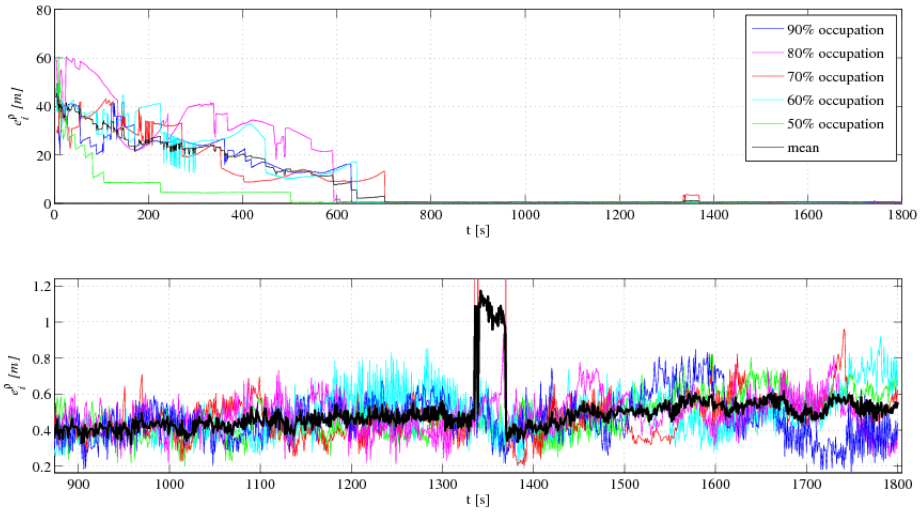


Figure 2.10. Simulation test 3: case study with variations in the map.

obtained in [39].
We performed three experiments: two of them with 3 and 6 rovers respectively, allowing exchange of mutual information (so running the **3SMCL** algorithm) and the third one with 6 rovers, with no information exchange. In this test case the algorithm reduces to a classical MCL algorithm with *KLD* resampling.

The number of repetitions is 50, as in Test 2. Figure 2.11 shows the average position error $E^r_{N_R}(t)$ for $N_R = 3$, with mutual information exchange allowed, and for $N_R = 6$, with and without mutual information allowed. The position error obtained with three rovers is comparable with that with six rovers and local communication allowed and it is significantly reduced with respect to the experiment with six rovers and communication not allowed. These results are comparable with those obtained in [39], where at 600 s, in absence of mutual exchange of information a final error than 4 m was achieved, while in our case it is less than 3 m. Considering instead the cases with mutual exchange of information, the final error is decreased to less than 1.2 m (while in [39] it was almost 2 m). It must be noted that in both cases ultrasound sensors were used, but the results reported in [39] are relative to eight rovers and averaged over 8 experiments only.

The proposed algorithm shows only a slight improvement in the localization time with respect to that in [39]. From our point of view the main improvement given by our work is relative to the possibility of checking if rovers are well-localized, i.e. they have all reached the PT state. In particular in all these experiments *all* rovers reach the PT state; the switching times are reported in Figure 2.12. We notice again

that the switching times on average decrease significantly passing from 3 rovers to 6 rovers. Considering in particular the experiment with 6 rovers, the last switching time is approximately 500 s on average and at that time the average position error is below 1.5 m. A remarkable result is that even if the number of rovers increases, they do not become overconfident about their belief. PT state is always reached when the rovers are correctly localized.

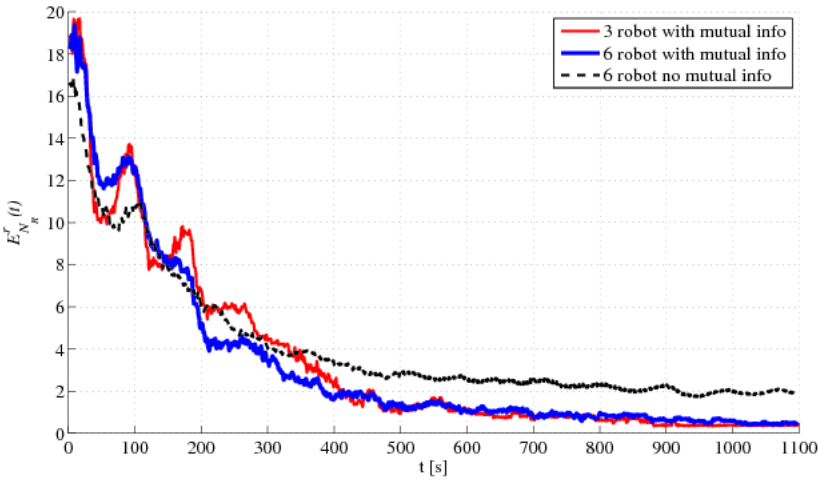


Figure 2.11. Simulation test 4: average localization errors.

Remark 1 *In Section 2.3.2 we claimed that the number of particles employed to approximate the probability distribution of the position hypotheses is generally less than $N_{max} - N_{kld}$. Figure 2.13 shows the results of a generic i-th rover in a single experiment.*
The blue plot is the number of particles resampled at time t, the red plot is the maximum number of particles that could be used to approximate each element of $h_i(k)$ at each time k and the black plot is $\frac{N^i_{mkld}}{nHyp}$ (see Algorithm 3), i.e., the actual number of particles that the Mutual KLD *algorithm uses to approximate each position hypothesis.*

Experimental tests

We also tested the proposed algorithm in the real world using real robotic platforms. The environment is 20 × 12 m and possesses only little asymmetries. We use a team of three Pioneer P3DX endowed with ultrasound sensors, SICK LMS200 laser rangefinders and low cost monocular vision sensors. Each rover of the team

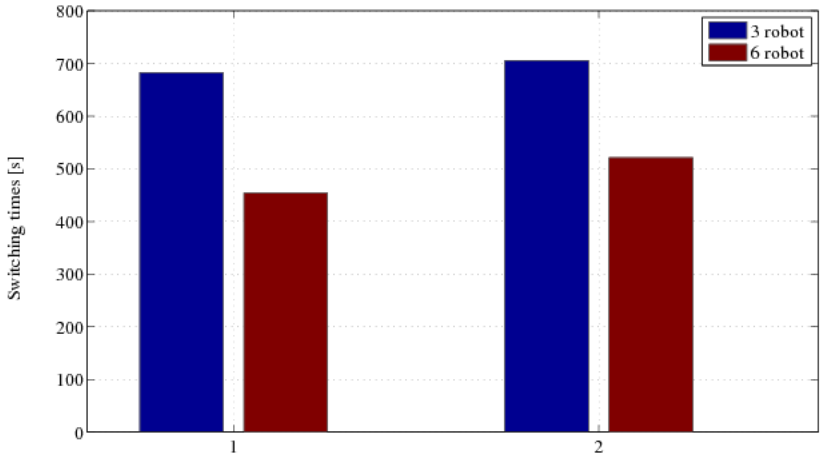


Figure 2.12. Simulation test 4: switching times.

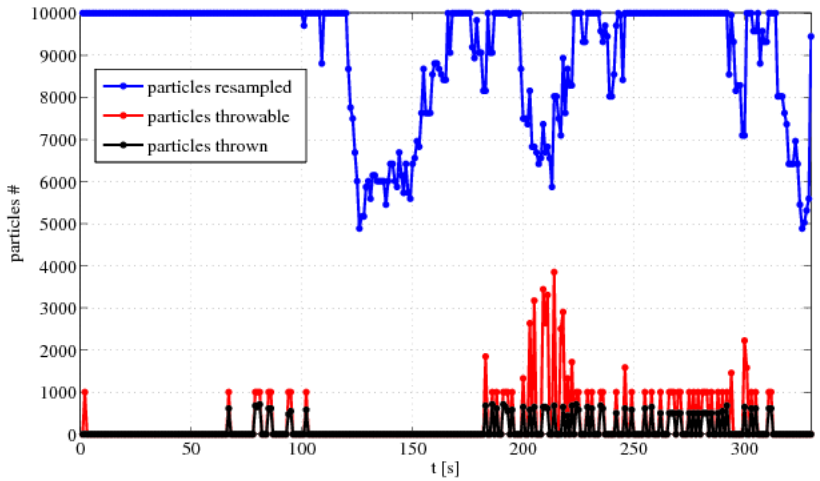


Figure 2.13. Comparison among the number of particles employed to approximate the position hypotheses

is able to identify the others when they are in a certain field of view, using barcodes as identificators and vision sensors to detect the barcodes. Figure 2.14 shows the rovers team, while Figure 2.15 reports a grid representation of the site where the experiments have been carried out, showing in particular the minor differences (asymmetries) that distinguish the sides of the environment.

The first test (A) has been performed using two rovers, to show that a well localized rover can speed up the localization of the other that is still uncertain about its position. A rover is positioned near the main asymmetry of the environment (R1 in Figure 2.14), while the other is positioned in a place where there are no discriminant features that can ensure fast localization (see the same Figure). The initial robot heading the rovers are indicated by the arrows in Figure 2.14.
The result of the test is reported in Figure 2.16, where the red plot is relative to the rover R1 and the blue plot to rover R2. The sudden decrease in the localization error

Figure 2.14. The team of Pioneer P3DX rovers.

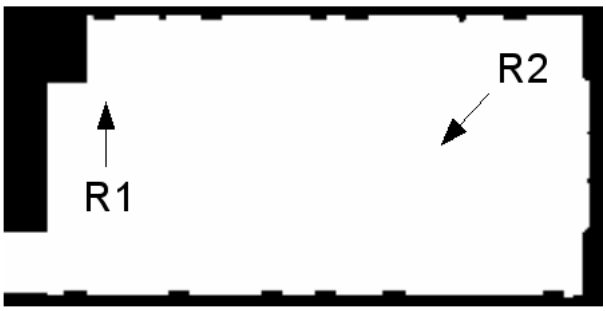


Figure 2.15. The map of the site where the localization experiments have been carried out.

for rover R2 at 116 s is due to the fact that at a certain point during the experiment the rovers detect each other and the belief of the rovers are mutually influenced. The overall result is that the well localized rover (R1) speeds up the localization of the other rover (R2). This simple example demonstrates that the mutual exchange of position information among rovers using the proposed algorithm can significantly speed up the localization process in a real experiments.

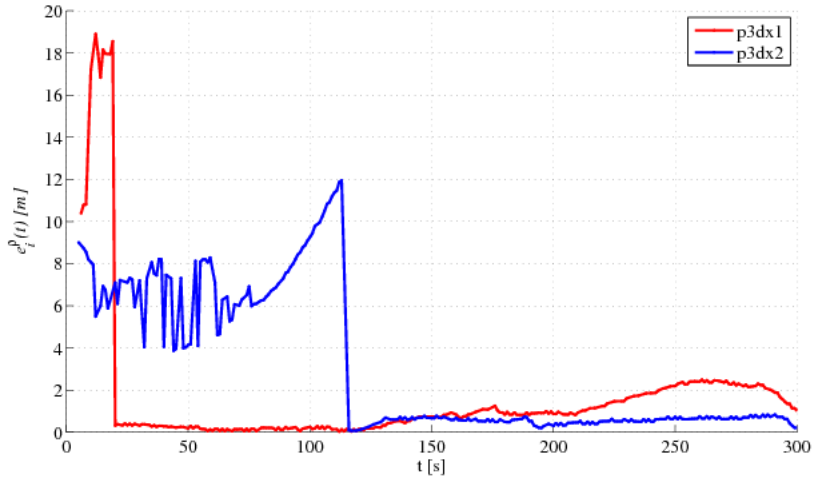


Figure 2.16. Experimental test A: localization errors

The second test (B) is a classical kidnapping test. The two rovers are randomly placed in the environment and, as shown in Figure 2.17, one rover (blue plot) is

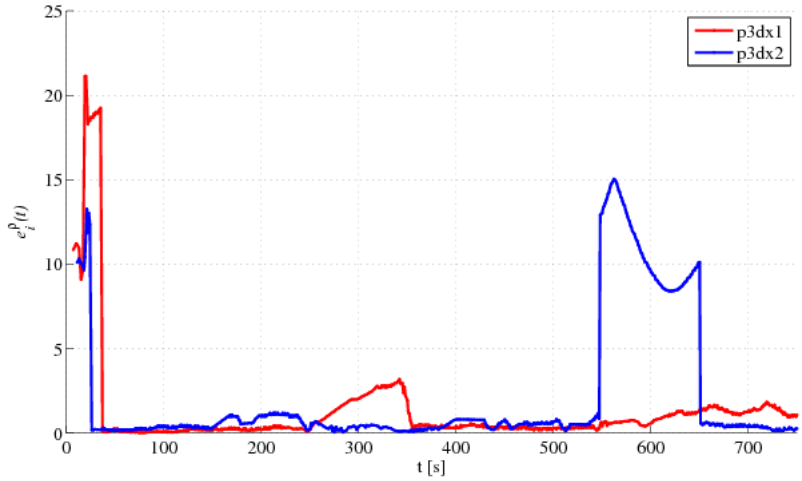


Figure 2.17. Experimental test B: localization errors

kidnapped approximately at 500 s, and after 150 s is localized again by the other. Differently from test (A), which only demonstrates that localization can be speed up using the **3SMCL** algorithm, here we notice that the belief of a not well localized

rover is influenced by the belief of a correctly localized one.

In the final test (C), we performed $r = 5$ mutual localization experiments using $N_R = 3$ rovers randomly placed in the environment. Figure 2.18 shows the behavior of the average localization error, defined as:

$$E^j_{N_R}(t) = \sum_{i=1}^{N_R} \frac{e^\rho_i(t)}{N_R} \quad j = 1, \ldots, r. \tag{2.9}$$

In this experiment we observed that the first and last switching times are the same,

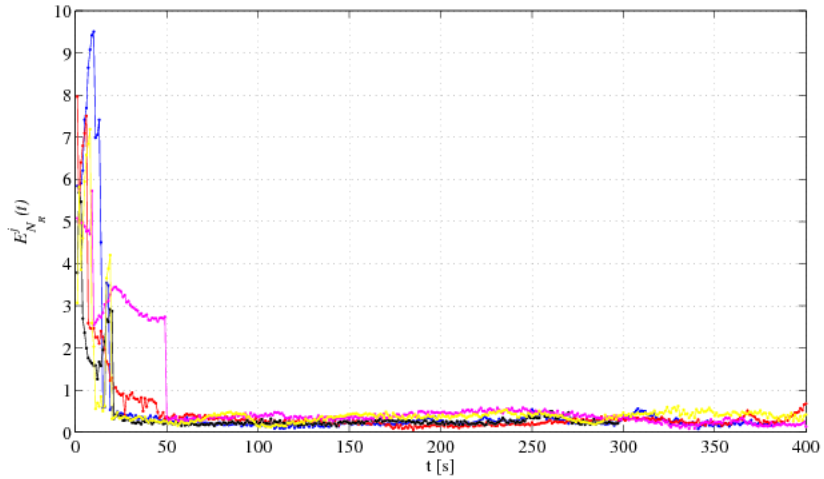


Figure 2.18. Experimentak test C: average localization errors

so the rovers do not switch back from the PT state to the UN state. On average these switching times are around 450 s.

2.4 Localization with sensor fusion and markers detection

Robustness issues still exist for classical laser-based localization approaches, in particular when dealing with large, symmetric and dynamic environments such as corridors and office-like environments. In these scenarios, localization algorithms can fail over time. This is due to lack of significative features (in a corridor-like environment the only recognizable obstacles are usually doors) combined with non-accurate wheel odometry. For this reason we investigated ways of improving the robustness of these approaches by adding two other sensors: an an inertial measurement unit (IMU) and later an Asus Xtion RGB-D camera. While it is clear that the use of IMU-corrected odometry rotation can improve localization performances, we found that this is still not enough for robust robot localization. In particular, we found in our experimental tests that the localization algorithm can still fail over long distances along corridors. We decided to add a visual marker detector in order to provide a recovery tool for the AMCL localization algorithm, in particular when the robots travel inside long corridors or very symmetric areas.

2.4.1 IMU-corrected odometry

Relative displacement of robots is usually based on odometry. Odometry is simple, inexpensive, and easy to accomplish in real-time. For wheeled nonholonomic robotic platform odometry estimation is usually based on different kinds of proprioceptive sensors, such as wheel encoders (wheel odometry), compass, gyroscope, accelerometers, inertial measurement units (IMU) (IMU odometry). The disadvantage of odometry is its unbounded accumulation of errors.

Wheel odometry is usually obtained by measuring the number of turns of the wheels using optical shaft encoders and integrating those measures over time in order to estimate the displacement of the robot. Magnetic compasses can be used as a method for correcting heading estimation errors. The major drawback is that the compass is subject to local magnetic anomalies, which can lead to large errors in heading estimates. Also, because the Earth's magnetic lines of flux "dip" in declination, the compass must remain level for the readings to be accurate. An alternate method for maintaining an accurate heading for odometry calculations is the use of a gyroscope. With the introduction of optical fiber gyros the use of gyros has become more attractive for mobile robot applications. However, gyros have relatively large drift rates, which cause unbounded growth in orientation errors.

We developed a simple sensor fusion algorithm which is able to fuse wheel odometry and gyroscopes measures to obtain a corrected odometry which is much more accurate, and can be used for localization as well as a starting point for mapping.

(citare graph-slam) We use displacement from wheel odometry, since it is usually accurate enough, and correct heading using measure from the IMU's gyroscope, since heading estimation from wheel odometry is usually bad due to wheel slipping. The IMU that was used is a medium-cost 3-axis commercial sensor (XSens MTI) which is commonly used for mobile robotics applications. We first experimented with the heading estimation given by the internal Extended Kalman filter running onboard of the sensor, but we found that the heading estimation was very bad when used onboard of the robot. This is due to the fact that the magnetometers are hugely affected by magnetic disturbances in the environment. Being the IMU placed on a metallic platform, the heading information is always biased. For this reason, we chose to use gyroscopes only.

When the robot is not moving, angular velocity measurements are used for gyroscope bias error estimation with a moving average.

2.4.2 Planar markers detection

There are many practical vision systems that use two-dimensional patterns to carry some kind of information [16]. The fields of application range from industrial systems, where markers are designed to label goods or areas or carry certain information, e. g. shipping data, to systems where markers are used for pose localization, e. g. augmented reality and robot navigation systems. Examples for the first case are Maxicode, used by the US Postal Service, and DataMatrix and QR (Quick Response), used in industrial settings for the purpose of part labeling. Examples for the second case are ARToolKit, ARTag and ARSTudio, three systems for Augmented Reality. In order to reduce sensitivity to lightning conditions and camera settings, these planar marker systems typically use bitonal markers (black and white). Furthermore many systems use some of the marker's data bits to convey redundant information, which allows for error detection and correction. The design of the markers mainly depends on the application.

DataMatrix, Maxicode and QR are applicable for encoding information under controlled environments, e. g. conveyor belts, but are not very suitable for systems that use markers for localization. The markers of these three systems are not designed for large fields of view and the perspective distortions involved. Furthermore they require a large area in the image, so that the range at which these markers can be used is very limited. And finally they do not provide enough points in the image to enable three-dimensional pose calculation. For Augmented Reality applications on the other hand it is very important that markers can be found within a large field of view. This means that they should also be detected in the present of large distortions in the image. On the other hand, the information stored inside the marker must not be too dense in order to increase the maximum distance at which data can be recovered from the marker. The only data an augmented reality marker

should carry is a way to identify itself (for example an ID number).

Our requirements for a marker detection system are:

- Single camera pose tracking
- The ability to use arbitrary markers
- Fast enough for real time applications
- Open source implementation

Based on these requirements, in our application we use the ARToolkit library [8]. ARToolKit is a popular open source planar marker system for augmented reality applications. The bitonal markers consist of a square black border and a pattern in the interior. The library is able to recognize arbitrary 2D visual markers (called ARTags, see Figure 2.19). The first stage of the recognition process is finding the markers' black borders, that is finding connected groups of pixels below a certain gray value threshold. Then the contour of each group is extracted, and finally those groups surrounded by four straight lines are marked as potential markers. The four corners of every potential marker are used to calculate a homography matrix in order to correct perspective distortion. Once the internal pattern of a marker is brought to a canonical front view one can sample a grid of NxN (usually 16x16 or 32x32) gray values inside. These gray values form a feature vector that is compared to a library of feature vectors of known markers by correlation. The output of this template matching is a confidence factor. If this confidence factor is greater than a threshold, a marker has been found.

Although ARToolKit is useful for many applications, there are some draw-backs. First of all the detection process is threshold based. A single threshold can easily fail to detect markers under different illumination conditions, even within the same image. Furthermore the marker verification and identification mechanism using correlation causes high false positive and inter-marker confusion rates. With increasing library size the marker uniqueness is reduced, which again increases the inter-marker confusion rate. Moreover, pose tracking, in particular regarding 3D orientation estimation, is not stable enough for our application.

We decided to improve the robustness of pose estimation by using an RGB-D camera instead of a monocular camera. This makes it possible to better estimate the pose of markers in the environment, by fusing 2D and 3D data. We use ARToolKit on the planar image for finding the corners of the marker, and then we identify a plane which corresponds to the detected marker using the distance information from the RGB-D sensor. We then compute a transformation to an ideal marker geometry in order to retrieve the correct 3D pose of the marker.

We then implemented a modified version of the classical Adaptive Monte Carlo Localization (AMCL) algorithm. We introduce a new sensor model in the update

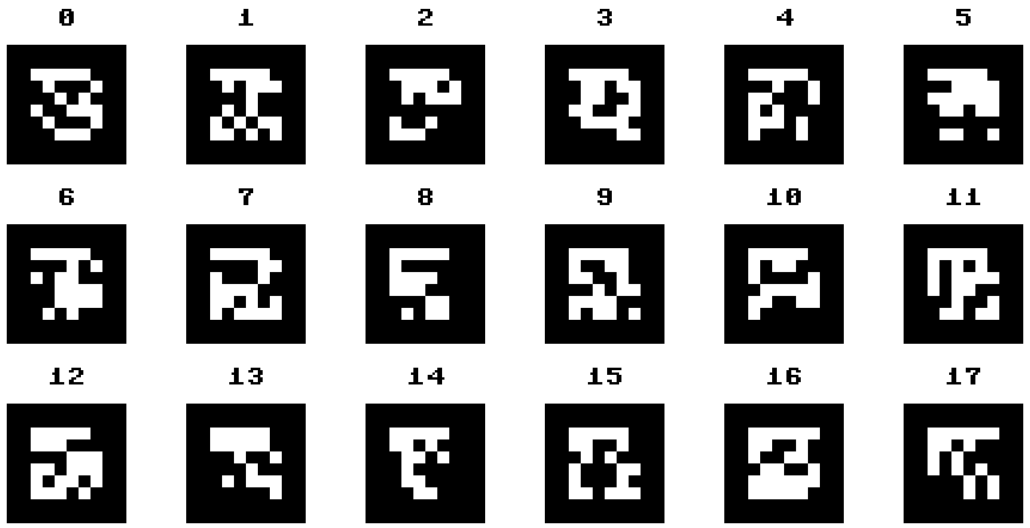


Figure 2.19. Example of binary ArTags.

phase of the algorithm, which integrates marker detection into the particle filter. We proceeded to integrate the result of markers detection into our localization algorithm considering two levels of integration, a *loose integration* and a *tight intregration*.

In the loose case detected markers provide directly a robot pose hypothesis, with a fixed covariance error; in the second case the detected markers are used as additional inputs for the particle filter-based localization.

In tight coupling the position estimates come both from the laser range finder and from marker detection algorithm described before. At each time instant t for each particle we have a measurement $z^{(i)}(t)$ defined as:

$$z^{(i)}(t) \quad = \quad \begin{cases} r_j(t), j = 1, \ldots, N_r & \text{if laser measurement} \\ \hat{x}_t^{marker}, \hat{y}_t^{marker} & \text{if maker-based estimate} \end{cases} \tag{2.10}$$

where N_r is the number of rays of the laser range finder. If a laser measurement occurs, the update phase is the usual one. If an output from the algorithm described before is provided, particles are weighted using the pose estimated from the marker detection algorithm, using a fixed covariance.

2.4.3 ROS implementation

We implemented a ROS node for planar marker detection based on the ARToolkit library [8] and the *ar_kinect* ROS node. The ROS node that we developed is

based on the use of a Microsoft Kinect camera. In particular, markers are first detected using the ARToolkit library, and their pose is then estimated by searching for square plain areas in their correspondent positions in the 3D point cloud. The node transforms the 3D pose of the recognized ArTags from the coordinate frame of ArToolkit to the coordinate frame of the map used for localization. A number of checks is done in order to discard false positives, which are very common in difficult lightning conditions such as closed indoor environments. In particular all detection over a range of 2.5 m are discarded, as well as spot detections and makers poses which are not parallel to the walls and floor (presenting large roll and pitch angles). In order for the node to work, two things are needed:

- A number of ARTags have to be put in the environment
- The appearance and the pose of the markers in the map have to be written into a configuration file

The localization algorithm that we use is a modified version of the *amcl* ROS package. In our version of the algorithm global localization is also possible, as in the approach that we developed and described in Section 2.3.2. We use the *likelihood field* laser sensor model for particle update, since it is faster and better represent sensor readings compared to the classical *beam model*, it is smooth with respect to small changes in robot position and is better suited for small obstacles.

When the robot detects a marker, the correct pose of the robot in the map is estimated relative to the marker and AMCL is re-initialized around this pose (with a suitable covariance, in order to deal with visual detection uncertainties) in the case of loose integration. In the case of tight integration the particles are weighted according a gaussian sensor model around the estimated pose with a fixed covariance.

Notes on parameters selection

It should be noted that the choice of the AMCL parameters is critical for achieving low localization error, in particular with a challenging scenario. Good localization is crucial for the subsequent path planning. The most important parameter values that we use in our modified ROS implementation of the AMCL algorithm are reported in Table 2.2. The error in laser readings caused by the metal grids has been modeled in a trivial way by raising the *laser_z_hit* value of the likelihood field sensor model. The higher *laser_z_rand* value accounts for the presence of glass-covered racks. We experimentally found that a maximum particle size of 10000 is enough for reliable global localization, and a minimum of 500 is enough for modeling the robot pose during position tracking.

Table 2.2. Parameters used for our implementation of the AMCL ROS node.

Parameter	Value
max_particles	10000
min_particles	500
laser_z_hit	0.5
laser_z_rand	0.5
update_min_d	0.1
update_min_a	0.25
resample_interval	1

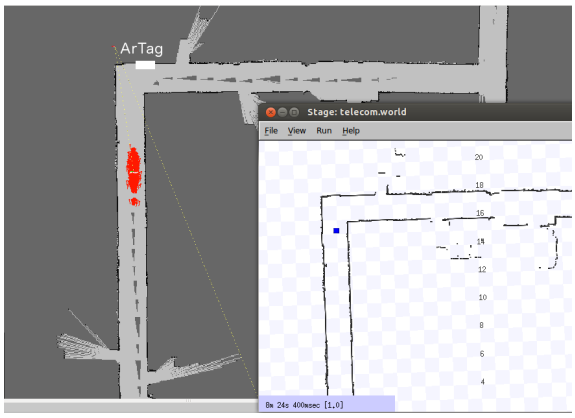
2.4.4 Results

The approach has been first tested in a corridor-like environment at Telecom Italia Lab, using a Coroware Corobot Classic robotic platform with a Microsoft Kinect sensor and a XSens MTI IMU. During preliminary experiments two issues have shown up:

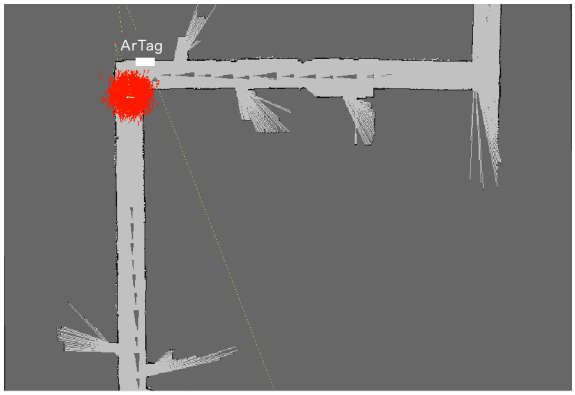
- false detections of markers, due to difficult lightning conditions
- markers not being detected, due to their bad placement in the environment

In a successive implementation of the algorithm false detections have been ruled out by implementing a number of checks on the placement of the markers: for instance, markers are only detected within a range of 2.5m, since the accuracy of the Kinect sensor degrades at larger distances. Moreover, marker detections which are not aligned with the walls are discarded. Marker placement is crucial, as markers need to be placed only at the end of long symmetric corridors and should face the camera mounted on the robot, in order to be correctly detected. From the results we saw that the robot is able to recover its pose estimate at the end of the longer corridors most of the times.

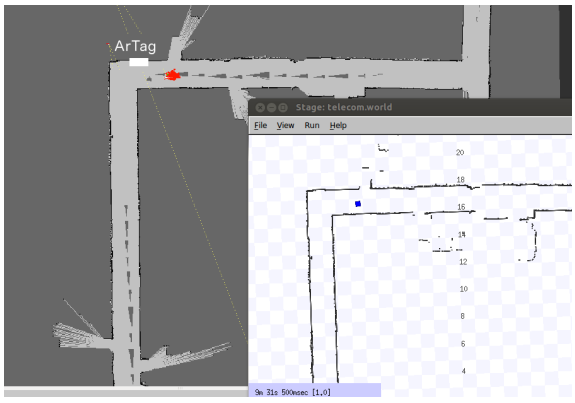
Figure 2.20 shows an example of an experiment in this environment. In 2.20(a) the robot is travelling along a corridor with few asymmetries (open doors). Moreover many of the doors in the corridor are in a different state (open, closed) than what is present on the map. The real robot position in the environment is shown as a blue square in the simulator (right), while the position estimated by the localization algorithm is represented by the red particles on the left. It can be seen the estimated position is incorrect, due to inaccurate odometry and lack of map features. An ArTag is attached on the wall in the corridor in front of the robot, and its pose (position and orientation) is known inside the map of the environment (it has been measured by hand). In 2.20(b), as the robot moves forward, it is able to detect and recognize



(a)



(b)



(c)

Figure 2.20. Example of binary ArTags.

the marker using the onboard Kinect sensor. The pose of the marker is estimated relative to the robot, and the pose of the robot relative to the map is retrieved. In this example, we use the loose integration, and the particle filter is re-initialized aroud the estimated robot pose with a suitable covariance, in order to take into account detection errors. In 2.20(c) the robot continues to move along the corridor and is able to recover a correct pose estimate.

Another test has been done inside the corridors of the data center. Although the Corobot platform was able to correctly localize even without the need of markers, two markers have been placed in key points of the environment for improving robustness and implementing a fail-safe service.

In a third experiment, we tested localization and path planning performances in another datacenter room, with a previously created map. This particular scenario presents some challenges for localization, such as its symmetry and irregular surfaces. Data-center areas are organized as regular grids, with rows of racks and corridors. In particular, server racks are usually covered with metallic grids, which introduce large errors in range measures, as laser rays are passing through the grids introducing measurement errors. Moreover, the symmetry of the environment poses a challenge for loop closings detection.

A path composed by a certain number of waypoints was created. The tasks associated with each waypoint were thermal camera image acquisition and temperature measurement. Figure 2.21 shows the results of a typical experiment. It can be noted that the robot correctly localized itself and was able to follow the given path.

We measured the localization accuracy at different points inside the map. The ground-truth for each reference point (position and orientation) has been measured by hand. Table 2.3 show that the average position and rotation errors are higher than ..., but are still adequate for the subsequent path planning, as shown in the next experiment.

Table 2.3. Average localization errors.

	Average	**Variance**
Position [m]	0.158	0.0016
Rotation [deg]	1.47	0.0229

2.5 Map Updating in Dynamic Environments

In this chapter we present an approach called Δ-mapping [14] that allows a correctly localized robot (being in the *position tracking* state described in the previous

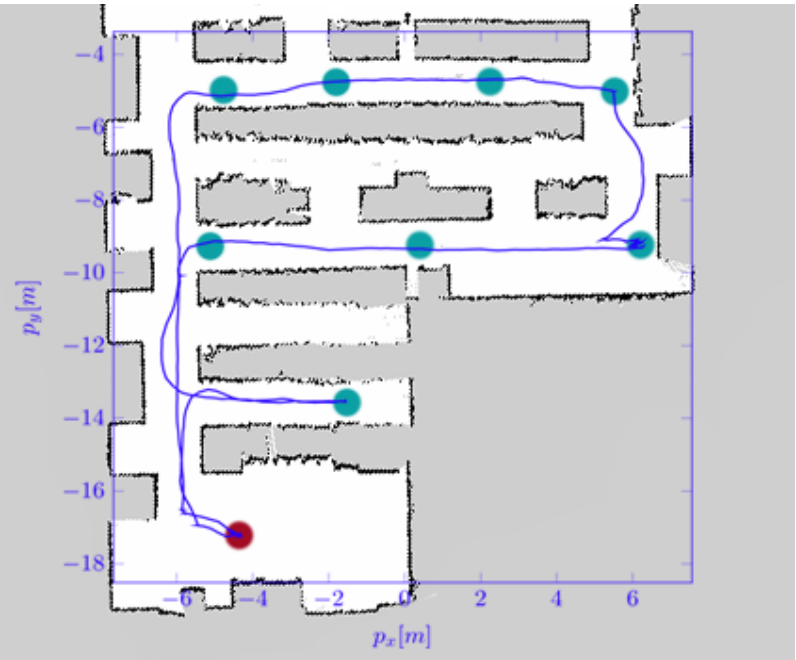


Figure 2.21. Localization and path planning performances. (a) Trajectory followed by the robot, as estimated by localization, is shown in blue. Waypoints are shown as points.

Section) to perform map updating in dynamic environments, and we show some preliminary results in simulated and real experiments.

2.5.1 Introduction

Localization and map building are closely linked problems, and are often referred to as *Simultaneous Localization and Mapping* (SLAM) (see Chapter 3). While building maps when robot poses are known is a tractable problem with limited computational complexity, in the SLAM case the uncertainty in measures and in robot pose estimates makes the problem much more complex in terms of data association and computational requirements. Moreover, the vast majority of the effort has been devoted to the case of mapping of static environments.

Many real applications require updated maps of the environment that vary over time, starting from a given initial condition. This is for instance the case of robotic applications in logistic spaces, where robots have to track the presence of goods in a certain area. The goods are stored in appropriate places, but during the day they can be removed and substituted by other items many times.
This work deals with the problem of keeping the map updated, in order to guarantee the robots localization; specific goods tracking procedure are not investigated.

The initial map can be a-priori known or built using a classical SLAM algorithm. In the latter case, the environment is supposed to be static during the initial SLAM process. Then the environment starts to change, and the map needs to be updated, as modifications are sensed by the robot.
During the map update process, there are mainly two issues to be considered:

- long-term operativity is required;
- the algorithm performing the map update has to be computationally light and to use limited memory.

The first item is mandatory since in real applications map variations may occur continuously for a long period of time (even an entire day), and the map updating process should be carried out as long as possible without diverging.
The second item is very important when the map updating process has to be carried out in parallel with other tasks, (e.g., team coordination and planning, surveillance).

A combined algorithm for map update and robot localization is proposed in [90], while a method for updating the map dynamically during the process of localization is developed in [63]. This method seems very promising but its experimental validation is still only partial. The problem of grid mapping through belief propagation is investigated in [76], assuming that good estimates of the robot position are available. A spatial Markov random field model is applied to a minefield mapping. Two things are required: the list of the type of objects that populate the environment and the a-priori knowledge about where the objects are positioned.
Finally [18] introduces a dynamic map representation for mobile robots that adapts continuously over time. It solves the stability-plasticity dilemma (the trade-off between adaptation to new patterns and preservation of old patterns) by representing the environment over multiple timescales simultaneously.

In this Section we propose a methodology that is able to:

- detect variations in the environment;
- generate a local map containing only the persistent variations;
- merge the local map with the global one used for localization.

The variations of the environment are detected using a technique called *weighted recency averaging*, while the local maps are merged employing the Hough transform.

2.5.2 Problem Formulation

A mobile robot, endowed with a laser rangefinder, is supposed to be correctly localized with respect to the available environment map. Let its estimated pose at time

t be denoted by

$$\hat{p}(t) = \{\hat{x}(t), \hat{y}(t), \hat{\theta}(t)\}. \tag{2.11}$$

By the expression *correctly localized robot* we indicate a robot that is in the *position tracking* state, as defined in [11] and [12].
An occupancy grid map of the environment (used in the localization algorithm to track the robot position over time) is available to the robot. Such a map could have been manually created or previously built by a SLAM algorithm.
At discrete instants k the environment changes, and consequently the robot has to modify its map, to take into account the variation. We call this phase a Δ-*mapping* step.
We define the set of new maps collected up to time k as

$$\mathcal{M}(K) = \{M_k\},\ k = 0, \ldots, K.$$

M_0 is the initial map, obtained by the SLAM procedure. The goal of the algorithm developed in the next section is to provide an estimate $\hat{M}_k$ of the map at each time step k.

2.5.3 The Approach

The architecture of the approach can be represented by separate functional blocks (as shown in Figure 2.22).

The Δ-Awareness block implements the technique used to detect when a change has occurred in the map, and consequently communicates to the robot that it is time to enter the so called Δ-*mapping* phase.

The *Store-scan* block decides which scans acquired during the Δ-mapping phase are suitable to create a local map containing the variations.

The *Alignment* block registers in a consistent way the set of measurements frames (range scans) colleced by the *Store-scan* block. The approach maintains all the local frames of data as well as the relative spatial relationships between local frames. The adopted approach is described in [57].

The *Map Merge* block merges the output of the *Alignment* block at time k with the map $\hat{M}_{k-1}$.
The most relevant blocks are detailed in the next subsections.

Δ-Awareness

The Δ-Awareness block detects persistent variations in the environment, using a technique called *weighted recency averaging*, which is normally applied in problems of tracking non-stationary processes.

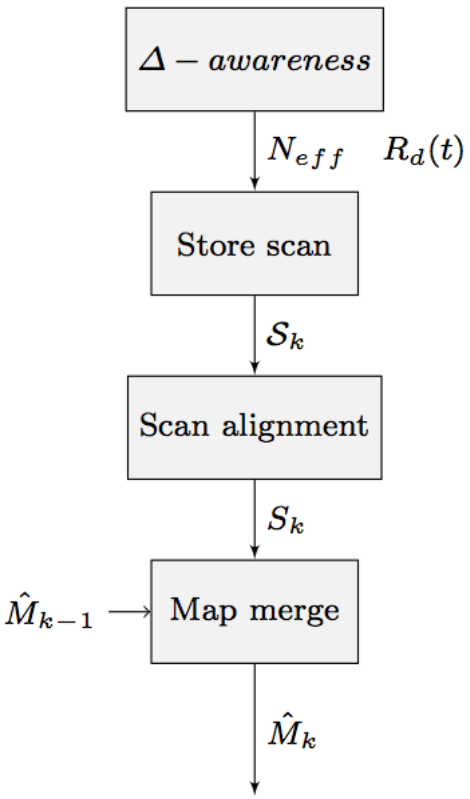


Figure 2.22. The architecture of the approach.

An example of application of this technique can be found in recovering from localization failures [89], where it is used to calculate the empirical measurement likelihood and maintain short-term and long-term averages of this likelihood, deciding when to randomly inject particles to the filter.
In our setting, the weighted recency averaging is employed to recognize changes in the environment, under the hypothesis that the robot is correctly localized and never kidnapped.

In particular, we call $w_{avg}(t)$ the weight of the position hypothesis with the highest likelihood, and we calculate the short-term likelihood and the long-term likelihood as

$$\begin{aligned} w_{slow}(t+1) &= w_{slow}(t) + \alpha_{slow}(w_{avg}(t) - w_{slow}(t)) \\ w_{fast}(t+1) &= w_{fast}(t) + \alpha_{fast}(w_{avg}(t) - w_{fast}(t)) \end{aligned} \tag{2.12}$$

where the parameters α_{slow} and α_{fast} are decay rates for the filters that estimate the long-term and short-term averages. As a rule of thumb, we should choose $\alpha_{slow} \ll \alpha_{fast}$, noticing that these parameters influence directly how quickly the Δ-Awareness

block perceives a variation occurred in the environment.
The $w_{avg}(t)$ is in general a non-stationary process, since at least one of its parameters (e.g. the mean value) changes over time.
In Figure 2.23 it is shown a comparison between the trend of the $w_{avg}(t)$ over time when no modifications in the map occur (a), and when the robot passes near an area with a variation (b). The sudden changes of $w_{avg}(t)$ in the latter case allow to use such process to detect variations.

The divergence between the short-term and the long-term average of the measurement likelihood is considered in the computation of the following *divergence ratio* $r_d(t)$

$$r_d(t) = 1 - \frac{w_{fast}(t)}{w_{slow}(t)} \tag{2.13}$$

which is one of the indexes considered by the Store Scan block to decide if a laser scan should be discarded or not, as discussed in Subsection 2.5.3.

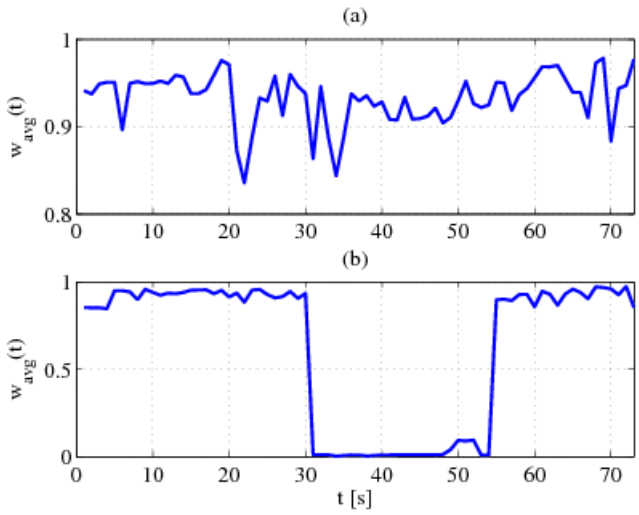


Figure 2.23. Comparison between the trend of the $w_{avg}(t)$ over time when no modifications in the map occur (a), and when (b) the robot passes near a variation.

Store Scan

The purpose of the Store Scan block is to select the laser scans suitable for building the local updated sub maps. These scans are stored in a set called $\mathcal{S}(k)$.
In order to maximize the probability of storing scans when modifications in the map have occurred, if $r_d(t) > 0$ a scan is selected with a probability given by $v < r_d(t)$, where v is a uniformly distributed random variable $U(0,1)$.

Figures 2.24 and 2.25, respectively, clarify what happens when there are no modifications in the map and when a robot perceives a variation in the environment.
In Figure 2.24 the values taken by $w_{fast}(t)$ are higher than those taken by $w_{slow}(t)$;

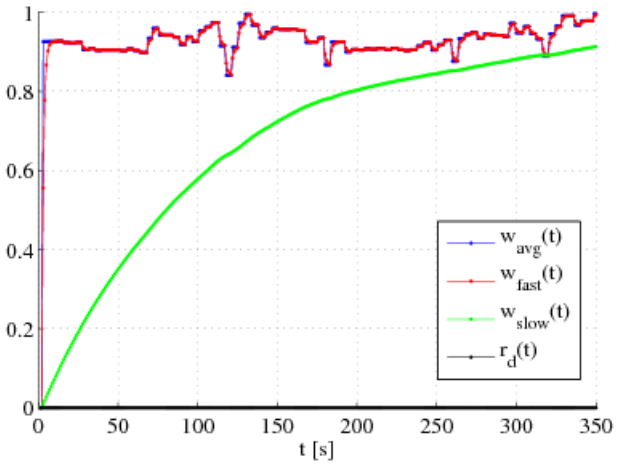


Figure 2.24. Trend of $w_{avg}(t)$, $w_{fast}(t)$, $w_{slow}(t)$, $r_d(t)$ in absence of modifications in the environment.

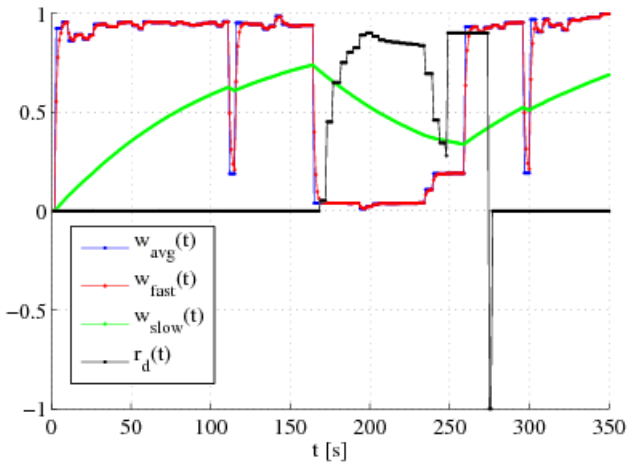


Figure 2.25. Trend of $w_{avg}(t)$, $w_{fast}(t)$, $w_{slow}(t)$, $r_d(t)$ in presence of modifications in the environment.

from (2.13) it follows that $r_d(t)$ is negative (in Figure 2.24 it has been set to zero) and the Δ-mapping process does not begin. In Figure 2.25 the values assumed by $w_{fast}(t)$ are lower than those assumed by $w_{slow}(t)$ and hence $r_d(t)$ is greater than

zero.
The Δ-mapping process begins when $r_d(t)$ becomes positive for the first time, and ends when $r_d(t)$ falls again below zero.
Unfortunately we experienced that this method suffers from a couple of problems.
It can be observed that even if the environment does not change, the *divergence ratio* may increase in specific situations, such as when the rotational velocity of the robot is greater than zero, causing the beginning of the Δ-mapping process also when the environment has not changed.
The second problem is that even if the Δ-mapping process has correctly begun due to a variation in the environment, the quality of the scans acquired by the laser rangefinder could be insufficient.
The accuracy of the scans is related to the precision of the robot pose estimate, in particular to its heading estimation. A significant source of inconsistency is to be found in heading variance which, if large, can cause divergence in just a few updates. In our case, scans acquired with a wrong heading may lead to bad-aligned sub-maps. Moreover, the problem of a bad quality of the scans can be related to the first highlighted matter, i.e. when the robot is rotating.
In fact it frequently happens that the robot encounters map variations while turning, hence when its rotational velocity is greater than zero. In this situation, even if the map has changed, some acquired scans may have a wrong heading, and therefore they should be discarded.
The proposed solution for a proper scan discarding is discussed in the next subsection.

The local degeneracy of the particle filter algorithm is measured by the *effective sample size* N_{eff} of the particle filter defined as

$$N_{\text{eff}} = \frac{N_s}{1 + \text{Var}(w_t^{*i})} \tag{2.14}$$

where N_s is the number of particles and w_t^{*i} is the true weight. N_{eff} cannot be exactly calculated, but an estimate can be obtained by computing

$$\hat{N}_{\text{eff}} = \frac{1}{\sum_{i=1}^{N_s}(w_t^i)^2} \tag{2.15}$$

A relation has been empirically found between the local degeneracy of the particle filter and the moments in which the robot is turning. Three cases have been considered to investigate such a relation. In the first one we compute the values of N_{eff} while the robot is going straight and some environment changes occur. In the second one the robot turns with a constant rotational velocity (without any environmental modification), while in the third one it follows the blue (not straight) path shown in Figure 2.26. The results of computing N_{eff} in the three examples are shown in

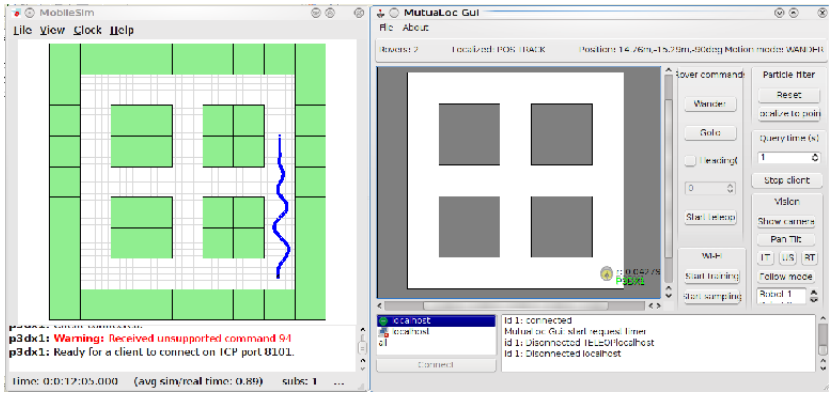


Figure 2.26. Path followed by the simulated robot to evaluate N_{eff} in the third case. On the left there is the simulated robot and environment, on the right our graphical user interface.

Figure 2.27, in blue, red and black, respectively.
The red and black plots show that the robot turning decreases the value of N_{eff}, whereas the variation in the map while the robot is *not* turning does not influence its value. We can thus argue that the evaluation of N_{eff} allows us to reject those scans acquired while the robot is turning, which can be bad aligned with high probability.

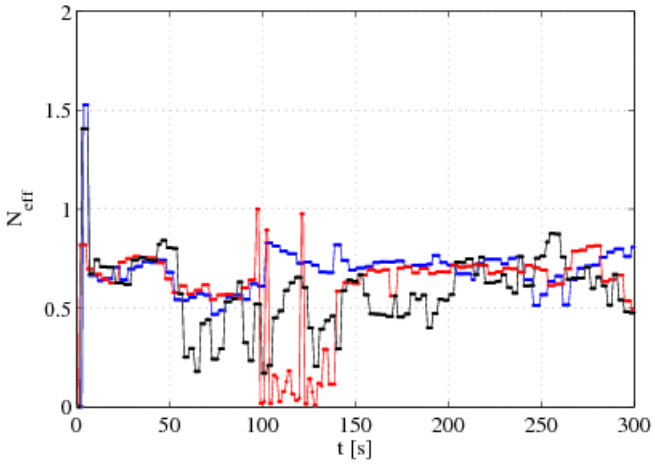


Figure 2.27. Comparison of the N_{eff} trend.

Scan Alignment

The Scan Alignment block produces a local map performing a consistent registration of the collection of scans contained in $\mathcal{S}(k)$. The approach maintains all the local frames of data as well as the relative spatial relationships between local frames, modeled as random variables and derived from matching pairwise scans or from odometry. Then all the spatial relations are combined using the *Rmap* algorithm (see [5]).
The output of this block is a consistently aligned map S_k.

Map Merge

The Map Merge block receives the aligned map S_k provided by the *Scan Alignment* block and merges this map with $\hat{M}_{k-1}$, which is the map the robot is currently using for localization. The output of this merge process is $\hat{M}_k$. We adopted the algorithm proposed in [22], whose theoretical foundations are briefly recalled in the next subsection.

Theoretical background about Map Merging

We assume that a grid map M is a matrix with r rows and c columns. Each cell $M(i, j)$ may contain three different values, indicating whether the cell is free, occupied, or if its status is unknown.
Given two maps $M1$ and $M2$, the goal of map merging is to find a rigid transformation T so that the two maps can be overlapped. The transformation $T = T(\Delta x, \Delta y, \phi)$ is the combination of a rotation ϕ, followed by a translation along the x and y axis of magnitude Δx and Δy, respectively.

The overall transformation T is computed in two separate steps. First the rotation ϕ is determined, and then the translations Δx and Δy are deduced. In order to do this, Hough transform is applied to detect lines expressed in polar coordinates (i.e. ρ and θ). Line detection is performed using the Discretized Hough transform (DHT), that discretizes the domain for ρ and θ, so that the DHT can therefore be represented by a matrix with ρ_S rows and θ_S columns. In addition it is also necessary to set a bound for ρ, while θ is naturally bounded to $[0, 2\pi)$. The DHT can be applied to detect lines in an occupancy grid map M, by converting it into a binary image. The conversion can be performed setting all occupied cells to black, and all other cells to white, for example. If M is a grid map, we indicate its DHT with HT_M. Given HT_M we define its associated Hough Spectrum as the following signal:

$$HS_M(l) = \sum_{i=1}^{\rho_S} HT_M(i, l)^2, \ 1 \leq l \leq \theta_S$$

Translations of the Hough spectra correspond to rotations of the associated map. The computation of the circular cross correlation gives information about how the maps have to be rotated in order to be overlapped. Formally, if HS_{M1} and HS_{M2} are two Hough spectra with the same sampling period, their circular cross correlation CC_{M1M2} is a signal with the same sampling period defined as follows:

$$CC_{M1M2} = \sum_{i=1}^{\theta_S} HT_{M1}(i) HT_{M2}(i+l), \ 1 \leq l \leq \theta_S \tag{2.16}$$

Local maxima in the spectra cross correlation reveal how $M2$ should be rotated in order to align it with $M1$. The proposed algorithm therefore extracts a set of n local maxima (n being a specified parameter), and returns n transformations.

Given a candidate rotation ϕ_i , the corresponding translations Δx^i , Δy^i can be in principle easily determined. Let $M3$ be the map obtained rotating $M2$ of ϕ_i , i.e.

$$M3 = T(0, 0, \phi_i) M2. \tag{2.17}$$

Translations needed to overlap $M3$ to $M1$ can be obtained computing the bidimensional correlation in the following way. First we compute the *X-spectrum* of a binary image M, having r rows and c columns, as follows:

$$SX_M(j) = \begin{cases} \sum_{i=1}^{r} M(i,j) & 1 \leq j \leq c \\ 0 & \text{otherwise} \end{cases} \tag{2.18}$$

Similarly, the *Y-spectrum* of image M is defined as:

$$SY_M(j) = \begin{cases} \sum_{i=1}^{c} M(i,j) & 1 \leq j \leq r \\ 0 & \text{otherwise} \end{cases} \tag{2.19}$$

Given SX_{M1} and SX_{M3}, Δx^i can be easily inferred by looking at the global maximum of the cross correlation between them, defined as

$$CCX_{M1M3}(\tau) = \sum_{l=-\infty}^{+\infty} SX_{M1}(l+\tau) SX_{M3}(l) \tag{2.20}$$

Similarly to the case of correlation between Hough spectra, multiple local maxima may emerge when computing the cross correlation between *X-spectra*. Each of the maxima is associated with a candidate translation to align the two maps and can be individually tracked.

Map Merge Validation

We have performed an intensive test of the Map Merge block, in order to properly set the parameters of the algorithm proposed in [22] and to verify the effectiveness

and speed of the merging algorithm.
To perform the tests we have considered two different parts of the map of Figure 2.28 (used also in the simulation tests). These submaps are shown in Figure 2.29 (a) and (b).

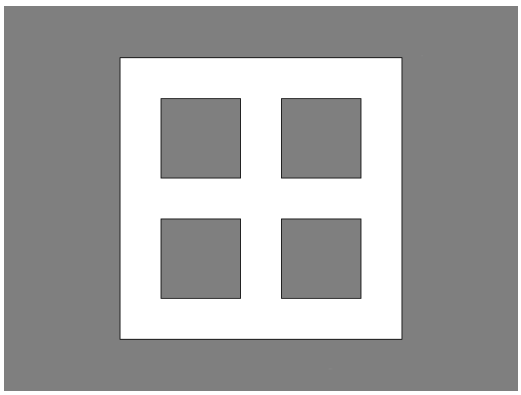


Figure 2.28. The map used to validate the Map Merge block, used also in the simulation tests.

(a) (b)

Figure 2.29. The two parts of map in Figure 2.28 used to validate the Map Merge block.

For each submap we have generated every possible x translation in the range [-10;10] pixel with resolution 1 pixel, for every x translation every possible y translation in the range [-10;10] pixel with resolution 1 pixel, and for every y translation every possible rotation in the range [0;360] degrees with resolution 1 degree. To evaluate the performances of the merging procedure, given two maps $M1$ and $M2$ we consider the following simple acceptance index:

$$w(M1, M2) = \begin{cases} 0 & \text{if agr} = 0 \\ \frac{agr(M1,M2)}{agr(M1,M2)+dis(M1,M2)} & \text{if agr} \neq 0 \end{cases} \tag{2.21}$$

where $agr(M1, M2)$ is the *agreement* between $M1$ and $M2$, the number of cells in $M1$ and $M2$ that are both free or both occupied and $0 \leq w \leq 1$. The *disagreement* between $M1$ and $M2$ (indicated by dis($M1$,$M2$)) is the number of cells such

that $M1$ is free and $M2$ is occupied and vice-versa. Notice that only free or occupied cells are considered, while unknown cells are ignored.
We have considered four possible merging algorithms. The first one (MA1) is the one presented in Section 2.5.3, while the other two are the "robust" versions of the first one.
The second one (MA2) evaluates other candidate solutions rotating the ones computed by MA1 in the range [-0.5;0.5] degrees with a step of 0.25 degrees. The third one (MA3) evaluates other candidate solutions rotating again the ones obtained by MA1 in the range [-1;1] degrees with a step of 0.25 degrees. In the fourth one (MA4) a further translation is added. The other candidates solutions are determined rotating the MA1 solution in the range [-0.25;0.25] degrees with a step of 0.25 degrees, and adding to each rotation a translation of +1 and -1 pixel.
The results in terms of acceptance and computation time of the four merging algorithms are indicated in Table 2.4.

Table 2.4. Results of the four methods.

Methods	Acceptance	Computation Time [s]
MA1	0.9977	0.67
MA2	0.998	1.38
MA3	0.998	1.99
MA4	0.998	2.01

For each method we averaged all the acceptance indexes obtained for each rotation (and also translation for M4). The computation time for each method has been computed as the average among all the single computation times for each rotation/translation. The Map Merge validation has been performed on a 2.5 GHz Intel© Core2 Duo platform with 2 GB of RAM memory.
After the evaluation of the accuracy of the four proposed methods and their computation time, we chose to use the MA1 method in the Map Merge block, because the acceptance obtained is comparable with the other methods, and the computation time is lower than the others.

2.5.4 Simulation Tests

We consider a simulated environment of a logistic area (see Figure 2.28). The occupied black areas can be thought as containers or similar bulky items stored before distribution. The dimension of the whole environment is 35×35 m, the black areas are 10×10 m and the corridors are 5 m wide.
We assume that, when the rover is correctly localized, a virtual fork-lift removes or adds one container every minute.
To demonstrate the effectiveness of the proposed approach we first provide results

related to $nr = 10$ averaged runs, and the Δ-mapping updating process lasts for approximately two hours each run.
We define the localization error of the robot as the distance between the ground-truth Cartesian position $(x_i^{gt}(t), y_i^{gt}(t))$ and its Cartesian position estimation given by (2.11) as

$$e^{\rho}(t) = \sqrt{(x^{gt}(t) - \hat{x}(t))^2 + (y^{gt}(t) - \hat{y}(t))^2}. \tag{2.22}$$

We then define the average localization error over nr runs as

$$\bar{e}^{\rho}_{nr}(t) = \sum_{i=1}^{nr} \frac{e^{\rho}(t)}{nr}. \tag{2.23}$$

The localization error is shown in Figure 2.30. The localization error remains lower than 0.5 m, except for a period of time where the error increases. This is due to the loss of the localization by the rover in one of the runs. However the robot quickly recovers correct localization.

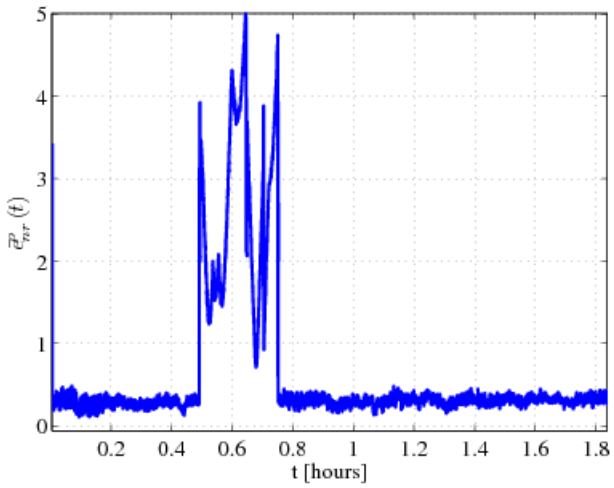


Figure 2.30. Localization error of a robot in nr runs.

Table 2.5 shows the acceptance index (2.21) mediate over the single run, along with the number of variations occurred detected per run.
The number of detected variations in each run depends on the path followed by the robot. In this work the robot wanders in the environment, and there is no active mechanism to ensure the detection of *all* the variations occurred in the environment. Future works will be devoted to enhance the motion strategy. The average quality of the map is comparable with the quality of a map obtained by a Rao-Blackwellized

Run	Acceptance	Number of variations
R1	0.979	23
R2	0.98	30
R3	0.9695	68
R4	0.9758	77
R5	0.9728	97
R6	0.9784	47
R7	0.9804	62
R8	0.9731	200
R9	0.9681	197
R10	0.97	100

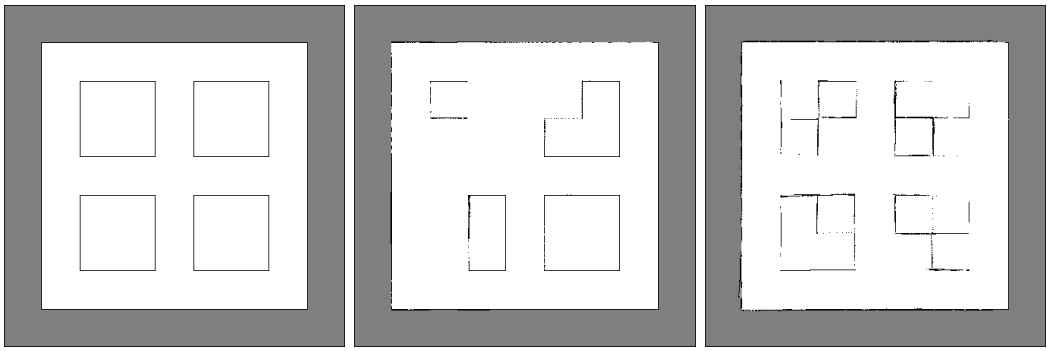
Table 2.5. Acceptance values and number of variations of the *nr* runs.

SLAM process in static conditions, since in that case the value of the index obtained is 0.98.

In another test, we have performed a Δ-mapping process lasting for approximately seven hours, for a total number of 420 variations, to check the long operativity performance. Figure 2.31 shows the status of the grid map respectively after few variations and after many variations. The map shown in Figure 2.31 (c) contains open squares because sometimes the robot may not be able to completely map a variation. Moreover, the a-priori knowledge of the geometric aspect of the elements inside the environment (e.g., the goods shape) is not used to complete the final map obtained by the proposed procedure.
In Figure 2.32 the localization error of the simulated rover is shown. Figure 2.38(b) shows the acceptance index (in the range 0-1 from worst to best) related to the map quality over time. After seven hours of operation the rover localization error remains acceptable (below 1 m) and the quality of the map is still comparable with the quality of a map obtained by a Rao-Blackwellized SLAM process.

2.6 Extension to the multi-robot case

A team of mobile robots, each endowed with a laser rangefinder and wireless connectivity, is supposed to be correctly localized with respect to the available environment map. In particular, each robot is assumed to be in the *position tracking* state, as defined in [11] and [12].
Each robot uses an occupancy grid map of the environment in the localization algorithm to track its position over time. Such a map could have been manually created or previously built by a SLAM algorithm.
At discrete time instants k the environment changes, and consequently the robots



(a) (b) (c)

Figure 2.31. The initial map (a), the map after few variations (b), and the map at the endo of the Δ-mapping process.

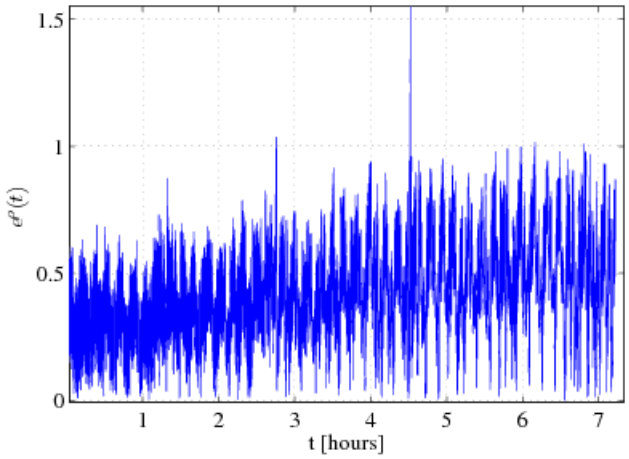


Figure 2.32. The localization error in the long operativity test.

have to modify their map to take into account the variation. This phase is called Δ-*mapping* step.
The set of new maps collected up to time k is defined as

$$\mathcal{M}(k) = \{M_k\},\ k = 0, \ldots, K.$$

M_0 is the initial map, obtained by the SLAM procedure. The goal of the developed algorithm is to provide for each robot an estimate $\hat{M}_k$ of the map at each time step k. In order to take advantage of the multi-robot scenario these updated maps must be spread among the other robots, and this information has to be merged in order to create a map that is a good estimate of the current state of the environment.

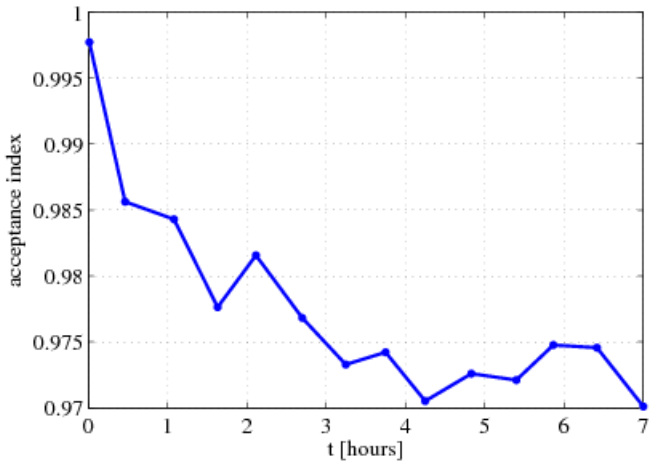


Figure 2.33. The trend of the quality of the map over time.

Correct map merging is not sufficient; a coordination strategy of the team of robots it also needed in order to maximize the number of detected variations, balancing at the same balancing the number of Δ-*mapping* processes among the robots.

2.6.1 The Approach

The guidelines of the proposed approach are described hereafter, whereas details about the specific processes of variations awareness, local Δ-*mapping* and map merging are given in subsection 2.6.2.

In the proposed Δ-*mapping* approach the concept of *time-map* is introduced to merge in an appropriate way the changes detected locally by a robot and the updated maps received from the other team members. In a grid map each cell represents the belief on the occupation value of the corresponding area. Since the environment changes over time, the reliability of the stored value for the cells decreases over time. Therefore to each cell in the map a value in the range $[0 - 1]$ is assigned, related to the time passed since the cell has been visited for the last time. The set of these values at each time step is called *time-map* and defined as T_t.

The outline of the Δ-*mapping* algorithm, which runs on board of each rover, is described in Algorithm 4.

The algorithm takes as inputs the previous map $\hat{M}_{k-1}$ and the time-map T_{t-1}. p and l are the current robot pose and the current laser range reading respectively,

Input: $\hat{M}_{k-1}$, T_{t-1}, p, l, P, L
Output: $\hat{M}_k$, T_t

T_t = **updateTimeMap**(T_{t-1}, p, l);
if *received map* $\hat{M}'$ T' **then**
  $[\hat{M}'_{k-1}, T_t]$ = **mergeMap**($\hat{M}_{k-1}$, T_t, $\hat{M}'$, T');
  $\hat{M}_{k-1} = \hat{M}'_{k-1}$;
end
if $\Delta - awareness$ **then**
  $P = P + p$;
  $L = L + l$;
else
  if $P! = \emptyset$ **then**
    $[\hat{M}_k]$ = **updateMap**($\hat{M}_{k-1}$, P, L);
    $P = \emptyset$;
    $L = \emptyset$;
    dispatchUpdatedMap($\hat{M}_k$, T_t);
  end
end

Algorithm 4: the Δ-mapping algorithm

P and L are two matrices in which the values of p and l are collected as

$$P = \begin{bmatrix} \hat{x}^1, \hat{y}^1, \hat{\theta}^1 \\ \vdots \\ \hat{x}^n, \hat{y}^n, \hat{\theta}^n \end{bmatrix} \tag{2.24}$$

$$L = \begin{bmatrix} l^1 \\ \vdots \\ l^n \end{bmatrix} \tag{2.25}$$

where the n-th entry is the last element stored. These matrices are used to create a local Δ-map containing the changes in the environment detected by the robot.

The time-map T_t is updated every time a laser scan is available to the robot; a ray tracing procedure is applied for each angle of the scan, assigning a maximum value equal to 1 to every cell crossed by a ray. At each time step all the values in T_t are updated according to (2.26), where Δt is the time elapsed from the last update of T_t, and C_t is a time constant which defines the forgetting speed.

$$T_t(i,j) = T_{t-1}(i,j) \cdot \left(1 - \frac{\Delta t}{C_t}\right) \tag{2.26}$$

The time-map update depends only on Δt, therefore a common timebase among the team members is not required, avoiding the need of synchronization techniques over the net, such as Network Time Protocol (NTP).

The algorithm is then divided into two parts. The first part (lines 2-4) is performed only when the robot receives a map from another robot member of the team, while the second part (lines 6-15) is performed only if a variation in the environment has been detected.

If the robot receives a new map $\hat{M}'$ and the relative time-map T', it updates the state of its map and its time-map by merging them with $\hat{M}_{k-1}$ and T_t respectively (line 3). At this point the resulting map contains the modifications perceived by the other robots (line 4).

If a modification is detected by the Δ-*awareness* block, recalled in Section 2.6.2, the algorithm stores the current robot pose and the relative laser range reading (lines 7-8).

If the Δ-*awareness* block does not detect any modification and P and L are not empty, a local Δ-*mapping* is performed, following the approach recalled in subsection 2.6.2 (line 10). The content of these vectors is used to create a local Δ-map $\Delta\hat{M}$, then $\Delta\hat{M}$ is aligned and merged with the old map $\hat{M}_{k-1}$, in order to obtain an updated map $\hat{M}_k$.

Finally the resulting map $\hat{M}_k$ and the current time-map T_t are dispatched to the other team members.

2.6.2 Δ-awareness, local Δ-mapping and map merging

In [13] the authors presented an approach to maintain an updated grid map of a dynamic environment for a single robot, where an initial occupancy grid map is supposed to be available. The algorithm is able to detect persistent variations in the environment and merge this variations with the previous map by using limited computational resources and is composed by four blocks as shown in Figure 2.22.

The Δ-*awareness* block detects persistent variations in the environment, using a technique called *weighted recency averaging*, which is normally applied in problems of tracking non-stationary processes.
In this setting, the weighted recency averaging is employed to recognize changes in the environment, under the hypothesis that the robot is correctly localized and never kidnapped.

The purpose of the *Store Scan* block is to select the laser scan readings suitable for building the local updated sub maps. These readings are stored in L with the corresponding robot poses stored in P.

The *Scan Alignment* block produces a Δ-map performing a consistent registration of the collection of scan readings contained in L. The approach maintains all the local frames of data as well as the relative spatial relationships between local

frames, modeled as random variables and derived from matching pairwise scans or from rover poses stored in P.

The *Map Merge* block merges the output of the *Scan Alignment* block with the map $\hat{M}_{k-1}$. The goal of this block is to find a rigid transformation in order to overlap Δ-map and $\hat{M}_{k-1}$ to create the current environment occupancy map $\hat{M}_k$. We adopted the algorithm proposed in [22], which uses Discretized Hough transform and bidimensional correlation. The Discretized Hough transform allows to find the rotation that aligns Δ-map with $\hat{M}_{k-1}$, then the bidimensional correlation is applied to compute the translation to overlap the two maps.

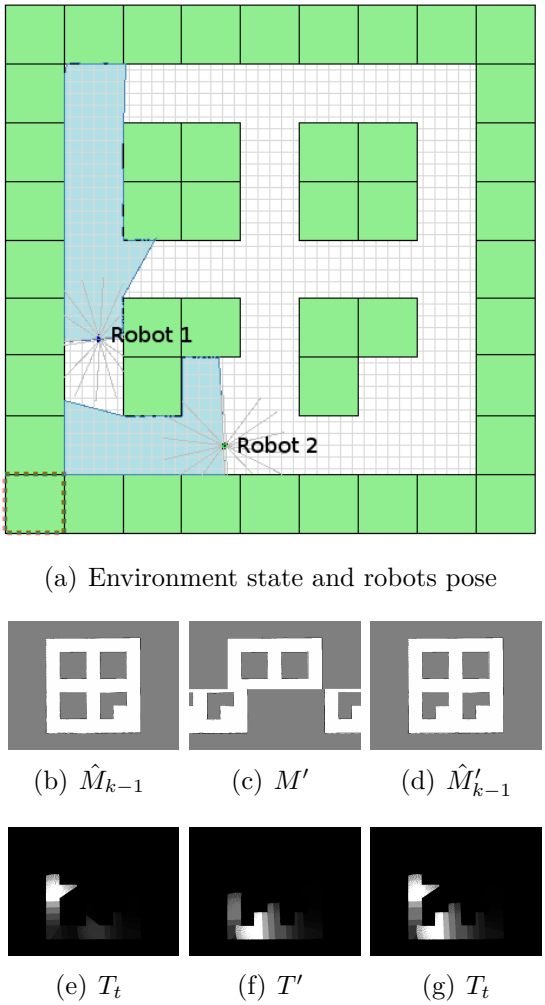
Local Δ-*mapping* in this work consists in the application of the *Scan Alignment* and *Map Merge* blocks.

In the *updateTimeMap* function in line 3 of Algorithm 4 the current maps $\hat{M}_{k-1}$ and T_t are updated according to M' and T' received from the other robots. For all couples i, j every cell $\hat{M}_{k-1}(i, j)$ is updated if its value is older than the corresponding cell $\hat{M}'(i, j)$, so that the most reliable value is used. The information about the reliability is given by the time-maps T_t and T'.

When a robot receives a new map $\hat{M}'$ and a time-map T' from another teammate, it merges it with the previous map $\hat{M}_{k-1}$ and the local time-map T_t in order to produce $\hat{M}'_{k-1}$. T_t is also updated. For all couples i, j the value of the cell $\hat{M}'_{k-1}(i, j)$ is set equal to the cell $\hat{M}'(i, j)$ if $T'(i, j) > T_t(i, j)$, otherwise it is set equal to $\hat{M}_{k-1}(i, j)$. The value of the cell $T_t(i, j)$ is set equal to $T'(i, j)$ if $T'(i, j) > T_t(i, j)$, otherwise it is not modified. Figure 2.34 shows the map merging in a typical case. It can be noticed how changes received from another robot and local changes detected by the local Δ-*mapping* are both merged in a consistent way. This happens because cells belonging to areas that have been recently mapped have high corresponding time-map values (close to 1), so recent changes in the map resulting from a local Δ-*mapping* process are not discarded.

2.6.3 Exploration strategy

A team coordination strategy that actively searches modifications in the map has been developed. Without any coordination strategy all the robots could follow the same path or leave some areas unexplored for a long time. This problem can be treated in partial similarity with the problem of multi-robot exploration. In the exploration approaches the aim is to discover a map starting from a completely unknown environment. In our case the initial map is known, as well as the robot pose, but since the environment is persistently changing (pallets are added and removed), the reliability of the initial map decreases over time on the basis of the number of changes in the environment. For this reason, areas that have not been recently visited may become completely unknown, as the reliability of the map in those areas is very low.



(a) Environment state and robots pose

(b) $\hat{M}_{k-1}$ (c) M' (d) $\hat{M}'_{k-1}$

(e) T_t (f) T' (g) T_t

Figure 2.34. Figures show the map merging in a typical case: 2.34(a) shows the pose of the robots, Robot 1 receives a map from Robot 2 and it uses it to update its map; 2.34(b) is the current map, 2.34(e) is the current time-map, 2.34(c) is the received map, 2.34(f) is the received time-map, 2.34(d) and 2.34(g) are the resulting map and time-map after the merging process.

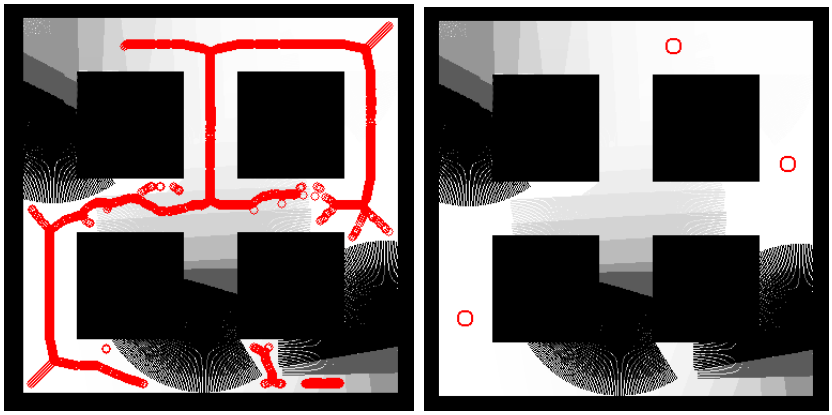
Areas that need to be explored are then the ones for which the corresponding value of the time-map is below a given threshold. For each robot, a set of points is extracted to feed the path planning algorithms from a topological map, which is constructed from the grid-map representing the areas to be explored.

Many approaches exist to obtain a topological representation from a grid-map, such as Voronoi diagrams or topological operations. A morphological skeleton representation of the map is extracted. The skeleton of an image is a good representation of the geometrical and topological properties of its shape.

Then a set of points belonging to the skeleton is identified, with the constraint that each point has to be at a minimum distance from every other point.

Each point becomes one goal point for a suitable path planning algorithm: in this case the wavefront algorithm [56] is used. These goal points are then allocated to the team members by a distributed market-based task allocation algorithm described in the following subsection.

Figure 2.35 shows how the goal points are obtained. The time-map has inverse colors (black cells have the highest reliability and white ones have the lowest reliability). In Figure 2.35(a) red points belong to the skeleton of the areas with reliability below a given threshold. In this case the team is composed by three robots, so three goal points are obtained as indicated in Figure 2.35(b).



(a) (b)

Figure 2.35. Time-map and skeleton of areas to explore (a) and final goal points for three robots (b)

2.6.4 Distributed auction-based task allocation

Each goal point generated by the exploration strategy must be efficiently assigned to one of the robots in order to minimize travel time.

A feasible way is the so called *Hungarian method* that performs a combinatorial optimization to solve the assignment problem in polynomial time. It guarantees the optimal solution, but it is a centralized algorithm, that requires a supervisor node and a matrix containing a row for each robot and a column for each task. Each cell contains the cost for the relative task. Moreover this approach requires the ability for all the robots to communicate, but this condition is not assured due to unreliable WiFi communication.

The used approach is then based on auctions, and it has been developed starting from the one proposed in [92]. Every goal point is assigned to an auction over a multicast network channel; the robots that receive the auction compute and send back a bid. The auctioneer assigns the task to the robot with the best bid. The bid is computed according both to the robot's current position and to its queue of pending tasks. This approach does not guarantee the optimal solution, but it is robust to communication failures. Moreover, the auctioneer is always a different robot, thus avoiding the problem of single point of failure.

2.7 Simulation Tests

The simulated environment of a logistic area already used in [13] and shown in Figure 2.36 is considered. The occupied green areas can be thought as containers or similar items stored before distribution. The environment is 35×35 m, the green areas in the center are 10×10 m and the corridors are 5 m wide.

$n = 3$ rovers are endowed with wheel encoders, a laser range finder and a WiFi

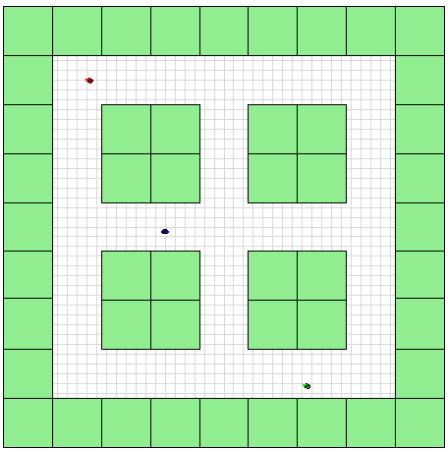


Figure 2.36. The simulation environment.

board, and are able to localize themselves in the given environment. It is assumed

that, once the rovers are correctly localized, a virtual fork-lift adds or removes one container every minute.
The rovers start moving with a simple obstacle avoidance policy. Then when the Δ-*mapping* process starts the rovers move according to the exploration strategy described in Section 2.6.3. The quality of the map over time and the localization error are measured. The error on the estimate of the robot pose is strictly related to the quality of the map. Every Δ-*mapping* process induces some degradation of the map, due to the localization error which cannot be fully compensated by the *Map Merge* block.

Even after a consistent number of changes in the environment the rovers keep a map that is consistent with the environment and therefore the localization error remains low.

2.7.1 Simulation test 1

To demonstrate the effectiveness of the proposed approach first results related to $r = 10$ averaged runs are provided, where the Δ-*mapping* updating process lasts for approximately two hours each run.
The localization error of the i-th robot is defined as the distance between the ground-truth Cartesian position $(x_i^{gt}(t), y_i^{gt}(t))$ and its Cartesian position estimation as

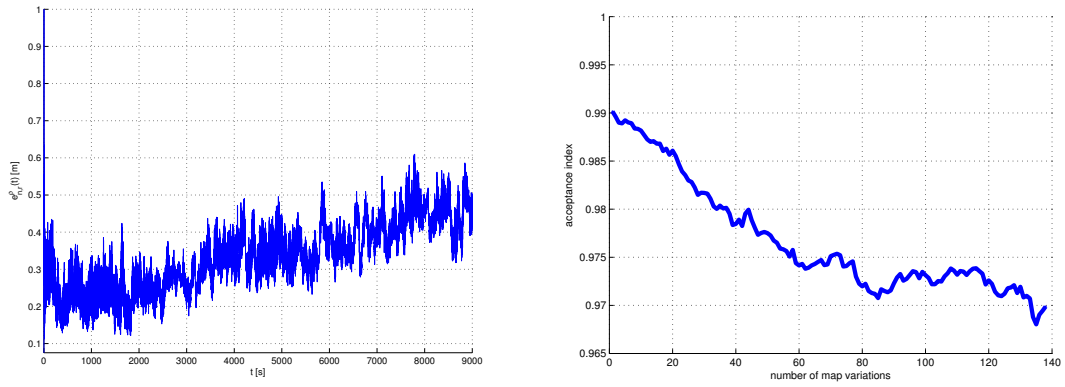
$$e_i^{\rho}(t) = \sqrt{(x_i^{gt}(t) - \hat{x}_i(t))^2 + (y_i^{gt}(t) - \hat{y}_i(t))^2}. \tag{2.27}$$

We then define the average localization error for n robots over r runs as

$$e_{n,r}^{\rho}(t) = \frac{1}{r} \sum_{j=1}^{r} \sum_{i=1}^{n} \frac{e_i^{\rho}(t)}{n} \tag{2.28}$$

The localization error is reported in Figure 2.37(a). It can be noticed that the mean localization error remains lower than 0.6 m after approximately 2.5 hours. The quality of the map for the duration of the test is also inspected. Visual inspection is often used, and numerical results by using the acceptance index described in [22] are also provided. They can be used as a measure of similarity between the real map and the estimated map.

Figure 2.37(b) shows the acceptance index mediate over the $n = 3$ robots and over $r = 10$ runs. After 140 variations the value obtained is 0.97, which is comparable with the one obtained with a typical grid-based SLAM algorithm (0.98).



(a) (b)

Figure 2.37. Localization error and acceptance index for test 1.

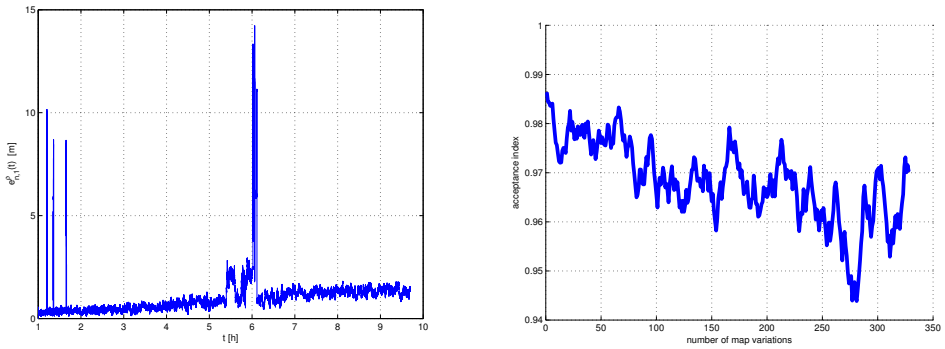
2.7.2 Simulation test 2

Here the performances of the Δ-*mapping* process in long term operativity are tested. The simulation scenario is the same as for the previous test, but the virtual fork-lift adds and removes containers every two minutes. In this test the map updating process lasts for approximately 9.5 hours, for a total number of 328 variations. Figure 2.38(a) shows the localization error for a single run, while Figure 2.38(b) shows the acceptance index over 328 variations. The sudden increase of the localization error at approximately 6 hours is due to one of the robots losing its localization for a short period. However as the robot receives an updated map it is able to recover itself. After 328 variations the acceptance index is still comparable with the one obtained in the previous test (see figure 2.7.1). Moreover, this acceptance index decreases to 0.97 after 9.5 hours, while in [13] the same error occurs after only 6 hours.

2.7.3 Simulation test 3

In this test the $n = 3$ robots perform different actions. The first robot performs Δ-*mapping* and sends map variations to the others team members; the second one only receives map variations but does not perform Δ-*mapping*; the last one neither perform Δ-*mapping* nor receives changes from the other team members. This test is aimed at demonstrating the advantage in receiving map updates from other robots.

Figure 2.39 shows the localization error $e^{\rho}(t)$ for the three robots during a single run. Robot 1 remains well localized, while for robot 3 the error increases after approximately 3800 seconds; localization error for robot 2 starts to increase after 4720 seconds. This is due to the fact that robot 2 is able to merge the map updates



(a) (b)

Figure 2.38. Localization error and acceptance index for test 2.

received from robot 1, but this is still not sufficient in order to maintain a consistent map of the environment.

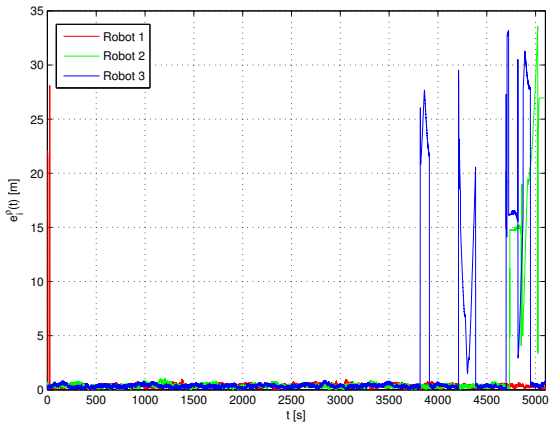


Figure 2.39. Localization error for test 3.

2.7.4 Computational load

In Figure 2.40 the CPU usage and memory occupation for each robot are reported. The algorithm runs on an Intel Core 2 Duo 2.4 Ghz with 2 GB of RAM. After approximately one minute the simulated fork lift starts to remove and add pallets, and the Δ-*mapping* process starts. The peaks in CPU usage and memory occupation

refer to the end of each local Δ-*mapping*, while the peaks in CPU usage only refer to the computation and assignment of the exploration points. It can be noticed that after the beginning of the Δ-*mapping* process the memory usage steadily increases by only 12 MB, with peaks corresponding to the last phase of each local Δ-*mapping* process.

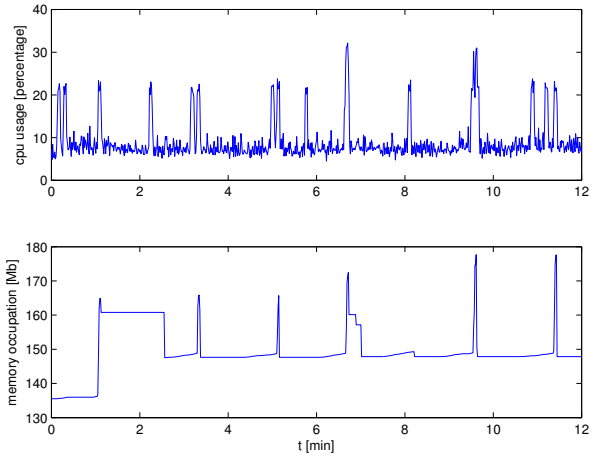
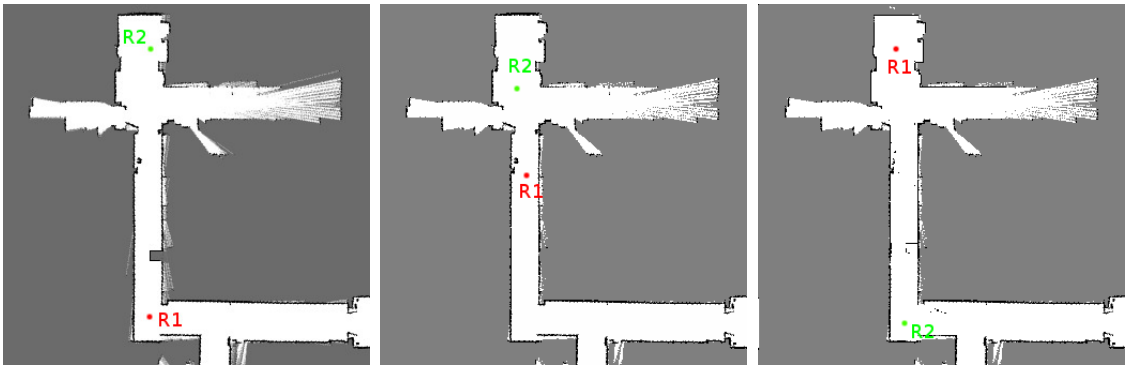


Figure 2.40. CPU usage (upper plot) and memory usage (lower plot) in a simulated experiment.

2.8 Experimental Tests

An experiment in a real environment using two Pioneer P3DX robots has been carried out. Each robot is endowed with a SICK LMS200 laser rangefinder and a WiFi board. A 1×1 m box has been placed in a 30×3 m corridor, and a classical Rao-Blackwellized SLAM process is first performed to obtain the map of the environment, as shown in Figure 2.41 (a). Then, the box is removed, R1 detects the absence of the box while travelling in the corridor, performs a Δ-*mapping* process and dispatches the map to R2, which updates its map (see Figure 2.41 (b)). Finally, the box is placed again in the previous place and R2 detects the presence of the box while travelling in the corridor, performs a Δ-*mapping* process and dispatches the map to R1, which updates its map (see Figure 2.41 (c)). It is worth noting that maps in Figure 2.41 (a),(b),(c) are the same for R1 and R2, even if they have not all perceived the same variations at the same time.
This preliminary test demonstrates the effectiveness of the proposed methodology in a simple but real scenario, since robots are able to merge the received maps from

team members in a consistent way.



(a) Initial map

(b) Dispatched map after the first change

(c) Dispatched map after the second change

Figure 2.41. Results for the experimental test.

Chapter 3

Simultaneous Localization and Mapping

3.1 Introduction

The problem of learning maps is one of the fundamental problems in mobile robotics. Models of the environment are needed for a series of applications such as transportation, cleaning, rescue, and various other service robotic tasks. Learning maps passively, i.e. by perceiving sensor data only, requires the solution of two tasks, which are mapping, and localization. Mapping is the problem of integrating the information gathered with the robot's sensors into a given representation. It can intuitively be described by the question "What does the world look like?" Central aspects in mapping are the representation of the environment and the interpretation of sensor data. Localization and mapping cannot be solved independently of each other. Whenever robots operate in the real world, their observations and motion are affected by noise. Building accurate spatial models under those conditions is widely known as the *Simultaneous Localization And Mapping* (SLAM) problem.

SLAM problems arise when the robot does not have access to a map of the environment or its own poses. Instead, all it is given are measurements $z_{1:t}$ and controls $u_{1:t}$. The term "simultaneous localization and mapping" describes the resulting problem: In SLAM, the robot acquires a map of its environment while simultaneously localizing itself relative to this map. SLAM is significantly more difficult than all robotics problems discussed thus far: It is more difficult than localization in that the map is unknown and has to be estimated along the way. It is more difficult than mapping with known poses, since the poses are unknown and have to be estimated along the way [86].

There are two main forms of the SLAM problem, which have both been thoroughly addressed by the scientific community. One is known as the online SLAM

problem: It involves incrementally estimating the posterior over the current pose along with the map:

$$p(x_t, m|z_{1:t}, u_{1:t}) \tag{3.1}$$

where x_t is the robot pose at time t, m is the map, $z_{1:t}$, $u_{1:t}$ are the measurememts and the controls. This is typically the case of a robot that acquires proprioceptive and exteroceptive measurements from the environment and, at each time step, incrementally includes such information in the posterior describing the robot pose and a representation of the environment.

The second SLAM problem is called the full SLAM problem. In full SLAM, the objective is to calculate a posterior over the entire path $x_{1:t}$ along with the map, instead of just the current pose:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) \tag{3.2}$$

Full SLAM approaches tackle the problem of retrieving the whole trajectory posterior, taking into account at the same time all the available measurements; this batch solution may occur, for instance, when the data, acquired during robot operation, have to be processed off-line to produce a meaningful representation of the scenario.

From the algorithmic point of view, literature on SLAM can be roughly divided in three mainstream approaches: Gaussian filter-based, Particle filters-based and graphical approaches. The first category includes Extended Kalman Filter SLAM (EKF-SLAM) [82], Sparse Extended Information Filter [87], delayed-state EKF-SLAM [35].

3.1.1 Online SLAM

EKF-like techniques require the linearization of both process and measurement models; the first-order approximation, however, often comes at a price of inconsistency and divergence [47]. Moreover, the complexity of the filter is quadratic in the number of landmarks, hence a naive implementation prevents large scale mapping [86]. Finally, the effects of partial observability in standard EKF-SLAM implementation are not fully understood, this being witnessed by recent results [46].

The second class of approaches, namely Rao-Blackwellized particle filters [68], is based on a factorization of the posterior and was successfully employed for building both landmark-based and occupancy grid map representations of the environment [86]. Also in this case, fundamental limitations are well known to the robotic community [61]: a finite number of particles may not be able to correctly represent the trajectory posterior, and this issue becomes critical when the uncertainty in the posterior increases. Grisetti et al. [44], proposed a solution based on particle filters called *gmapping*, which reduces the number of particles needed by computing an

accurate proposal distribution taking into account not only the movement of the robot but also the most recent observations. It has since become one of the most widely used approaches for robotic mapping.

While EKF-SLAM and FastSLAM are the two most important solution methods, newer alternatives, which offer much potential, have been proposed including the use of the information-state form [34].

3.1.2 Full SLAM

The third category encompasses the techniques in which measurements acquired during robot motion are modeled as constraints in a graphical model. The poses assumed by the robot during its motion correspond to vertices of a directed graph, whereas the constraints are modeled as edges in such graph. Hence SLAM can be stated in terms of maximum likelihood and solved through nonlinear optimization techniques [58]. Although graph-based approaches are recognized to outperform other SLAM techniques in terms of accuracy, they present the classical drawbacks of nonlinear optimization techniques: convergence to a global minimum of the cost function cannot be guaranteed in general and, if the initial guess for optimization is outside the basin of attraction of the global optimum, the iterative process is likely to be stuck in a local minimum.

3.2 Graph-based SLAM

Pose graph optimization has recently emerged as an effective solution for SLAM. In a *pose graph*, each node represents a pose assumed by a mobile robot at a certain time, whereas an edge exists between two nodes if a relative measurement (inter-nodal constraint) is available between the corresponding poses. Inter-nodal constraints are usually obtained by means of proprioceptive sensors (*odometry*) or exteroceptive sensor-based techniques (*scan matching*, etc.). Given these contraints, the objective of pose graph optimization is to estimate the configuration of the nodes' poses (*pose graph configuration*) that maximizes the likelihood of inter-nodal measurements. A solution to the problem of retrieving the constraints based on sensor data acquired by the robot is often referred to as the SLAM *front-end*. The module in charge of correcting the trajectory of the robot by means of graph optimization techniques in order to obtain a consistent trajectory is often referred to as the SLAM *back-end*.

The advantage of graph-based SLAM approaches stems from the fact that, once the trajectory has been estimated, it is then easy to build a map of the environment by simply attaching laser scans to the corresponding estimated poses. The approach was first proposed in [58]. After that, several authors put their efforts in enhancing computational time and accuracy of algorithms for pose graph optimization. Thrun

and Montemerlo [88] enabled the estimation of large maps by reducing the problem and then using conjugate gradient optimization. Konolige [50] stressed the problem of large-scale mapping, by exploiting the structure of the graph. Frese et al. proposed a multilevel relaxation approach for full SLAM [41].

Olson et al. [70] introduced an efficient method to pose graph optimization by applying stochastic gradient descent. A tree-based implementation called TORO was proposed by Grisetti et al. [45]. In [52] the authors introduced a flexible framework for the optimization of graph-based nonlinear error functions. Some attempts to exploit the mathematical structure of the optimization problem and the properties of the underlying graph have been proposed in [33] and [32]. Recently, Sünderhauf et al. [84] introduced the idea of switchable constraints for robust pose graph optimization, and proposed a strategy for discarding outliers during the optimization. All the aforementioned techniques are iterative, in the sense that, at each iteration, they solve a local convex approximation of the original problem, and use such local solution to update the configuration. This process is then repeated until the optimization variable converges to a local minimum of the cost function. In particular, when a linear approximation of the residual errors in the cost function is considered, the problem becomes an unconstrained quadratic problem and the local correction can be obtained as solution of a system of linear equations, known as normal equation [32]. In such a case the source of complexity stems from the need of repeatedly solving large scale linear systems. Moreover, all mentioned techniques require the availability of an initial guess for nonlinear optimization, which needs be sufficiently accurate for the technique to converge to a global solution of the problem. A partial answer to this two problems (computational complexity and need of an accurate initial guess) came from the work [20]. In [20] the authors proposed a linear approximation for the pose graph configuration, assuming that the measurement covariance matrices have a block diagonal structure. Roughly speaking, each inter-nodal measurement describes a relative pose between two nodes in the pose graph, and the work [20] assumes that the relative position and relative orientation measurements (that together give the relative pose measurement) are uncorrelated. The approach requires no initial guess and is shown to be accurate in practice.

While a large amount of work has been done on graph optimization, less work has been devoted to the implementation of robust front-ends for the creation of the actual graphs for real-time applications involving mobile robots using traditional distance sensors, such as laser range finders [51, 59]. In particular the graph optimizer assumes that all constraints are correct. A number of consistency checks are thus required to minimize the number of false detections. For example in an office-like environment there can be lack of significant features along a corridor or there can be ambiguity due to symmetries in the environment and repetitive structures.

In this thesis work we propose two contributions: (i) we propose a linear approximation to the problem of graph optimization in the 2D case, extending the approach

previously proposed in [20], which is shown to be faster than most state-of-the-art iterative approaches, while being as accurate in practice, to the case of full covariance matrices. (ii) we propose an implementation of a laser-based front-end for the creation of the pose graph using the Robot Operationg System (ROS) framework. Finally, the front-end and the back-end are integrated into a full graph-SLAM solution, which is validated in simulation scenarios, real scenarios and on standard benchmarking datasets.

3.2.1 Problem Formulation

The objective of pose graph optimization is to provide an estimate of the poses assumed by the robot, namely $X = \{x_0, \dots, x_n\}$, that maximizes measurements likelihood. X is called the *configuration* of poses; the $n+1$ poses are in the form $x_i = [p_i^\top \ \theta_i]^\top \in \mathrm{SE}(2)$, where $p_i \in \mathbb{R}^2$ is the Cartesian position of the i-th pose, and θ_i is its orientation. The available measurements usually describe the relative pose between pairs of nodes, for instance $\xi_{i,j} = p_j \ominus p_i$.

For instance a measurement $\bar{\xi}_{i,j}$ between node i and node j is in the form

$$\bar{\xi}_{i,j} = \xi_{i,j} + \epsilon_{i,j} = \begin{bmatrix} R_i^\top (p_j - p_i) \\ \theta_j - \theta_i \end{bmatrix} + \begin{bmatrix} \epsilon_{i,j}^{\Delta} \\ \epsilon_{i,j}^{\delta} \end{bmatrix}, \tag{3.3}$$

where $\xi_{i,j}$ is the true (unknown) relative pose between node i and node j, $\epsilon_{i,j} \in \mathbb{R}^3$ is the *measurement noise*, $R_i \in \mathbb{R}^{2\times 2}$ is a planar rotation matrix of an angle θ_i and $\epsilon_{i,j}^{\Delta}$ and $\epsilon_{i,j}^{\delta}$ are the (possibly correlated) *Cartesian* and *orientation noise*. According to related literature, we assume $\epsilon_{i,j}$ to be zero mean Gaussian noise, i.e., $\epsilon_i^j \sim \mathcal{N}(\mathbf{0}_3, P_{i,j})$, being $P_{i,j}$ a 3 by 3 covariance matrix. $\xi_{i,j}$ describes the relative transformation that leads pose i to overlap with pose j. We can rewrite each measurement as $\bar{\xi}_{ij} = [(\bar{\Delta}_{i,j}^l)^\top \ \check{\delta}_{i,j}]^\top$, where $\bar{\Delta}_{i,j}^l \in \mathbb{R}^2$ denotes the *relative position* measurement, and $\check{\delta}_{i,j} \in \mathrm{SO}(2)$ denotes the *relative orientation* measurement. The superscript l in $\bar{\Delta}_{i,j}^l$ remarks that the relative position vector is expressed in a local frame. By convention the pose of the first node is assumed to be the reference frame in which we want to estimate all the other poses, i.e., $x_0 = [0\ 0\ 0]^\top$. The idea behind the pose graph formulation is shown in Figure 3.1.

In [20] the authors showed that the relative orientation measurements can be made linear by adding a suitable multiple of 2π, i.e, $\langle \theta_j - \theta_i \rangle_{2\pi} = \theta_j - \theta_i + 2k_{i,j}\pi$, with $\theta_i, \theta_j \in \mathbb{R}$, and $k_{i,j} \in \mathbb{Z}$ ($k_{i,j}$ is called *regularization term*), thus rewriting 3.3 as

$$\bar{\xi}_{i,j} = \xi_{i,j} + \epsilon_{i,j} = \begin{bmatrix} R_i^\top (p_j - p_i) \\ \langle \theta_j - \theta_i \rangle_{2\pi} \end{bmatrix} + \begin{bmatrix} \epsilon_{i,j}^{\Delta} \\ \epsilon_{i,j}^{\delta} \end{bmatrix}, \tag{3.4}$$

where $\langle \cdot \rangle_{2\pi}$ is a modulo-(2π) operator that forces angular measurements in the manifold $\mathrm{SO}(2)$,

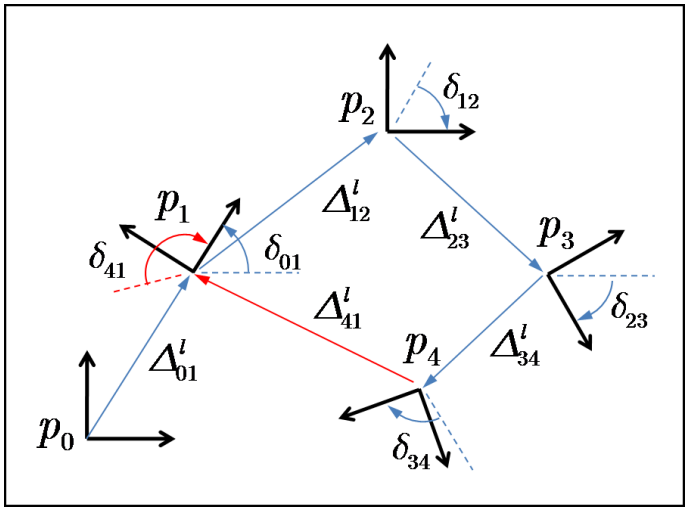


Figure 3.1. The pose graph.

Now we assume that the regularization terms have been correctly computed, according to [20], and we call $\bar{\delta}_{i,j}$ the regularized measurements, i.e., we define $\bar{\delta}_{i,j} = \check{\delta}_{i,j} - 2k_{i,j}\pi$. Then the measurement model becomes:

$$\begin{bmatrix} \bar{\Delta}^l_{i,j} \\ \bar{\delta}_{i,j} \end{bmatrix} = \begin{bmatrix} R_i^\top (p_j - p_i) \\ \theta_j - \theta_i \end{bmatrix} + \begin{bmatrix} \epsilon^{\Delta}_{i,j} \\ \epsilon^{\delta}_{i,j} \end{bmatrix}. \tag{3.5}$$

It is convenient to adopt graph formalism to model the problem: each pose can be associated to a node of a directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ (often referred to as *pose graph*), where $\mathcal{V} = \{v_0, \ldots, v_n\}$ is the set of nodes and $\mathcal{E}$ (graph edges) is the set containing the unordered node pairs (i, j) such that a relative pose measurement exists between i and j. By convention, if an edge is directed from node i to node j, the corresponding relative measurement is expressed in the reference frame of node i. We denote with m the number of available measurements, i.e., $|\mathcal{E}| = m$.

It is well known in literature that the maximum likelihood estimate of network configuration X attains the minimum of the following cost function (see [20] and the references therein):

$$f(X) = \sum_{(i,j)\in\mathcal{E}} \begin{bmatrix} R_i^\top (p_j - p_i) - \bar{\Delta}^l_{i,j} \\ \theta_j - \theta_i - \bar{\delta}_{i,j} \end{bmatrix}^\top \Omega_{i,j} \begin{bmatrix} R_i^\top (p_j - p_i) - \bar{\Delta}^l_{i,j} \\ \theta_j - \theta_i - \bar{\delta}_{i,j} \end{bmatrix}$$

where $\Omega_{i,j} = P_{i,j}^{-1}$ is the *information matrix* of measurement (i, j). Therefore, the pose-based SLAM problem reduces to find a global minimum of the weighted sum of the residual errors squared, i.e., $X^* = \arg\min f(X)$.

3.2.2 A Linear Approximation

In this section we present the first contribution to the problem of graph-based SLAM, which has been proposed in [21]: a linear approximation for problem (3.6) that relaxes the assumption of previous work [20]. In [20] it was assumed that $P_{i,j}$ (and then $\Omega_{i,j}$) has the following structure:

$$P_{i,j} = \begin{bmatrix} P_{i,j}^{\Delta} & \mathbf{0}_{2\times 1} \\ \mathbf{0}_{2\times 1}^{\top} & P_{i,j}^{\delta} \end{bmatrix}. \tag{3.6}$$

Roughly speaking, this essentially requires that the relative position and relative orientation measurements (that together give the relative pose measurement) are uncorrelated. In order to present the subsequent derivation we need to rewrite the cost function (3.6) in a more compact form. For this purpose we define the unknown *nodes' position* $p = [p_1^{\top} \ \dots \ p_n^{\top}]^{\top}$ and the unknown *nodes' orientation* $\theta = [\theta_1 \ \dots \ \theta_n]^{\top}$; therefore the to-be-computed network configuration may be written as $x = [p^{\top} \ \ \theta^{\top}]^{\top}$ (note that we have excluded from x the pose of the first node that was assumed to be known). Then, we number the available measurements from 1 to m and we stack the relative position measurements in the vector $\bar{\Delta}^l = [(\bar{\Delta}_1^l)^{\top} \ (\bar{\Delta}_2^l)^{\top} \ \dots \ (\bar{\Delta}_m^l)^{\top}]^{\top}$, and the relative orientation measurements in the vector $\bar{\delta} = [\bar{\delta}_1 \ \ \bar{\delta}_2 \ \ \dots \ \ \bar{\delta}_m]^{\top}$. Accordingly, we reorganize the measurement information matrices $\Omega_{i,j}, (i,j) \in \mathcal{E}$, into a large matrix

$$\Omega \doteq \begin{bmatrix} \Omega_{\Delta} & \Omega_{\Delta\delta} \\ \Omega_{\delta\Delta} & \Omega_{\delta} \end{bmatrix}, \tag{3.7}$$

such that Ω is the information matrix of the vector of measurements $[(\bar{\Delta}^l)^{\top} \ \ \bar{\delta}^{\top}]^{\top}$.

Then, according to [20], the cost (3.6) can be written as:

$$f(x) = \begin{bmatrix} A_2^{\top} p - R\bar{\Delta}^l \\ A^{\top}\theta - \bar{\delta} \end{bmatrix}^{\top} \begin{bmatrix} R\Omega_{\Delta}R^{\top} & R\Omega_{\Delta\delta} \\ \Omega_{\delta\Delta}R^{\top} & \Omega_{\delta} \end{bmatrix} \begin{bmatrix} A_2^{\top} p - R\bar{\Delta}^l \\ A^{\top}\theta - \bar{\delta} \end{bmatrix} \tag{3.8}$$

where:

- A is the *reduced incidence matrix*, obtained from the incidence matrix $\mathcal{A}$ (see "Preliminary and Notation" Section), by deleting the row corresponding to node v_0;
- $A_2 = A \otimes \mathbf{I}_2$ is an expanded version of A, see [17, 20];
- $R = R(\theta) \in \mathbb{R}^{2m,2m}$ is a block diagonal matrix, whose nonzero entries are in positions $(2k-1,2k-1), (2k-1,2k), (2k,2k-1), (2k,2k)$, $k = 1, \dots, m$, such that, if the k-th measurement correspond to the relative pose between i and j, then the k-th diagonal block of R is a planar rotation matrix of an angle θ_i.

The residual errors in the cost function (3.8) are described by the following vector, whose entries represent the mismatch between the relative poses of a given configuration x and the actual relative measurements.

$$r(x) \doteq \begin{bmatrix} A_2^\top p - R\bar{\Delta}^l \\ A^\top \theta - \bar{\delta} \end{bmatrix} \tag{3.9}$$

Before presenting the proposed approach we anticipate the main intuition behind the algorithm. The cost function is quite close to a quadratic cost function, since the last part of the residual errors in (3.9) is linear, and the overall cost function (3.8) becomes quadratic as soon as the rotation matrix R is known. Therefore, the basic idea is (i) to obtain an estimate of nodes orientations exploiting the linear part of the residual errors in (3.9), (ii) to use the estimated orientation to compute an estimate of R, and (iii) to solve the overall problem in the optimization variable x. This basic intuition is the same motivating [20], although here the derivation is made more complex by the presence of the correlation between position measurements $\bar{\Delta}^l$ and orientation measurements $\bar{\delta}$.

We are now ready to present the proposed linear approximation for pose graph optimization, whose properties will be analyzed in Theorem 1.

The effectiveness of the linear approximation computed using Algorithm (5) is assessed by the following result.

Theorem 1 *Given the inputs $\{\bar{\Delta}^l, \bar{\delta}, \Omega, A, A_2\}$, and assuming the the information matrix Ω to be positive-definite, the following statements hold for the quantities computed in Algorithm 5:*

1. *$\Omega_z, \Omega_y, \Omega_x$ are full rank;*
2. *The vector $\hat{\theta}$ in $\hat{z} \doteq [(\hat{\Delta}^l)^\top \ \hat{\theta}]^\top$, with desired probability α, is contained within an ellipsoid centered on the true nodes orientation θ and having* shape matrix *$P_\alpha = \chi^2_{n,\alpha} \left(A \left[\Omega_\delta + \Omega_{\delta\Delta} \Omega_\Delta^{-1} \Omega_{\Delta\delta} \right]^{-1} A^\top \right)^{-1}$, where $\chi^2_{n,\alpha}$ is the quantile of the χ^2 distribution with n degrees of freedom and upper tail probability equal to α;*
3. *The combination of the three phases is equivalent to applying a Gauss-Newton step to the cost function (3.6), starting from the initial guess $\hat{\theta}$.*

The first claim assures the uniqueness of the outcome of the proposed algorithm (no indetermination in the solution of the linear systems). The second claim states that, with arbitrary high probability α, we can bound the Mahalanobis distance between the estimate $\hat{\theta}$ and the true (unknown) nodes' orientation. Finally, the last claim assures that the proposed approximation improves over the initial guess $\hat{x}$, applying a Gauss-Newton step.

1 A linear approximation for the maximum likelihood pose-graph configuration can be computed in three phases, given the relative measurements $\bar{\Delta}^l$ and $\bar{\delta}$, the corresponding information matrix Ω, and the graph incidence matrices A and A_2:

1. Solve the following linear system in the unknown $z \doteq [(\Delta^l)^\top \ \theta]^\top$:

$$\Omega_z \ z = b_z \tag{3.10}$$

with:

$$Z = \begin{bmatrix} \mathbf{I}_{2m} & \mathbf{0}_{2m,n} \\ \mathbf{0}_{m,2m} & A^\top \end{bmatrix}, \quad \text{and,} \quad \begin{matrix} b_z = Z^\top \Omega \left[(\bar{\Delta}^l)^\top \ \bar{\delta}^\top\right]^\top \\ \Omega_z = Z^\top \Omega Z \end{matrix} \tag{3.11}$$

Call the solution of the linear system $\hat{z} \doteq [(\hat{\Delta}^l)^\top \ \hat{\theta}]^\top$.

2. Transform in the global frame the relative position measurements in $\hat{z}$, obtaining:

$$\hat{y} = T(\hat{z}) \doteq \begin{bmatrix} \hat{R} & \mathbf{0}_{2m\times n} \\ \mathbf{0}_{2m\times n}^\top & \mathbf{I}_n \end{bmatrix} \begin{bmatrix} \hat{\Delta}^l \\ \hat{\theta} \end{bmatrix} = \begin{bmatrix} \tau_1(z) \\ \tau_2(z) \end{bmatrix}_{z=\hat{z}} \tag{3.12}$$

with $\hat{R} = R(\hat{\theta})$; compute the corresponding information matrix (preserving correlation):

$$\Omega_y = (\hat{T}\Omega_z^{-1}\hat{T}^\top)^{-1} = (\hat{T}^{-1})^\top \Omega_z (\hat{T}^{-1}), \tag{3.13}$$

where $\hat{T}$ is the Jacobian of the transformation $T(\cdot)$:

$$\hat{T} \doteq \begin{bmatrix} \frac{\partial \tau_1}{\partial \Delta^l} & \frac{\partial \tau_1}{\partial \theta} \\ \frac{\partial \tau_2}{\partial \Delta^l} & \frac{\partial \tau_2}{\partial \theta} \end{bmatrix} = \begin{bmatrix} \hat{R} & J \\ \mathbf{0}_{n\times 2m} & \mathbf{I}_n \end{bmatrix}. \tag{3.14}$$

3. Solve the following linear system in the unknown $x = [p^\top \ \theta^\top]^\top$, given $\hat{y}$, see (3.12), and Ω_y, see (3.13):

$$\Omega_x \ x = b_x \tag{3.15}$$

with:

$$B = \begin{bmatrix} A_2^\top & \mathbf{0}_{2m\times n} \\ \mathbf{0}_{n\times 2n} & \mathbf{I}_n \end{bmatrix}, \quad \text{and,} \quad \begin{matrix} b_x = B^\top \Omega_y \hat{y} \\ \Omega_x = B^\top \Omega_y B \end{matrix} \tag{3.16}$$

The solution of the linear system (3.15) is the proposed linear approximation of the pose graph configuration: $x^* = [(p^*)^\top \ (\theta^*)^\top]^\top$. □

3.2.3 Experimental results

In this section we compare our methodology with several state-of-the-art optimization approaches, namely Gauss-Newton [58], TORO [45], g^{2}o [52], and the the linear approximation in [20]. The experimental tests are conducted on publicly available datasets: Freiburg Indoor Building 079 (FR079), MIT CSAIL Building (CSAIL), Intel Research Lab (INTEL), Manhattan World M3500 (M3500), Manhattan World M10000 (M10000). The number of nodes and loop closing constraints for each dataset is shown in Table 3.1.

Table 3.1. Number of nodes and edges for each dataset.

	Number of nodes $(n+1)$	Number of edges (m)
FR079	989	1217
CSAIL	1045	1172
INTEL	1228	1505
M3500	3500	5453
M10000	10000	64311

The relative pose measurements of the datasets {FR079, CSAIL, M3500, M10000} are available online [55], while the measurements of the dataset INTEL were obtained through a scan matching procedure, from the raw sensor data, available at [55]. The INTEL dataset is the same studied in the work [20]. The relations available at [55] only describe the relative pose measurements, while we are interested to test the behavior of the approaches for different measurement covariance matrices. In particular, for each dataset we consider three variants, each one corresponding to a different choice of the covariance matrix. The first variant (e.g., FR079, I) uses identity matrices as measurement covariances, i.e., the noise of the relative pose measurement between node i and node j is $\epsilon_i^j \sim \mathcal{N}(\mathbf{0}_3, \mathbf{I}_3)$. The second variant (e.g., FR079, P_s) uses a structured covariance matrix, as in eq. (3.6). The third variant (e.g., FR079, P_f) uses full covariance matrices obtained as follows.

Accuracy. We first show qualitative results for the proposed approach on each dataset. The estimated trajectory for each dataset is reported in Figure 3.2. For a quantitative evaluation of the accuracy of the approaches we report the optimal value of the cost function (3.6) attained by each of the compared techniques.

we use the SLAM benchmark metrics proposed in [53]. Such metrics provide a tool for comparing SLAM approaches that use different estimation techniques or different sensor modalities in terms of accuracy. For each dataset we consider three scenarios, in which different measurements covariance matrices are considered: in the first case the covariance matrix is chosen to be the identity matrix ($P_{ij} = I^3$); in

the second scenario we use a structured covariance matrix; in the third scenario the covariance matrix is full. Our implementation of the algorithm is written in C/C++ and uses the *CSparse* library [31]; the tests are conducted on a PC equipped with an Intel Core i7 3.4 GHZ and 8 GB of RAM.

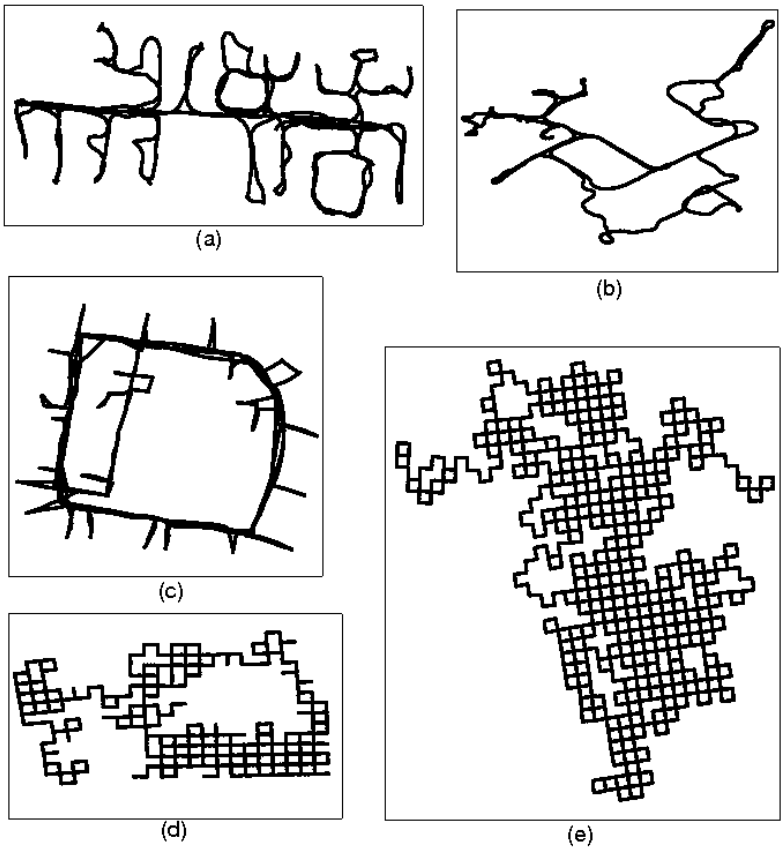


Figure 3.2. Estimated trajectory for each of the considered datasets: fr079 (a), csail (b), intel (c), M3500 (d), M10000 (e)

We now compare our approach, both in the case of structured covariance matrix and full covariance matrix, with the other optimization techniques. Table 3.2 shows the results in terms of the of the constraint satisfaction metrics while Table 3.3 reports the results in terms of computational time.

From the results, we can see that the Gauss-Newton approach is accurate enough to be used as a reference value but slow. g^2o is as accurate as Gauss-Newton on

Table 3.2. Cost function value attained by the compared approaches

		Linear Approximation (structured covariance)	Linear Approximation (unstructured covariance)	GAUSS NEWTON	TORO	g²o
FR079	$_I$	7.20E-02	7.20E-02	7.20E-02	8.60E-02	7.19E-02
	$_{Ps}$	3.94E+01	3.94E+01	3.88E+01	4.74E+02	3.89E+01
	$_{Pf}$	2.76E+02	2.90E+02	1.47E+02	8.99E+03	1.47E+02
CSAIL	$_I$	1.07E-01	1.07E-01	1.07E-01	1.18E-01	1.07E-01
	$_{Ps}$	4.06E+01	4.06E+01	4.06E+01	2.41E+03	4.06E+01
	$_{Pf}$	2.45E+02	2.33E+02	1.57E+02	4.57E+04	1.57E+02
INTEL	$_I$	8.07E-01	8.07E-01	7.89E-01	1.17	7.89E-01
	$_{Ps}$	1.45E+04	1.45E+04	2.15E+02	1.03E+05	2.15E+02
	$_{Pf}$	1.51E+06	1.07E+05	3.95E+02	2.53E+07	1.08E+03
M3500	$_I$	3.03	3.03	3.02	5.42	3.02
	$_{Ps}$	3.73E+03	3.73E+03	3.55E+03	2.18E+06	3.55E+03
	$_{Pf}$	1.15E+04	6.81E+03	2.09E+03	5.78E+08	2.09E+03
M10000	$_I$	3.03E+02	3.03E+02	3.03E+02	3.29E+02	3.03E+02
	$_{Ps}$	1.99E+05	1.99E+05	1.98E+05	7.65E+06	2.28E+05
	$_{Pf}$	9.00E+05	9.61E+05	6.79E+05	2.07E+08	1.90E+07

most scenarios, with only slight differences, while being much faster. TORO is both less accurate and slower compared to the other approaches. Our three-phase approaches are shown to be faster than all the other optimization techniques, while approaching the reference value provided by Gauss-Newton. Only in one scenario ($M3500_{Pf}$) the accuracy is lower than Gauss-Newton and g^2o. While the technique proposed in this work is relatively more accurate than the approach with structured covariance matrices, this comes at the cost of a slightly higher computation time. As only in the third scenario for each dataset we are using full covariance measurement matrices, we do not see any difference between our old approach and the new one on the first two scenarios in terms of accuracy. We should also remark that while our approach is based on a linear approximation, as opposed to the other approaches, the approximation is always more accurate than TORO in terms of the constraint satisfaction metric, and in one case more accurate than g^2o, which is the state of the art. g^2o is more accurate then our approach in term of cost-function evaluation in the other cases, but slower than linear optimization approaches.

Table 3.3. Average computation time (in seconds) for the compared approaches.

		Linear Approximation (structured covariance)	Linear Approximation (unstructured covariance)	GAUSS NEWTON	TORO	g²o
FR079	I	5.80E-03	8.15E-03	1.99E-01	3.19E-01	1.05E-02
	Ps	5.76E-03	7.87E-03	2.00E-01	3.39E-01	1.07E-02
	Pf	5.81E-03	8.16E-03	2.00E-01	3.04E-01	1.06E-02
CSAIL	I	5.72E-03	7.55E-03	2.65E-01	2.89E-01	1.01E-02
	Ps	5.53E-03	7.46E-03	2.01E-01	2.90E-01	1.01E-02
	Pf	5.59E-03	7.50E-03	2.66E-01	2.88E-01	1.03E-02
INTEL	I	7.10E-03	9.49E-03	5.87E-01	4.15E-01	1.32E-02
	Ps	7.01E-03	9.49E-03	4.90E-01	3.89E-01	1.31E-02
	Pf	6.98E-03	9.47E-03	5.91E-01	4.01E-01	1.31E-02
M3500	I	3.26E-02	4.04E-02	5.81	1.57	7.07E-02
	Ps	3.25E-02	4.03E-02	4.84	1.61	7.06E-02
	Pf	3.26E-02	4.05E-02	5.82	1.61	7.14E-02
M10000	I	3.55E-01	4.86E-01	2.21E+02	1.73E+01	6.93E-01
	Ps	3.57E-01	4.89E-01	2.21E+02	1.83E+01	6.96E-01
	Pf	3.55E-01	4.86E-01	4.11E+02	1.77E+01	6.91E-01

3.3 Robust front-end

Once we have a defined the problem of by means of graph optimization, we need a way to construct the graph itself, by estimating the nodes which represent the robot trajectory and the constraints relative to the edges. A solution to the problem of retrieving the constraints based on sensor data acquired by the robot is often referred to as the SLAM *front-end*.

The front-end computes the measurements in 3.5, providing an initial estimate of successive robot poses and constraints between them. The front-end is also responsible for recognizing previously observed places (loop closings) and generating constraints. The graph is then sent to the back-end for optimization. In Section 3.3.1 we describe the main algorithm, while in 3.3.2 we illustrate loop closing constraints creation.

3.3.1 Main algorithm

Each node of the graph represents a pose assumed by the robot at a certain time instant. To each node a corresponding laser scan is associated. A list containing the nodes is created, representing the robot trajectory.

Odometric edges can be built starting from the results of scan matching, from wheel odometry only, from a combination of wheel odometry and scan matching (in this case the translation estimation is given by wheel odometry, while the angular rotation estimation is given by scan matching), or a combination of scan matching and IMU (used for rotation estimation only as well) (Algorithm 5 lines 1-2). The best way to retieve odometric edges depends on the sensors and the nature of the environment (i.e., in a corridor-like environment the translation estimated by wheel odometry could be more accurate than the one estimated by scan matching).

For scan matching we use the *Canonical Scan Matcher* (CSM) algorithm described in [23]. The algorithm implements iterative closest point (ICP) procedure using a custom point-to-line distance metric, in order to find the relative roto-translation between two laser scans. ICP can benefit from an initial guess, if present.

The pseudo-code describing this module is shown in Algorithm 5. In our implementation the initial guess for scan matching can be omitted, or given by odometric information. Odometric information can be provided by wheel odometry, if present, or by an inertial measurement unit (IMU) if present.

A new graph node is created and added to the list every *nodeStep* steps (line 4) and only if the robot has moved a minimum linear or angular distance since the last node was added (line 7).

For each odometric constraint the covariance matrix is taken from the one estimated by CSM if scan matching is used, otherwise it is set to default values.

We then check for possible loop closings between the current node and all previous nodes (line 11). Loops detection is described in the next Section. If at least one loop is found (line 12), we save the previous graph (nodes and constraints) (line 13) and add the new loop constraints to the graph (line 14). We then optimize the graph using the optimizer described in Section 3.4 (line 15). Finally, we check the consistency of the optimized graph (see Section 3.3.2) and, in case the consistency is not validated by the checks, we remove the loops found from the graph and we restore the previous graph (lines 16-17).

3.3.2 Loop closing constraints

At each step the algorithm also checks for possible loop closings using the same scan matching algorithm between the current pose of the robot and all the candidates taken from the past poses. The pseudo-code describing this module is shown in Algorithm 6.

Two kinds of loop closing constraints are implemented. If the relative rotation between the two candidate poses is larger that a certain threshold the constraint is treated as *orientation-only constraint* (lines 11-12). This is due to the fact that, since the laser scans do not have a large overlap, we cannot trust the translation estimated from ICP. This is also clear from the estimated covariance matrix given

Data: $\boldsymbol{z}_k$, $\boldsymbol{z}_{k-1}$, $\tilde{\xi}_{Odom}, \tilde{\xi}_{IMU}, p_{k-1}$, Graph
Result: p_k, Graph

```
1  ξ̃_scan ← performICP(z_{k-1}, z_k, ξ̃_Odom);
2  ξ̃_corr ← sensorFusion(ξ̃_Odom, ξ̃_IMU, ξ̃_scan);
3  p_k ← p_{k-1} ⊕ ξ̃_corr;
4  if k mod nodeStep = 0 then
5      n* ← Graph.getLastNode();
6      ξ ← n*.p ⊖ p_k;
7      if ‖ξ‖ > ξ_th then
8          n ← initNewNode(p_k, z_k);
9          Graph.addNode(n);
10         Graph.addOdometricContraint(n*, n, ξ);
11         loops_list ← Graph.searchLoops(n);
12         if loops_list.size() > 0 then
13             Graph_old ← Graph;
14             Graph.addLoops(loops_list);
15             Graph.optimize();
16             if not Graph.checkEdges() or not Graph.checkNodes() then
17                 Graph ← Graph_old;
18             end
19         end
20     end
21 end
```

Algorithm 5: Graph creation.

by CSM. Another case is when the two poses have time-stamps which are very close together (lines 11-12). For example, in the case of a robot travelling along a corridor, the estimated translation may be unreliable. For orientation-only constraints we set the rotation part of the information matrix to zero. In all other cases, the constraint is treated as a *full-pose constraint* (i.e., the full covariance matrix is used).

Since using simple scan matching between two laser scans leads to several wrong matches, some rejection procedures must be implemented in order to discard bad matches.

Initial guess distance check

At first, a check is implemented before scan matching. If the euclidean distance between the current pose and the candidate pose is larger than half the maximum range of the laser scanner, the loop is discarded, since this means that the two poses

are probably far away from each other (line 3).

Visibility check

A visibility check ensures that no large obstacle is present between the current pose and the candidate one using ray tracing. This check establishes if a large fraction of the laser rays, which lie along the edge connecting the two poses, are crossed by some obstacle. In this case the loop is discarded (line 6). Experimental tests showed that the visibility check is useful only in some particular scenarios.

After this checks scan matching is performed and a set of candidate loop closings is obtained. We then implement other checks in order to further discard possible wrong matches.

ICP error check

Another check is done on the ICP error given by the scan matcher. If the ICP error is too large or too small, the loop constraint is discarded, since it is not considered reliable (line 14).

Cartesian and angular error check

If the difference between the predicted translation and the one returned by the scan matcher is over a certain threshold, the candidate is discarded. The same check is done between the predicted and the estimated rotation (line 17). The threshold increases based on how far the robot has been travelling; in this way we account for odometric drift.

3.4 Graph optimization

The pose graph, which is the output of the front-end, is then fed to a graph optimization algorithm to create a globally consistent trajectory. We optimize the pose graph using the LAGO linear pose-graph optimizer presented in the previous Section.

Anyway, any graph optimization algorithm can be used, as long as it can read a pose-graph written in the same format of TORO [45] or g^{2}o [52]. In case g^{2}o is used, the *Vertigo* library [83] for robust optimization using switchable constraints is supported. In this case any of the loop closing constraint can be discarded.

Consistency checks are also performed after the optimization phase (see Algorithm 5, line 16). If any of the tests fails, the loops added in the last step are removed and the previous trajectory is restored (Algorithm 5, line 17).

Data: n, Graph
Result: loop_list
foreach n_i **in** *Graph* **do**
  $\tilde{\xi}_i = n_i.p \ominus n.p$;
  if $\|\tilde{\xi}_i.\rho\| > \frac{\Delta L}{2}$ **then**
    continue;
  end
  if not `visibilityCheck`(n, n_i) **then**
    continue;
  end
  $\xi_i \leftarrow$ `performICP`$(n_i.\boldsymbol{z}, n.\boldsymbol{z}, \tilde{\xi}_i)$;
  $\xi.type \leftarrow full\_contraint$;
  if $|\xi_i.\theta| > \theta_{th}$ **or** $\xi_i.\Delta t < \Delta t_{th}$ **then**
    $\xi.type \leftarrow orientation\_only$;
  end
  if $\xi_i.ICP\_err \notin [\epsilon_{ICP}^{min}; \epsilon_{ICP}^{max}]$ **then**
    continue;
  end
  if $\|\xi_i.\rho - \tilde{\xi}_i.\rho\| > \tau_\rho$ **or** $\left|\xi_i.\theta - \tilde{\xi}_i.\theta\right| > \tau_\theta$ **then**
    continue;
  end
  loop_list.`addContraint`(n, n_i, ξ_i);
end

Algorithm 6: Loop closing constraints creation.

Trajectory consistency check

After optimization we check for anomalous displacements in the trajectory, which can result from errors in the optimization process. For each two successive poses we check for the presence of large displacements on the y axis (since we assume a nonholonomic robot).

Residual errors check

We also check the fitting errors after the optimization, by comparing the distance of the nodes before and after the optimization. If any error is above a certain threshold the test is failed. In particular, the residues are too large if the relation

$$\max_i \|\xi_i \ominus \xi_i^*\| < \xi_{res}, \tag{3.17}$$

does not hold true, where ξ_i and ξ_i^* are the edges before and after the optimization.

3.5 Map creation

In our approach we keep the mapping process separated from the actual SLAM process. Since graph creation and optimization phases are fast, due to the simple scan-matching process and the fast backed, computation time can be considered negligible even for medium and large-scale environments. The critical part thus becomes the creation of the actual map. By keeping the mapping process separated we ensure scalability.

Each time the poses from the trajectory are recalculated by the graph optimizer, we feed the new poses, along with the corresponding laser scans attached to a second process, which is in charge of reconstructing the actual grid-map by raytracing the localized laser scans.

The trajectory sent to the mapping node can be sub-sampled in order to lower data transfer between processes and computational time.

3.6 ROS Implementation

The software has been developed using the Robot Operating System (ROS) [6] in C++ under Linux. The Eigen3 library [2] was used for matrix operations and the CSM library was used for scan matching, as stated in Section 3.3.1.

The functional architecture of the approach is shown in Figure 3.3. The *graph_slam* node subscribes to laser scan, odometry and IMU data and it is in charge of generating graph nodes and edges. To each pose the corresponding laser scan is attached. The *csm* package is used for scan matching. We call the association of robot pose and corresponding laser scan *localized scan*. The graph is then sent to an external program (the optimizer), which returns the optimized poses to the *graph_slam* node. These poses, along with the corresponding laser scans (localized scans), are finally sent to the *grid_mapper* node, which is in charge of generating the map. The *occupancy_grid_utils* stack is used for raytracing the grid-map starting from the list of localized scans.

In our implementation the input graph containing the trajectory and the loop closing constraints is written to a text file, than the optimization algorithm is run externally as a program, giving the text file as input. Finally, the results of the external algorithm is written to a new text file, which is read by our package and used to update the previous robot trajectory with the optimized one. While serialization of the graph data to a file and the execution of an external program for optimization leads to a big drop in computational speed, this implementation makes it straightforward to use any custom optimizer with no changes in the source code.

The full source code has been made available online.

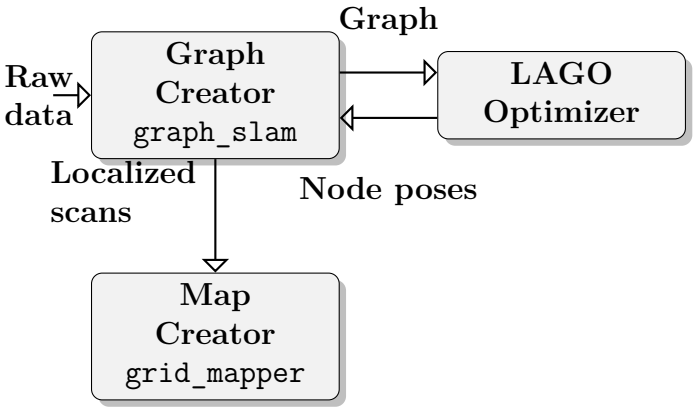


Figure 3.3. Architecture of the system

3.7 Experimental Analysis

In order to evaluate the proposed mapping algorithm, experimental tests have been carried out in simulation scenarios as well as real tests in medium and large-scale areas. Our system is composed by two robotic platforms: a Pioneer P3DX robot equipped with a laser range finder (SICK LMS-200), and a Corobot 4WD robot equipped with a Hokuyo URG-04LX-UG01 laser scanner and a XSens MTI IMU. In our experiments we set the different parameters of the algorithm to the values shown in Table 3.4, which have been found experimentally.

Performances on standard datasets are also provided.

Table 3.4. Parameters used in our experiments

Parameter	**Value**
nodeStep	10
ξ_{th}	0.01 m - 0.01 rad
ΔL	10 m
θ_{th}	0.5 rad
Δt_{th}	1 sec
ϵ_{ICP}^{min}	0.1
ϵ_{ICP}^{max}	0.3
τ_ρ	2 m
τ_θ	0.05 rad
ξ_{res}	0.15 m - 0.03 rad

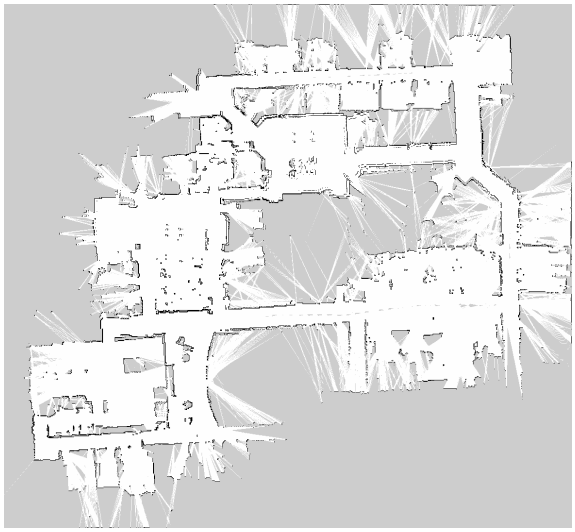


Figure 3.4. Resulting map for simulated environment.

3.7.1 Simulated environment

We first tested our algorithm in a large-scale simulated environment. We use the Stage simulator available in ROS, and the scenario represents the Willow Garage offices. Dimensions are 54x58.7m. The simulated robot was driven using manual tele-operation. Only simulated wheel odometry and laser range finder were used.

In Figure 3.4 we show the resulting map using our algorithm. Some inconsistencies (duplicate walls) can be seen in the resulting map. This is due to the fact that the accumulated odometric error was too large at the end of the run (after about 20 minutes), and thus the last loop closings were discarded.

3.7.2 Office-like environment

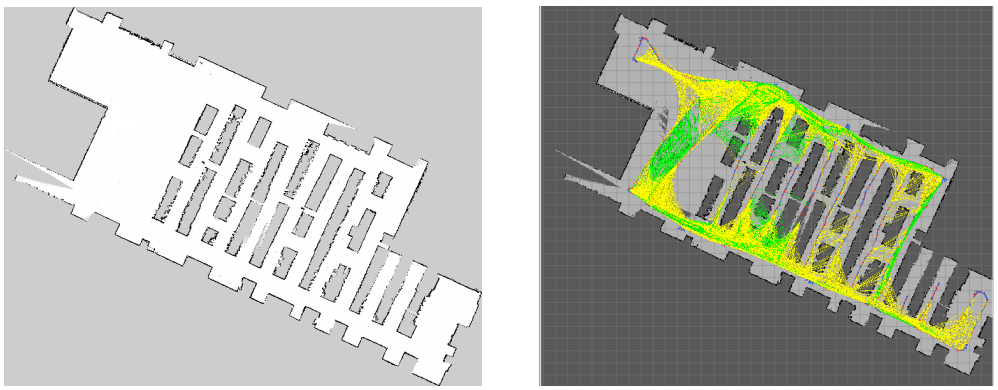
We show in Figure 3.5 the results of our approach in a real office-like environment. The map is of dimensions 37.5x17m and represents a corridor, a cluttered office and a hall. A Pioneer P3DX robot with wheel odometry and a laser range finder were used.

The resulting map has been succesfully used for localization and path-planning.

3.7.3 Data-center

We also show the results of our algorithm in two real large-scale indoor environments. The first scenario is a data-center room of dimensions 25x40m, composed by free corridors and rows of racks. In this experiment a Corobot rover was used. Wheel

Figure 3.5. Resulting map for office-like environment.



(a) (b)

Figure 3.6. Mapping results for the data-center room. (a) Resulting map using our approach; (b) Robot trajectory and loop closings. Red and blue arrows represent the trajectory; yellow lines represent orientation-only constraints; green lines represent full-pose constraints.

odometry was used for translation estimation, while an IMU was used for orientation estimation. The environment for this particular experiment was challenging for three reasons. Most of the server racks were covered by metal grills, thus making the scan-matching part prone to errors, as the laser rays were sometimes passing through the grills introducing big measurement errors. Moreover the symmetry of the environment posed a challenge for loop closing detection. Finally, the presence of slippery grills on the floor introduced large odometric errors. The result is shown in Figure 3.6.

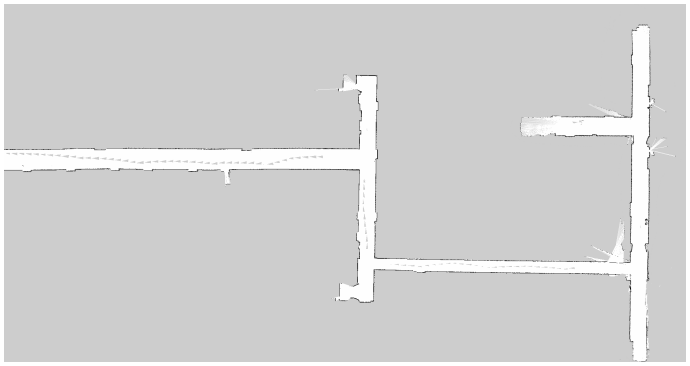


Figure 3.7. Mapping results for the data-center corridors.

The second scenario is composed by the corridors of the data-center leading to the various rooms. In this case the challenges are given by the lack of reference points along the corridors and by the lack of loops. The consistency of the map is based mostly on the effect of orientation-only constraints. Results are reported in Figure 3.7.

3.7.4 Benchmarking datasets

We finally report the values of the SLAM benchmark metrics proposed in [54]. Such metrics provide a tool for comparing the SLAM approaches in terms of accuracy. The metric uses only relative relations between poses and does not rely on a global reference frame.

In Table 3.5 we show the average values of the constraint satisfaction metrics for different standard datasets.

We also compared the performances in terms of memory footprint of our implementation with respect to the *gmapping* implementation available in ROS. Memory usage for *gmapping* remains stationary in a certain range, depending on map dimensions and number of particles. In our approach memory footprint of the *grid_mapper* node remains low, while memory usage for the *graph_slam* node increases based on the number of constraints, while never reaching the levels of *gmapping*.

Table 3.5. Benchmark metrics: translation error (η_c) and angular error (η_a).

Dataset	$\eta_c[m]$	$\eta_a[rad]$
INTEL	0.055	0.037
FR 79	0.065	0.032
MIT-CSAIL	0.070	0.053

Chapter 4

Human-robot interaction

4.1 Vision-based people tracking from a moving camera

4.1.1 Motivation

In the context of our work in collaboration with Thales Alenia Space we were interested in a scenario where an astronaut extravehicular team has to be accompanied by one or more semi-autonomous robots. In order to provide good awareness in human-robot interaction robust computer vision methods will be required, as for example people tracking and posture recognition. These methods should allow interaction in a wider range of environments and applications.

The starting point for such interaction is the capability for the robot to detect and track the position of the astronauts, in order to be able to follow them during operation, as well as to monitor their condition (e.g., the robots could be able to detect if an astronaut has fallen down). The following assumptions had to be taken into account during the design of the tracking system:

- The appearance of the targets (astronauts) is known, but variable in pose, dimensions, orientation, etc.
- The algorithm shall be able to correctly localize and follow an astronaut in a range usually between 1 m and 10 m
- The astronaut may only move forward, pointing his face towards the direction of motion
- The astronaut speed is supposed to be less than 1 m/s when the relative distance between the astronauts crew and the robot is 1 m

- The astronaut is not supposed to wear any visual marker, although, if they are already included in the mission equipment, they can support the tracking service

- The astronaut can be standing or can lie on the floor, but a given percentage of the astronaut body needs to be visible at any time (not occluded by obstacles)

4.1.2 Related work

The most recent years have shown a growing interest in human robot interaction for space robotics applications ([24], [37]). While many of these approaches rely on invasive techniques that require some modification of the environment (for example the need for the astronaut to wear particular markers), we are more interested in non-invasive approaches.

Vision-based people tracking approaches have several advantages as they are non intrusive, more natural and offer higher level information as compared to other sensors. However, there are some drawbacks that should be taken into account. Vision sensors may not be under consistent lighting or environmental conditions and items in the background or distinct features of the users may make recognition more difficult.

Different vision sensors have been used and different approaches have been proposed for each kind of sensor. Time-of-flight cameras can generate a depth map of what is being seen through the camera at a short range, and use this data to approximate a 3D representation of what is being seen; stereo camera systems can extract a 3D representation of the environment by comparing the output of the cameras; monocular cameras provide less information as opposed to the other sensors but allow a greater possibility of accessibility to a wider audience.

In particular, astronaut detection and tracking requires a robust people detector. Based on the kind of vision sensor that will be used different approaches have been explored. Different algorithms for people detection using monocular cameras exist, such as Histograms of Oriented Gradients [28], which is based on an analysis of gradients in the image and classification using linear support vector machines, or part detectors which try to detect different body parts of the person ([36], [25]). While gradient based detectors are easier to manage and can be easily optimized, multi-part detectors have the ability to detect objects even in presence of partial occlusions; on the other hand they require an accurate construction of models for the different body parts.

For stereo and time-of-flight cameras different approaches have been proposed ([67], [69], [10]). Sensor fusion has also been used to improve the robustness of people detection algorithms [80].

4.1.3 Monocular people detection and tracking

There are in general two different approaches in people detection: probability based detection and sliding window techniques. In the first, mostly part detectors are fused together and a model of how these parts can be configured to one another leads to a probability whether a human is present or not. This is especially useful for scenes where pedestrians are highly occluded. For the latter, a fixed-size detection window is moved over an image and at each position a classifier decides, whether an object is present or not. To detect different sizes, the image is step by step resized and the same detection method is used. In this work, the focus lies on sliding window techniques. A very good overview of different methods and detection results can be found in [91].

Pedestrian detection approaches can be decomposed into the generation of initial object hypotheses (also called ROI selection), verification (classification), and temporal integration (tracking). While the latter two require models of the pedestrian class, e.g., in terms of geometry, appearance, or dynamics, the initial generation of regions of interest is usually based on more general low-level features or prior scene knowledge.

ROI selection

The simplest technique to obtain initial object location hypotheses is the sliding window technique, where detector windows at various scales and locations are shifted over the image. The computational costs are often too high to allow for real-time processing. Significant speedups can be obtained by either coupling the sliding window approach with a classifier cascade of increasing complexity (rejection cascade) or by restricting the search space based on known camera geometry and prior information about the target object class. These include application-specific constraints such as the flat-world assumption, ground-plane-based objects and common geometry of pedestrians, e.g., object height or aspect ratio. In case of a moving camera in a real-world environment, varying pitch can be handled by relaxing the scene constraints or by estimating the 3D camera geometry online.

Classification

After a set of initial object hypotheses has been acquired, further verification (classification) involves pedestrian appearance models, using various spatial and temporal cues. These models can be divided into generative or discriminative. In both approaches a given image (or a subregion thereof) is to be assigned to either the pedestrian or non-pedestrian class, depending on the corresponding class posterior probabilities.

As for the classifier, two different approaches are common. Using Support Vector Machines (SVN), in the linear variant or with a kernel function, or AdaBoost. The latter is especially used to form rejection cascades as mentioned before. The advantage of support vector machines is, that the training is very easy and after training a linear support vector machine is easy to compute and thus faster.

Generative models try to create a full probabilistic model of all the variables and are usually divided into shape models and combined shape and texture models.

On the other hand, *discriminative models* approximate the Bayesian maximum-a-posteriori decision by learning the parameters of a discriminant function (decision boundary) between the pedestrian and non-pedestrian classes from training examples.

Another distinction is between single-part algorithms and part-based models. *Single-part* algorithms commonly use a sliding window technique. Sliding window techniques scan the image at all relevant positions and scales to detect a person. Consequently there are two major components: the feature component encodes the visual appearance of the person, whereas the classifier determines for each sliding window independently whether it contains the person or not. As typically many positions and scales are scanned these techniques are inherently computationally expensive. Fortunately, due to recent advances in GPUs, real-time people detection is possible (see [19],[75]).

Part-based models are composed by two major components. The first uses low-level features or classifiers to model individual parts or limbs of a person. The second component models the topology of the human body to enable the accumulation of part evidence. Part-based people models can outperform sliding-window based methods (such as HOG) in the presence of partial occlusion and significant articulations [80]. It should be noted however, that part-based models tend to require a higher resolution of the person in the image than most sliding-window based approaches.

Tracking

One line of research has formulated tracking as frame-by-frame association of detections based on geometry and dynamics without particular pedestrian appearance models. Other approaches utilize pedestrian appearance models coupled with geometry and dynamics.

Some approaches integrate detection and tracking in a Bayesian framework, combining appearance models with an observation density, dynamics, and probabilistic inference of the posterior state density. In the case of a single object, Bayesian filters are commonly used, such as Kalman filter and particle filters.

4.1.4 Stereo-based people detection and tracking

Stereo-based approaches have several advantages over monocular ones. On one hand, the information regarding disparities is more invariable to illumination changes than that provided by a single camera. Moreover, the ability to measure distances from the camera could be of great assistance for both detection and tracking ([30], [67], [69]).

4.2 Adaptive people and object tracking

We present in this Section a system for visual object tracking in 2D images for mobile robotic systems. The proposed algorithm is able to track people as well as objects and to adapt to substantial changes in object appearence during tracking. The approach has been presented in [78].

The algorithm is composed by two main parts: the detection part is based on *Histogram of Oriented Gradient* (HOG) descriptors [29] and *support vector machines* (SVM), while the tracking part is carried out by an adaptive particle filter. We now describe the main idea behind HOG descriptors and the people detection approach proposed in [29], which is at the basis of our approach.

4.2.1 Histograms of oriented gradients

The people detection approach proposed by Dalal and Triggs and based on *Histogram of oriented gradients* (HOG) features is widespread in literature and the detector shows good results at detecting pedestrians. According to a [91], no other approach using just one kind of feature outperforms HOG in terms of detection performances, and only approaches that use a mix of techniques show better performances, at the cost of much higher computational costs.

The idea at the basis of *historgrams of oriented gradients* (HOG) is that local object appearance within an image can be described by the distribution of intensity gradients or edge directions.

The detection procedure can be described as follows: first, the gradient of an image is calculated and then the image is divided into cells. For each cell, a discrete histogram of the orientation of the gradient is calculated. Now a detection window is moved over the image and at each location the histograms of each cell contained in the search window is converted into a feature vector. This feature vector is then fed into a support vector machine (which in the original work is trained on the INRIA people dataset) for classification. This is repeated for different scales of the image. The procedure is shown in Figure 4.1.

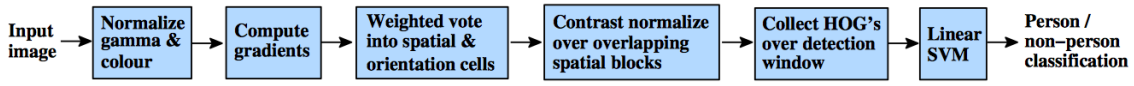


Figure 4.1. An overview of the feature extraction and object detection chain. The detector window is tiled with a grid of overlapping blocks in which Histogram of Oriented Gradient feature vectors are extracted. The combined vectors are fed to a linear SVM for object/non-object classification. The detection window is scanned across the image at all positions and scales, and conventional non-maximum suppression is run on the output pyramid to detect object instances.

4.2.2 The proposed approach

In our approach we apply the HOG detector described in the previous Section to a particle filter based tracking approach. The detector is coupled with the particle filter: detections are used in the update phase of the filter, as in usual tracking approaches, but in our method the tracked position predicted by the filter is used in order to acquire new samples for training of the classifier. The training is done online based on the acquired set of samples. Moreover the state of the particle filter is used to decide when a new training of the classifier is needed.

At each time only a certain region of interest is scanned for detections. This allows to considerably reduce detection time, which is the most expensive part in terms of processing time. The region of interest is updated at every time step based on previous detections.

Figure 4.2 illustrates the various parts of the algorithm in action on a single frame taken form a video sequence. The red rectangle is the output of the adaptive detector; the yellow rectangle represents the current region of interest; The blue dots represent the particles; the brighter ones represent particles with an higher weight. The yellow circle represents the position hypothesis with the highest weight, as explained in Section 4.2.5.

The HOG descriptor has a few advantages over other descriptors. It has been proved to be more invariant to changes in illumination and shadowing than other features. Moreover it upholds invariance to geometric and photometric transformations, except for object orientation. Such changes would only appear in larger spatial regions.

4.2.3 Detector

The detection part is based on HOG features and a linear-SVM classifier, which is trained online over a sliding window of samples taken from previous images on the basis of previous detections. In oder to reduce the probability of introducing a bias in the classifier, a subset of older samples is always preserved and introduced in the

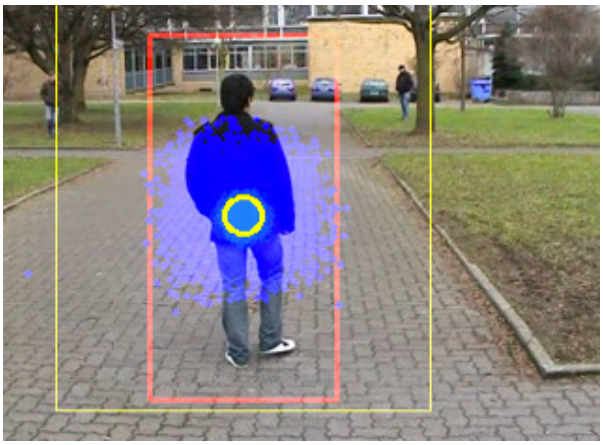


Figure 4.2. The algorithm in action.

training.

The classifier is re-trained only when the variance of the weights of the particles, given by the effective sample size N_{eff}, increases over a given threshold. We determined experimentally that we can take N_{eff} as a coarse measure of how the erformances of the detector are degrading. This is due to the fact that the appearance of the tracked object has changed significantly, and the classifier needs to be trained again on newer samples.

In order to improve the performances of the detector, at every time step only a small area around the last detection, a region of interest is fed into the detector. This can be done only under the assumption that the tracked object does not move very fast, but we found that the assumption holds for most kind of objects to be tracked. This simple expedient can provide frame rates up to 10-25 FPS depending on the size of the tracked object in the image.

In order to improve the tracking of objects that are moving away from the camera or the detection of objects in small resolution videos, we implemented a simple rule: if the last detected window is close to the minimum detectable height of the classifier, than the region of interest is magnified (digital zoom). There is one threshold parameter (default is 0) in the OpenCV implementation of the HOG algorithm that we are using, that allows to vary the accuracy of detection. For higher positive values of the parameter, less false positives, but more false negatives are produced. For negative values more false positives but also less false negatives are produced. Therefore, if no prior detection has occurred the threshold is set to an higher value(0.2 on our case) to avoid false detections and otherwise lower (-0.2 on our case) to avoid false negatives.

4.2.4 Online training

For the training of the SVM we use the approach described in [4]. The approach is composed of several steps: first, a HOG feature vector is calculated for every positive training image. At the same time 10 random windows for each negative training image are selected and HOG features are also calculated. We use the SVM training algorithm from the SVM-Light library [7] on this training set in order to obtain a detector. Since SVM-Light only outputs the support vectors v_i and their corresponding α_i, and the HOG algorithm from the OpenCV library needs a weight vector w, we simply calculate it with

$$w = \sum \alpha_i v_i$$

For bootstrapping our detector in the case of people detection we train our classifier on the INRIA Person Dataset [3]; in the case of object detection we use a suitable object class from the Pascal dataset [4].

The training is done over a sliding window of samples taken from previous smaples on the basis of previous detections. Every k_1 frames a rectangular area is taken from the current frame and put into a subset S_1 of samples. The position of the area corresponds to the current object position hypothesis with the highest weight from the particle filter (see Section 4.2.5 for details). The dimensions of the area are the same as the latest detection window, as we can assume that the tracked object will not have big changes is dimensions between subsequent frames. Every $k_2 > k_1$ frames the rectangular area is put into another subset S_2 of samples. This subset is used in order to include also old samples in the training, thus preventing the problem of overfitting in most cases.

When the dimensions of the subset S_1 is above a given threshold the classifier is re–trained on the current dataset, and the resulting weight vector w is fed to the classifier.

The state of the particle filter is also used to decide when a new training of the classifier is needed. When the effective sample size N_{eff} falls below a given threshold the classifier is re–trained on the current dataset, and the resulting weight vector is used instead of the older one.

4.2.5 Tracking

For the tracking of the detected object we use a particle filter implementation. Particle filters are sequential Monte Carlo methods based on point mass representations of probability densities. The first advantage of this method over classic Kalman filters is the ability to cope with non-linearity and non-gaussianity, which are critical in the case of a moving object. Another important aspect is the possibility to model multi-modal distributions.

In our approach the prediction phase is based on a simple linear motion motion model. The estimate of the new state of each particle is a linear extrapolation of the previous state plus Gaussian noise. We chose a simple motion model because for people and object tracking the advantage of using a stochastic model is not prominent compared to simpler motion models [73].

The update phase is based on object detections. Particles are weighted based on the distance from detection hypotheses provided by the adaptive HOG detector.

The resampling phase is based on *Kullback-Leibler divergence* (KLD). The number of particles representing the belief on the object pose at each step is adaptive, so that only the number of particles sufficient to represent the belief distribution is used.

We implement the adaptive particle filter approach proposed in [38] for robot localization. Since the approach is generic it can also be adopted for the particular case of object tracking. In this approach the key-point is to bound the error introduced by the sample-based representation of the particle filter. The underlying assumption is that the true posterior is given by a discrete, piecewise constant distribution such as a discrete density tree or a multi-dimensional histogram. Under this assumption we can determine the number of samples so that the distance between the Maximum Likelihood Estimate based on the samples and the true posterior does not exceed a pre-specified threshold ϵ. The distance between the Maximum Likelihood Estimate and the true distribution is measured by the Kullback-Leibler distance. As we saw in Section 2.3.2, the number of particles at each step i can be set to

$$n_i = \frac{1}{2\epsilon}\chi^2_{k-1,1-\delta}$$

where $\chi^2_{k-1,1-\delta}$ is a chi-square distribution with $1-k$ degrees of freedom. This value is the required number of particles to guarantee that with probability $1-\delta$ the Kullback-Leibler distance between the Maximum Likelihood Estimate of the position hypothesis and the true distribution is less than ϵ.

This is an advantage in terms of both memory occupation and computational resources. Moreover, we proposed to use the *effective sample size* N_{eff} as a measure of how well the current set of particles represents the true posterior of the object pose, by measuring the variance on the particles weight. If N_{eff} stays constant the new information does not help to identify unlikely hypotheses represented by the individual particles. In that case, the variance in the importance weights of the particles does not change over time. If, in contrast, the value of N_{eff} decreases over time, the new information can be used to identify that some particles are less likely than others.

The position hypotheses for the tracked object at each step are the result of a *Density-Tree* clustering procedure applied on the set of particles.

4.2.6 Experimental Tests

In this Section we present an evaluation of our algorithm on several video sequences and we compare the performances of our adaptive approach with other non-adaptive approaches.

In our implementation we used *OpenCV* libraries for the implementation of the detector. The algorithm was implemented in C++ and all the tests were conducted on a standard PC equipped with a 2.4 Ghz CPU and 2 Gb of RAM.

The algorithm has been tested on several video sequences from the *BoBoT* dataset (Bonn Benchmark on Tracking) [1], as well as on video sequences presenting strong changes in object appearence, illumination and occlusion. We show that the method is able to achieve a frame rate up to 20 fps on 320x240 video sequences on a standard PC.

In Experiment 1 we show a comparison of two different algorithms on some image sequences taken from the *BoBoT* dataset. This dataset contains both people and objects, and each video presents some challenging condition for detection and tracking. We also compare our algorithm with the results presented in [49] for three different sequences.

In Experiment 2 we show how the detection time and the number of particles changes over time.

Experiment 1

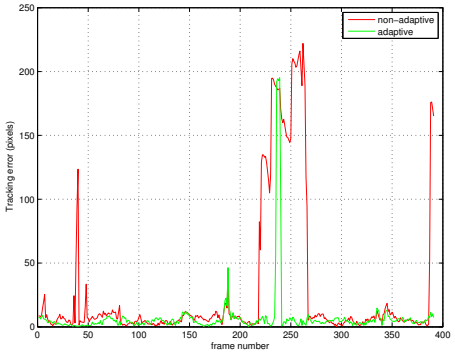
In Figure 4.2.6 we show a comparison of two different algorithms on some image sequences taken from the *BoBoT* dataset (*SeqA*, *SeqB*, *SeqI*). The first algorithm is a non-adaptive approach based on HOG features and particle filters. The second approach is the proposed approach with adaptive HOG detector and adaptive particle filters.

The ground-truth data which is available in this dataset is in the form of rectangular shapes surrounding the object to be tracked. Since in our approach the detection window is of fixed size, we do not use this ground-truth directly. We calculate instead for each frame the center of the detected object and we measure the error between this center and the output of our particle filter S_t.

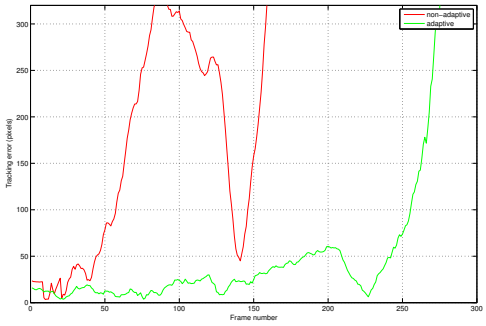
Sequence *SeqA* presents significative changes in appearence and fast movements of the tracked object; sequence *SeqB* presents strong changes in background appearence; sequence *SeqI* is an example of people tracking with many occlusions from crossing pedestrians.

We can see how the adaptive approach significantly outperforms the non-adaptive approach in every image sequence. In particular the adaptive method is able to recover more quickly from detection errors.

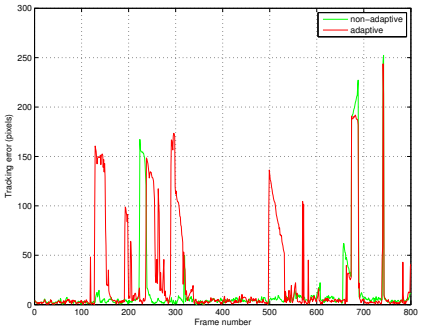
In Table 4.1 we present a comparison between 6 different algorithms. The first



(a) SeqA



(b) SeqB



(c) SeqI

Figure 4.3. Tracking error for three different sequences.

Table 4.1. Comparison of six methods

Seq.	Histogram	Multi-Comp.	non-adaptive H.-cs	adaptive H.-cs	adapt. p. part. H.-cs	Adaptive HOG
A	70.73	63.24	30.35	65.06	59.35	70.16
B	67.02	50.73	6.02	79.01	77.38	55.10
I	68.94	47.63	48.97	75.02	56.33	64.81

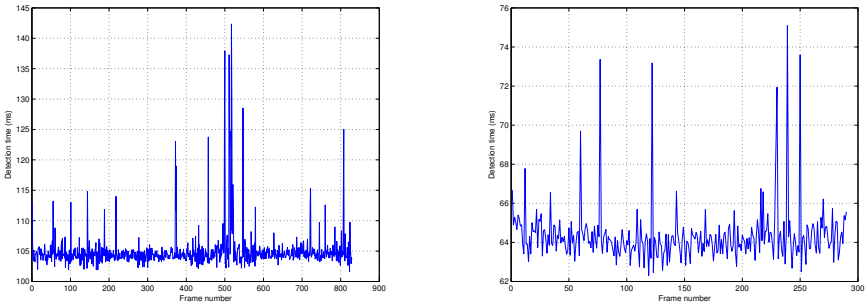
one is a simple tracking algorithm based on color histograms, the second one is based on multi-component tracking, the next three are different versions of the approach proposed in [49] and the last column contains the results form our adaptive algorithm. For every approach and for every sequence the mean score over the sequence is reported (calculated as suggested in [1]). Since our detection window is of fixed size, the results of our algorithm are generally better than the reported

score. We show that the results are comparable with the results presented in [49].

Experiment 2

In Figure 4.4(a) we show the detection time for each frame of sequence *SeqI* from the *BoBoT* dataset. The dimensions of the video sequence are 320 x 240 and the HOG detection window is 64 x 128. In Figure 4.6(b) we show the detection times for sequence *SeqB*. The dimensions of the video sequence are 320 x 240 and the HOG detection window is 64 x 64.

We show that by using a variable region of interest around the tracked object an high frame rate can be achieved. The peaks in detection time correspond to the time instants when the region of interests grows bigger or the full image is scanned.



(a) SeqI (b) SeqB

Figure 4.4. Detection time for two sequences.

Experiment 3

In figure 4.5 we show how the number of particles changes over time for a single run. The test video sequence is *SeqI* and the maximum number of particles was set to 5000. We also set a lower bound of 1000 to the number of particles, in order to avoid the convergence of the particle filter to a wrong position hypothesis in the case of bad detections. We can see how the number of particles drops and in some cases could be lower than the lower bound, without any significant downgrade in tracking performances.

Experiment 4

Successful tests of people tracking and following have been carried out by using a Microsoft Kinect camera mounted on a mobile robotic platform. The user is detected using the adaptive technique shown before. We also detect the distance of the user

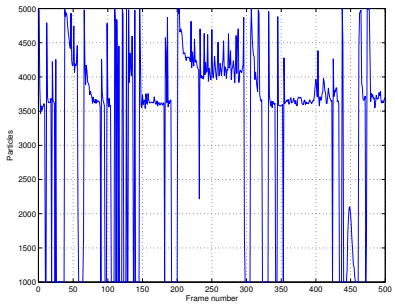
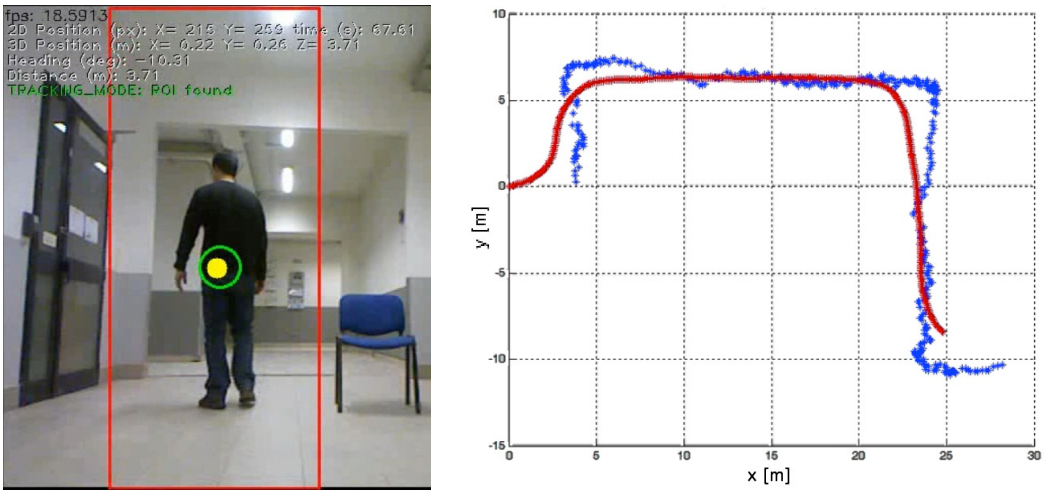


Figure 4.5. Number of particles over time.



(a) (b)

Figure 4.6. Example of robot following a user. (a) The algorithm in action. (b) Estimated trajectory of the user (blue dots) and trajectory followed by the robot (red line).

relative to the robot directly from the Kinect camera. An Extended Kalman Filter is implemented for smoothing the estimated position of the user on the ground plane. The state is composed by position and velocity on the (x, y) axes.

In this way, the robot is able to track the position of the user and follow it. Figure 4.6 reports the results of an experiment in which the robot is able to track and follow a user in real-time along a series of corridors.

Chapter 5

Conclusions

In this thesis we first presented solutions to different aspects of mobile robot navigation, in particular related to localization and Simultaneous Localization and Mapping, with a focus on real-world applications in industrial environments. Then we presented some methods related to Human-Machine Interaction; in particular we present an adaptive approach to people and object tracking for interaction between an autonomous robot and human operators, and finally we presented a novel method for human weight estimation based solely on vision sensors.

In the field of robot localization we proposed a multi-robot localization approach which is particularly suitable for large logistic environments, which present problems due to their large scale and in particular high symmetry. We showed how it is possible to exploit communication between the members of a team of robots in order to spread the knowledge about small asymmetries in the environment, as well as to provide some sort of recovery in the case of localization failures. Moreover, we introduce a novel way for the localization algorithm to aacquire insight about the localization state of each team member. Using our approach, the team is able to know with a certain confidence when all team members are correctly localized, so that they can start to execute high-level tasks.
We then introduced a series of practical improvements to the localization algorithm in the case of a single robot in the presence of high symmetry and noisy sensors. We combined a simple sensor fusion approach which is able to exploit gyroscopes to correct wheel odmetry and a marker detection algorithm which is able to correct the pose estimate of the robot using planar markers embedded in the environment. It should be noted that the planar markers can be in principle replaced by arbitrary planar objects already present in the environment.

In the field of Simultaneous Localization and Mapping we proposed a full SLAM approach based on pose graph optimization. We developed a graph optimization algorithm which is demonstrated to be faster than most state-of-the-art approaches, by exploiting a linear approximation of the problem that holds in the case of planar

environments. We then proposed a front-end for our approach which is based on laser scan matching only. We showed how the full approach is able to create consistent maps while running in real-time.

In the second part of the thesis we presented our contribution to the problem of people detection and tracking from a mobile camera. We presented an adaptive approach which is able to learn appearence changes over time and uses adaptive particle filters for tracking. We also showed how this approach can be used for tracking generic objects as well.

Finally, we presented some preliminary results on 3D body reconstruction for weight estimation using RGB-D cameras only. We presented some results of body reconstruction using two cameras and using a single camera, with the user moving in front of it.

5.1 Publications

5.1.1 Journal papers

- Bona B., Carlone L., Indri M., Rosa S., Supervision and monitoring of logistic spaces by a cooperative robotic team: methodologies, problems, and solutions, Intelligent Service Robotics, 2014, DOI: 10.1007/s11370-014-0151-0

- Abrate F., Bona B., Indri M., Rosa S., Tibaldi F., Multirobot Localization in Highly Symmetric Environments, Journal of Intelligent and Robotic Systems, 2013, DOI: 10.1007/s10846-012-9790-6

- Abrate F., Bona B., Indri M., Rosa S., Tibaldi F., Multi-robot map updating in dynamic environments, in Springer Tracts in Advanced Robotics, Volume 83, 2013, DOI: 10.1007/978-3-642-32723-0

5.1.2 Conference and workshop papers

- Russo L.O., Rosa S., Matteucci M., Bona B., A ROS Implementation of the Mono-SLAM Algorithm, International Conference on Artificial Intelligence & Applications (ARIA-2014), 2014

- Russo L.O., Airo Farulla G., Indaco M., Rosa S., Rolfo D., Bona B., Blurring prediction in Monocular SLAM, 8th IEEE International Design & Test Symposium 2013 (IDT), 2013

- Carlone L., Yin J., Rosa S., Yuan Z., Graph optimization with unstructured covariance: fast, accurate, linear approximation. In: Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR 2012), 2012.

- Rosa S., Paleari M., Ariano P., Bona B., Object Tracking with Adaptive HOG Detector and Adaptive Rao-Blackwellised Particle Filter. In: SPIE 8301, Intelligent Robots and Computer Vision XXIX: Algorithms and Techniques, 2012.
- Margaria V., Rosa S., Ariano P., HExEC: hand exoskeleton electromyographic control, Workshop on Human-Friendly Robotics, 2011

5.1.3 Preprints

- Rosa S., Paleari M., Velardo C. , Dugelay J.L., Bona B., and Ariano P., Towards Mass Estimation Using 3D Vision Sensors in Microgravity Environments

Bibliography

[1] Bobot bonn benchmark on tracking. Website. http://www.iai.uni-bonn.de/~kleind/tracking/.

[2] Eigen library. Website. http://eigen.tuxfamily.org/.

[3] Inria person dataset. Website. http://pascal.inrialpes.fr/data/human/.

[4] The pascal object recognition database collection. Website. http://pascallin.ecs.soton.ac.uk/challenges/VOC/databases.html.

[5] Rmap. Website. http://www-robotics.usc.edu/~ahoward/pmap/rmap_8h.html.

[6] Ros (robot operating system). Website. http://www.ros.org.

[7] Svm-light support vector machine. Website. http://svmlight.joachims.org/.

[8] Artoolkit website, 2013. http://www.hitl.washington.edu/artoolkit.

[9] A roadmap for us robotics, from internet to robotics. Technical report, 2013.

[10] W. Abd-Almageed, M. Hussein, and M. Abdelkader. Real-time human detection and tracking from mobile vehicles. In *Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE*, pages 149 –154, 2007.

[11] F. Abrate, B. Bona, M. Indri, S. Rosa, and F. Tibaldi. Switching multirobot collaborative localization in symmetrical environments. In *IEEE International Conference on Intelligent RObots Systems (IROS 2008), 2nd Workshop on Planning, Perception and Navigation for Intelligent Vehicles (PPNIV)*, 2008.

[12] F. Abrate, B. Bona, M. Indri, S. Rosa, and F. Tibaldi. Three state multirobot collaborative localization in symmetrical environments. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, pages 1–6, 7th May, 2009.

[13] F. Abrate, B. Bona, M. Indri, S. Rosa, and F. Tibaldi. Map updating in dynamic environments. In *International Symposium on Robotics (ISR)*, 2010. Accepted for publication.

[14] Fabrizio Abrate, Basilio Bona, Marina Indri, Stefano Rosa, and Federico Tibaldi. Multi-robot map updating in dynamic environments. In Alcherio Martinoli, Francesco Mondada, Nikolaus Correll, GrÃ©gory Mermoud, Magnus Egerstedt, M. Ani Hsieh, Lynne E. Parker, and Kasper StÃ¸y, editors,

Distributed Autonomous Robotic Systems, volume 83 of *Springer Tracts in Advanced Robotics*, pages 147–160. Springer Berlin Heidelberg, 2013.

[15] Indri M. Rosa S. Tibaldi S. Abrate F., Bona B. Multirobot localization in highly symmetrical environments. *Journal of Intelligent & Robotic Systems*, 71:403–421, 2013.

[16] M. Atiquzzaman and M.W. Akhtar. Complete line segment description using the hough transform. *Image and Vision Computing*, 12:267–273, 1994.

[17] P. Barooah and J.P. Hespanha. Estimation on graphs from relative measurements. *IEEE Control Systems Magazine*, 27(4):57–74, 2007.

[18] Peter Biber and Tom Duckett. Dynamic maps for long-term operation of mobile service robots. In *In Proc. of Robotics: Science and Systems (RSS*, 2005.

[19] B. Bilgic, B.K.P. Horn, and I. Masaki. Fast human detection with cascaded ensembles on the gpu. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, 2010.

[20] L. Carlone, R. Aragues, J.A. Castellanos, and B. Bona. A linear approximation for graph-based simultaneous localization and mapping. In *Proc. of Robotics: Science and Systems*, 2011.

[21] Luca Carlone, Jingchun Yin, Stefano Rosa, and Zehui Yuan. Graph optimization with unstructured covariance: Fast, accurate, linear approximation. In Itsuki Noda, Noriaki Ando, Davide Brugali, and JamesJ. Kuffner, editors, *Simulation, Modeling, and Programming for Autonomous Robots*, volume 7628 of *Lecture Notes in Computer Science*, pages 261–274. Springer Berlin Heidelberg, 2012.

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

[23] Andrea Censi. An ICP variant using a point-to-line metric. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Pasadena, CA, May 2008.

[24] S. Chien, R. Doyle, A.G. Davies, A. Jonsson, and R. Lorenz. The future of ai in space. *Intelligent Systems, IEEE*, 21(4):64 –69, 2006.

[25] E. Corvee and F. Bremond. Body parts detection for people tracking using trees of histogram of oriented gradient descriptors. In *Advanced Video and Signal Based Surveillance (AVSS), 2010 Seventh IEEE International Conference on*, pages 469 –475, 2010.

[26] I. J. Cox. Blanche: Position estimation for an autonomous robot vehicle. *Autonomous Mobile Robots: Control, Planning, and Architecture*, 2:285–292, 1991.

[27] Ingemar J. Cox and Gordon T. Wilfong, editors. *Autonomous robot vehicles*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[28] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 2005.

[29] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 2005.

[30] T. Darrell, G. Gordon, M. Harville, and J. Woodfill. Integrated person tracking using stereo, color, and pattern detection. In *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*, pages 601 –608, 1998.

[31] Timothy A. Davis. *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006.

[32] F. Dellaert, J. Carlson, V. Ila, K. Ni, and C. Thorpe. Subgraph-preconditioned conjugate gradients for large scale SLAM. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, 2010.

[33] F. Dellaert and M. Kaess. Square root SAM: Simultaneous localization and mapping via square root information smoothing. *Int. J. Robot. Res.*, 25(12):1181–1203, 2006.

[34] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localisation and mapping (slam): Part i the essential algorithms. *IEEE ROBOTICS AND AUTOMATION MAGAZINE*, 2:2006, 2006.

[35] R.M. Eustice, H. Singh, and J.J. Leonard. Exactly sparse delayed-state filters for view-based SLAM. *Int. J. Robot. Res.*, 22(6):1100–1114, 2006.

[36] P.F. Felzenszwalb, R.B. Girshick, and D. McAllester. Cascade object detection with deformable part models. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2241 –2248, 2010.

[37] Terrence W. Fong and Illah Nourbakhsh. Interaction challenges in human-robot space exploration. *Interactions*, 12(1):42–45, March 2005.

[38] D. Fox. Kld-sampling: Adaptive particle filters. In *In Advances in Neural Information Processing Systems 14*, pages 713–720. MIT Press, 2001.

[39] D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A probabilistic approach to collaborative multi-robot localization. *Autonomous Robots*, 8(3):325–344, 2000.

[40] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

[41] U. Frese, P. Larsson, and T. Duckett. A multilevel relaxation algorithm for simultaneous localization and mapping. *IEEE Trans. on Robotics*, 21(2):196–207, 2005.

[42] B. Gerkey, R.T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *11th Int. Conf. on Advanced Robotics (ICAR 2003)*, pages 317–323, 2003.

[43] D. Göring and H.-D. Burkhard. Multi robot object tracking and self localization using visual percept relations. In *IEEE/RSJ Int. Conf. on Intelligent Robots*

and Systems, pages 31–36, 2006.

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

[45] G. Grisetti, C. Stachniss, and W. Burgard. Non-linear constraint network optimization for efficient map learning. *IEEE Trans. on Intelligent Transportation Systems*, 10(3):428–439, 2009.

[46] G. Huang, A.I. Mourikis, and S.I. Roumeliotis. Observability-based rules for designing consistent EKF-SLAM estimators. *Int. J. Robot. Res.*, 29(5):502–528, 2010.

[47] Shoudong Huang and G. Dissanayake. Convergence and consistency analysis for extended kalman filter based slam. *Robotics, IEEE Transactions on*, 23(5):1036–1049, Oct 2007.

[48] N. Karam, F. Chausse, R. Aufrere, and R. Chapuis. Localization of a group of communicating vehicles by state exchange. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 519–524, 2006.

[49] D.A. Klein, D. Schulz, S. Frintrop, and A.B. Cremers. Adaptive real-time video-tracking for arbitrary objects. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 772 –777, oct. 2010.

[50] K. Konolige. Large-scale map-making. In *Proc. of the AAAI National Conf. on Artificial Intelligence*, 2004.

[51] K. Konolige, G. Grisetti, R. Kummerle, W. Burgard, B. Limketkai, and R. Vincent. Efficient sparse pose adjustment for 2d mapping. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 22–29, Oct 2010.

[52] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. G2o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3607 –3613, may 2011.

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

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

[55] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner. Slam benchmarking webpage. *http://ais.informatik.uni-freiburg.de/slamevaluation*, 2009.

[56] S.M. LaValle. Planning algorithms. *Cambridge University Press*, 2006.

[57] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1997.

[58] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1997.

[59] S. Hellbach H.-J. BÃ¶hme M. Himstedt, S. Keil. A robust graph-based framework for building precise maps from laser range scans. In *Proc. of the Workshop on Robust and Multimodal Inference in Factor Graphs. IEEE International Conference on Robotics and Automation (ICRA), 2013.*

[60] A. Martinelli. Improving the precision on multi robot localization by using a series of filters hierarchically distributed. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 1053–1058, 2007.

[61] R. Martinez-Cantin, N. de Freitas, and J. Castellanos. Analysis of particle methods for simultaneous robot localization and mapping and a new algorithm: Marginal-SLAM. In *Proc. of the IEEE lnternational Conf. on Robotics and Automation*, 2007.

[62] M. Di Marco, A. Garulli, A. Giannitrapani, and A. Vicino. Simultaneous localization and map building for a team of cooperating robots: A set membership approach. *IEEE Trans. on Robotics and Automation*, 19(2):238–249, 2003.

[63] Adam Milstein. Dynamic maps in monte carlo localization. In *Canadian Conference on AI*, pages 1–12, 2005.

[64] MobileRobots Inc. Mobilesim-the mobile robots simulator, 2011.

[65] Hans Moravec and A. E. Elfes. High resolution maps from wide angle sonar. In *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, pages 116 – 121, March 1985.

[66] A.I. Mourikis and S.I. Roumeliotis. Performance analysis of multirobot cooperative localization. *IEEE Trans. on Robotics*, 22(4):666–681, 2006.

[67] Rafael Muñoz-Salinas, Eugenio Aguirre, Miguel GarcÃa-Silvente, and Antonio Gonzalez. People detection and tracking through stereo vision for human-robot interaction. In *MICAI 2005: Advances in Artificial Intelligence*, volume 3789 of *Lecture Notes in Computer Science*, pages 337–346. Springer Berlin / Heidelberg, 2005.

[68] K. Murphy and S. Russell. Rao-blackwellized particle filtering for dynamic bayesian networks. *Sequential Monte Carlo Methods in Practice, Springer*, 2001.

[69] JosÃ© MÃ©ndez-Polanco, AngÃ©lica Muñoz-MelÃ©ndez, and Eduardo Morales. People detection by a mobile robot using stereo vision in dynamic indoor environments. In Arturo Aguirre, RaÃºl Borja, and Carlos GarciÃ¡, editors, *MICAI 2009: Advances in Artificial Intelligence*, volume 5845 of *Lecture Notes in Computer Science*, pages 349–359. Springer Berlin / Heidelberg, 2009.

[70] E. Olson, J.J. Leonard, and S. Teller. Fast iterative optimization of pose graphs with poor initial estimates. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 2262–2269, 2006.

[71] S. Panzieri, F. Pascucci, and R. Setola. Multirobot localization using interlaced extended kalman filter. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2816–2821, 2006.

[72] M. Peasgood, C. Clark, and J. McPhee. Localization of multiple robots with simple sensors. In *IEEE Int. Conf. on Mechatronics and Automation*, pages 671–676, 2005.

[73] Stefano Pellegrini, Andreas Ess, and Luc Gool. Predicting pedestrian trajectories. In Thomas B. Moeslund, Adrian Hilton, Volker KrÃ¼ger, and Leonid Sigal, editors, *Visual Analysis of Humans*, pages 473–491. Springer London, 2011.

[74] G. Pillonetto and S. Carpin. Multirobot localization with unknown variance parameters. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 1709–1714, 2007.

[75] Victor Adrian Prisacariu and Ian Reid. fasthog- a real-time gpu implementation of hog technical report no. 2310/09, 2009.

[76] Y. Rachlin, J.M. Dolan, and P. Khosla. Efficient mapping through exploitation of spatial dependencies. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3117–3122, Aug. 2005.

[77] I. Rekleitis, G. Dudek, and E. Milios. Probabilistic cooperative localization and mapping in practice. In *IEEE Int. Conf. on Robotics and Automation*, pages 1907–1912, 2003.

[78] Stefano Rosa, Marco Paleari, Paolo Ariano, and Basilio Bona. Object tracking with adaptive hog detector and adaptive rao-blackwellised particle filter, 2012.

[79] S.I. Roumeliotis and G.A. Bekey. Distributed multirobot localization. *IEEE Trans. on Robotics and Automation*, 18(5):781–795, 2002.

[80] Bernt Schiele, Mykhaylo Andriluka, Nikodem Majer, Stefan Roth, and Christian Wojek. Visual people detection: Different models, comparison and discussion. In *Proceedings of the IEEE ICRA 2009 Workshop on People Detection and Tracking*, pages 1–8, 2009.

[81] Amit Singhal. Issues in autonomous mobile robot navigation. Technical report, 1997.

[82] R. Smith and P. Cheesman. On the representation of spatial uncertainty. *Int. J. Robot. Res.*, 5(4):56–68, 1987.

[83] N. Sunderhauf and P. Protzel. Switchable constraints for robust pose graph slam. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1879–1884, Oct 2012.

[84] N. SÃ¼nderhauf and N. Protzel. Towards a robust back-end for pose graph slam. In *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[85] C.J. Taylor and J. Spletzer. A bounded uncertainty approach to cooperative localization using relative bearing constraints. In *IEEE/RSJ Int. Conf. on*

Intelligent Robots and Systems, pages 2500–2506, 2007.

[86] S. Thrun, W. Burgard, and D. Fox. Probabilistic robotics. *MIT press*, 2005.

[87] S. Thrun, D. Koller, Z. Ghahramani, H. Durrant-Whyte, and A.Y. Ng. Simultaneous mapping and localization with sparse extended information filters. In *Proc. of the 5th Int. Workshop on Algorithmic Foundations of Robotics*, 2002.

[88] S. Thrun and M. Montemerlo. The GraphSLAM algorithm with applications to large-scale mapping of urban structures. *Int. J. Robot. Res.*, 25:403–429, 2006.

[89] S. Thrun, W.Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

[90] N. A. Vlassis, G. Papakonstantinou, and P. Tsanakas. Dynamic sensory probabilistic maps for mobile robot localization. In *In Proc. IROS'98, IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 718–723, 1998.

[91] Christian Wojek and Bernt Schiele. A performance evaluation of single and multi-feature people detection. In Gerhard Rigoll, editor, *Pattern Recognition*, volume 5096 of *Lecture Notes in Computer Science*, pages 82–91. Springer Berlin - Heidelberg, 2008.

[92] R. Zlot, A. Stentz, M. Bernardine Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. pages 3016–3023, 2002.