Diogo Manuel da Silva Gonçalves

RobChair 2.0: Simultaneous Localization and Mapping and Hardware/Software Frameworks

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical Computer Engineering
September, 2013

UNIVERSIDADE DE COIMBRA

University of Coimbra
Faculty of sciences and technology
Departament of Electrical and Computer Engineering

RobChair 2.0: Simultaneous Localization and Mapping and Hardware/Software Frameworks

Diogo Manuel da Silva Gonçalves

Coimbra, 2013

RobChair 2.0: Simultaneous Localization and Mapping and Hardware/Software Frameworks

Advisor: Prof. Dr. Urbano José Carreira Nunes
Co-advisor: Dr. Ana Cristina Barata Pires Lopes

Jury:
Prof. Dr. Manuel Marques Crisóstomo
Prof. Dr. Rui Alexandre de Matos Araújo
Prof. Dr. Urbano José Carreira Nunes
Dr. Ana Cristina Barata Pires Lopes

Diogo Manuel da Silva Gonçalves

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical Computer Engineering

Department of Electrical and Computer Engineering

Faculty of Sciences and Technology, University of Coimbra

September, 2013

"One never notices what has been done; one can only see what remains to be done."

-Marie Curie.

Acknowledgements

This dissertation could not have been completed without the generosity and assistance of a large number of people to whom I would like to express my gratitude. First of all I am grateful to my advisors, Professor Urbano Nunes and Dr. Ana Lopes, or the possibility they gave me to work on a subject I like and for the support that they made available in a number of ways. I appreciate very much their suggestions and I am grateful for all the time they spent helping and making sure that I was fulfilling my objectives.

I am also very grateful to all my colleagues that helped me in some way to achieve this important milestone, in special to Luis Garrote, that always provided me with the best advices for keeping me in the right track and be motivated.

I thank ISR for providing the excellent conditions and resources that allowed me to accomplish my dissertation. This work has been supported by projet Grant "AMS-HMI12: Assisted Mobility Supported by shared control and advanced Human Machine Interfaces" (FCT Project RECI/EEI-AUT/0181/2012), co-funded by Fundação para a Ciência e Tecnologia and FEDER through "Programa Operacional Factores de Competitividade do QREN com referência COMPETE: FCOMP-01-0124-FEDER-027501".

I am very grateful to all my friends, and to my family for their continuous support, specially to my parents, who made a great effort to provide me with everything necessary so I could achieve my masters degree. And finally, last but not least, I would like to express my greatest gratitude to my girlfriend, Telma, who has been close to me from the very beginning. Her support and patience has been always present and stronger in the most difficult moments.

Resumo

A robótica tem sido alvo de constantes avanços nas últimas décadas, e está cada vez mais presente no nosso quotidiano. Esta ciência foca-se na percepção e manipulação do mundo físico através de dispositivos controlados por computador, dividindo-se em vários tipos de especialização, onde se destacam as aplicações orientadas à optimização de processos ou rotinas, ou simplesmente como uma solução para um determinado problema. Sistemas como plataformas móveis para exploração ou assistência, braços robóticos em linhas de montagem, veículos autónomos ou mesmo robôs educacionais, são exemplos de aplicações atuais bem sucedidas. Estes sistemas robóticos têm vários aspectos em comum: apercebem-se do ambiente circundante através de sensores e têm a capacidade de interagir com o mesmo. Assim, é essencial dotar o robô de certas capacidades, tais como a capacidade de se localizar, mapear e navegar. Para tal existem diversas técnicas disponíveis que são alvo de interesse por parte da comunidade científica, das quais se pode destacar o SLAM (Simultaneous Localization and Mapping), que confere ao robô a capacidade de se localizar e em simultâneo mapear o ambiente em que se insere. Uma das áreas que faz uso desta técnica é a robótica orientada à assistência do ser humano, e visa melhorar de forma substancial a independência e qualidade de vida de pessoas incapacitadas, interagindo com as mesmas e fornecendo-lhes a assistência necessária em tarefas específicas. Dispositivos tais como, cadeiras de rodas autónomas, robôs de acompanhamento ou braços manipuladores de assistência são exemplos de aplicações na área da robótica de assistência. Nesta dissertação é reestruturada uma plataforma móvel de assistência ao ser humano, baseada na *Robotic Wheelchair* (RobChair) do ISR-UC, na qual foram testadas e implementadas diversas técnicas que visam fornecer as funcionalidades e comportamento pretendidos nesta fase. Esta plataforma foi criada com o intuito de ser simples e versátil, no sentido de ser de fácil compreensão para utilizadores e investigadores futuros e de forma a ser uma plataforma de testes fiável para diversos algoritmos e técnicas desenvolvidas neste trabalho e em trabalhos futuros. Para tornar a nova RobChair num robô móvel e autónomo, fez-se uso do ROS (Robot Operating System) como plataforma de desenvolvimento e teste de diversos algoritmos. Várias técnicas de SLAM são estudadas e avaliadas para esta plataforma, através de métodos desenvolvidos para o efeito. Também é estudado o tema mais específico de "Loop Closure", melhorando assim o SLAM através de um método que faz uso de uma câmara para extração de características visuais.

Palavras-chave: Robótica, Robótica de assistência, Robô, SLAM, Mapeamento, Localização, Cadeira de Rodas, Robotic Wheelchair, RobChair, ISR-UC, ROS, Loop Closure.

Abstract

Robotics has gone through constant progress in the last decades, and is increasingly present in our quotidian. This science focus on the perception and manipulation of our physical environment through computer controlled devices and it is divided in several specialization types, such as applications that aim to improve processes or routines, or simply as a solution to a certain problem. Systems like mobile platforms for exploration or assistance purposes, robotic arms in assembly lines, autonomous vehicles, or even educational robots, are examples of current well succeeded applications of robotic systems. All these have some features in common: they can perceive their surrounding environment through sensors and are able to interact with it. Therefore, it is essential to provide certain skills to a robot, such as the ability of self-localization, environment mapping, and navigation. To achieve that, there are several available techniques that are the target of interest by the scientific community, from which SLAM (Simultaneous Localization and Mapping) can be highlighted. SLAM allows the robot to self-locate himself and simultaneously to acquire a map of the perceived environment in which it is inserted. One of the fields of robotics that use this technique is the human assistive robotics, and aims to substantially improve independence and life quality of impaired people, by interacting and providing the required assistance for specific tasks. Devices such as autonomous wheelchairs, companion robots, or assistive manipulators are examples of successful applications in the field of assistive robotics. In this dissertation, a human assistive mobile platform is restructured, based on a Robotic Wheelchair (RobChair) from ISR-UC, in which several techniques (such as SLAM) are tested and employed, in order to provide it with the required features and behavior. This platform is built to be simple and versatile, in a way that it is easy to use and understand by future users and researchers, and also to be a reliable test platform for several algorithms and techniques. To provide the RobChair with the required capabilities for this work, the ROS (Robot Operating System) frameworks is used as a development and test platform for several methods and algorithms that integrate with the RobChair. Several SLAM techniques are studied and evaluated for this platform. Finally, the more specific topic of "Loop Closure" is analyzed and it is implemented a method to improve SLAM, using a camera and extracting visual features.

Key words: Robotics, Assistive robotics, Robot, SLAM, Mapping, Localization, Navigation, Wheelchair, Robotic Wheelchair, RobChair, ISR-UC, ROS, Loop Closure, Visual Features.

Contents

List of Figures

List of Tables

Nomenclature

API	Application Programming Interface
ATE	Absolute Trajectory Error
BCI	Brain-Computer Interface
BoW	Bag of Words
CAN	Controller Area Network
CARMEN	Carnegie Mellon Robot Navigation
CML	Concurrent mapping and localization
EKF	Extended Kalman Filter
FAB-MAP	Fast Apearance-Based Mapping
GNU	GNU's not Unix
GPIO	General Purpose Input/Output
HDMI	High Definition Media Interface
HMI	Human-Machine Interaction
HOG-Man	Hierarchical Optimization for Pose Graphs on Manifolds
I2C	Inter-Integrated Circuit
ICP	Iterative Closest Points
ISR	Institute of Systems and Robotics
LIDAR	Light Detection and Ranging
MOOS	Mission Oriented Operating Suite
MRDS	Microsoft Robotics Developer Studio
OROCOS	Open Robot Control Software
PID	Proportional-Integral-Derivative
POSIX	Portable Operating System Interface
RANSAC	RANdom SAmple Consensus
RGB-D	Red, Green, Blue and Depth

RMSE	Root-Mean-Square Error
RobChair	Robotic Wheelchair (Referring to the ISR Robotic Wheelchair)
ROS	Robot Operating System
RPE	Relative Pose Error
RVIZ	Ruby Visualization Tool
SIFT	Scale-invariant feature transform
SLAM	Simultaneous Localization and Mapping
SPI	Serial Peripheral Interface
SURF	Speeded Up Robust Features
TCP	Transmission Control Protocol
ToW	Tree of Words
YARP	Yet Another Robot Platform

Chapter 1

Introduction

This chapter presents the introduction of this dissertation. Some insights concerning the motivation and context of the developed work will be presented, as well as the main goals and key contributions.

1.1 Motivation and context

Improving mobility of motor impaired people can be achieved through the use of assistive technologies, such as human-centered robots. Human-centered robots, and assistive robotics in particular, may contribute to help motor-impaired people to reach a better level of mobility, towards an improvement of their life standards. Furthermore, increasing the mobility levels of people with motor disabilities can also ultimately contribute to improve their social inclusion.

Robotic wheelchairs are one of the most common assistive robots used, so far, for mobility purposes. The RobChair project is being developed at ISR [Pires and Nunes, 2002, Lopes et al., 2011, Lopes et al., 2012, Lopes et al., 2013a] since the mid-90's, where the RobChair platform has been restructured several times in the past. The latest version was based in a low level distributed architecture and an abstraction layer that once again was becoming obsolete in terms of technological implementation. In order to obtain a reliable and up to date platform, an hardware and software restructuring was required. Hopefully, that will result in a new setup based on current technologies that allow for research and development of new assistive methodologies.

The localization problem can be described as the use of sensory information for robot self-localization in its environment, and it is one of the most fundamental problems to be solved in order to provide a mobile robot with autonomous capabilities. If a mobile robot does not know where it is, it will be difficult to decide what to do next. Most of deliberative tasks in mobile robotics are based in the assumption that the robot is able to answer to three fundamental questions, in particular: "Where am I?", "Where am I going?", and "How should I get there?". The first two questions are directly related with the localization problem.

Another important task is the capability of modeling the surrounding environment, by generating a map. By solving the localization problem alone, it is assumed that the robot was given a map in advance. This assumption is legitimate in a few real-world applications, as maps are often available a priori or can be constructed by hand. Being able to learn a map from scratch can greatly reduce the efforts involved in installing a mobile robot, and enable robots to adapt to changes without human supervision. In fact, mapping is one of the core competencies of truly autonomous robots [Thrun et al., 2005].

A SLAM technique aims to provide both localization and mapping that are required for the navigation method.

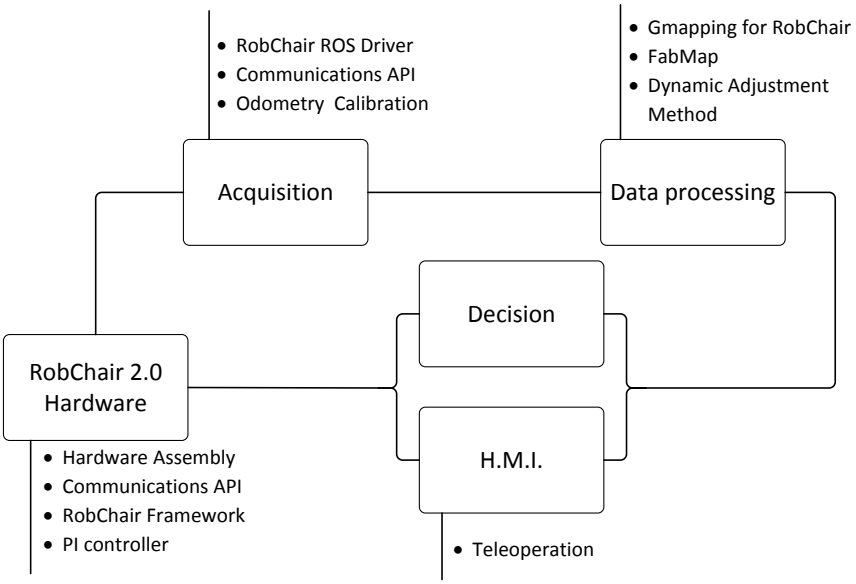


Figure 1.1: Key contributions

1.2 Goals

This work aims to upgrade the ISR Robotic WheelChair (RobChair) at the hardware, software and algorithmic levels.

To provide a background for autonomous navigation, a reliable SLAM technique needs to be implemented. In this sense, several techniques must be researched and compared in order to make sure the most appropriate solution is applied. To assure the implementation of a robust SLAM technique, further methods might be required, such as loop closure detection, which can be described as the task of deciding whether or not a robot has returned to a previously visited area. Such capability is crucial for enhancing the robustness and long life support of the SLAM algorithms and also to enable additional capabilities to mobile robots.

1.3 Implementations and key contributions

The following main implementations and contributions are described in this dissertation (see Fig. 1.1):

RobChair 2.0 Hardware (Chapter 3):

- Hardware assembly: The RobChair hardware restructuring was performed, where several obsolete devices were replaced by a new architecture based on a state of art motor controller and a less complex embedded computer system.
- Communications API: A new Application Programming Interface (API) was defined and implemented establish communication between low and high level devices.
- RobChair "Operating System": A UNIX software developed in C++ for the new microcomputer, which is able to manage the connections between RobChair devices. It also allows the communication with other external devices, such as a joystick, tablet, etc.

- PI controller design: PI controller design and implementation for RobChair motion controller was carried out.

Acquisition (Chapter 4):

- RobChair ROS Driver: A node developed in ROS (Robot Operating System) for the high level computer to exchange information among the RobChair devices.
- Odometry calibration: Rekleiteis's method[Rekleitis, 2003] was used to estimate and compensate the odometry errors of the RobChair 2.0.

Data processing (Chapter 6):

- Gmapping for RobChair: A ROS SLAM method (*Gmapping*) was adapted and implemented for RobChair 2.0 and it was changed to receive real time data from a dynamic adjustment method.
- FabMap: A method called openFABMAP (Open version of the Fast Appearance-based Mapping) was applied and explored for loop closure detection.
- Dynamic adjustment method: A ROS node was implemented to provide Gmapping with information related to loop closure detection. This information is provided by the openFABMAP algorithm.

Human-Machine-Interfaces (Chapter 4):

- Teleoperation: Two methods were deployed and implemented in ROS to directly steer the RobChair 2.0 with a keyboard or a game controller.

In Chapter 5, SLAM techniques in ROS are evaluated and a benchmarking method is proposed.

Chapter 2

State of the art

Robotics is defined as the branch of technology that deals with the design, construction, operation and application of robots [Oxford, 2013], or simply, the study of robots [NASA, 2011]. According to the Robot Institute of America, in 1979, Robot was described as a "*reprogrammable multifunctional manipulator designed to move material, parts, tools, or specialized devices through various programmed motions for the performance of a variety of tasks*" [R.I.A., 1979]. Although this committee-written definitions are still true, the field of robotics became wider and more diversified, due to enormous efforts and advancements in past decades from the robotic researcher community.

This work is in the field of assistive mobile robotics, with the main purpose of improving life quality of impaired people. This chapter describes the fundamentals required to understand the work presented in the remaining chapters of this thesis.

2.1 Mobile robotics

Mobile robotics is a branch of robotics related to movable robot systems, and it is one of the fastest growing fields in engineering [Meckstroth, 2009].

Types of mobile robots can be distinguished from its locomotion mechanisms that enables it to move through its workspace, making use of sensors and actuators. Because of all the existent environments and ways of locomotion, mobile robots have a large variety of possible ways to move, and so the selection of a robot's approach to locomotion is an important aspect of mobile robot design. Some are inspired in biological types of motion, others include human inventions, such as the wheel [Siegwart and Nourbakhsh, 2004].

Because of this diversity, mobile robots are constantly evolving and some can be very complex, creating a virtual unlimited number of robot configurations. Mobile robots can also be categorized according to its working purpose: search and rescue, surveillance, exploration, transportation, human assistance, and others.

From this wide range of mobile robots, this work focus on the field of assistive robotics, more specifically on a robotic wheelchair platform.

2.1.1 Robotic wheelchairs

Several intelligent wheelchair platforms were developed in recent years with the ultimate goal of improving mobility capabilities of disabled people, and also as development platforms for new technologies. Technically they consist in assistive navigation architectures based on semi-autonomous control systems for intelligent wheelchair platforms, such as the RobChair, developed at the Institute for Systems and Robotics at University of Coimbra [Pires and Nunes, 2002, Lopes et al., 2011, Lopes et al., 2012, Lopes et al., 2013a]. Most of the intelligent wheelchairs described in the literature incorporate a semi-

autonomous controller that belongs to one of three main groups of control approaches:

Direct user control: This one leaves control mostly to the user, and automatic navigation is only triggered when a given situation is detected (e.g. imminent collision).

Destination based: In this approach, the system works like an autonomous robot, for which the user simply provides a destination, and the robot is in charge of getting there.

Shared control: This subset, also referred as assisted semi-autonomous navigation, relies on a basic set of primitives (e.g. avoid obstacles, pass doorway, following wall) that can be used to assist the user in difficult maneuvers. The responsibility for selecting the most appropriate operating mode can be performed by the user (manual adaptation) or by the robotic wheelchair (automatic adaptation).

A list of some recent Robotic Wheelchairs are presented in Table 2.1, with a description of the main technologies applied, the implemented shared-control type and the human-machine interaction.

The RobChair, using a Brain-Computer Interface (BCI), has been an ISR research platform, where several projects related to navigation, obstacle detection and the BCI integration where successfully achieved in recent years, with the goal of improving life quality of impaired people [Lopes et al., 2012, Lopes et al., 2011, Lopes et al., 2013a, Pires and Nunes, 2002, Lopes et al., 2013b].

2.2 Robot middleware frameworks

Robot middleware frameworks, (often denominated as robot software frameworks), are essentially toolkits that provide a level of abstraction between a robot platform and the software algorithms, allowing the development of robot applications in a robust and modular way, while giving the freedom to implement, create/test software, and keep abstraction from the real platform.

There are several frameworks available today to the community, with different functionalities and tools. Some of them are listed in Table 2.2. Choosing the most appropriate one, can take into account the purpose of the robot application, as well as the personal preference and experience.

2.3 Simultaneous Localization and Mapping

For the last two decades, the robotics community has experienced a tremendous effort to find robust and general solutions for the Simultaneous Localization and Mapping (also know as SLAM) problem.

Many approaches to this problem have been made in the past by many [Montemerlo et al., 2002, Nieto et al., 2007, Grisetti et al., 2006, Maddern et al., 2012, Endres et al., 2012, Surmann et al., 2004, Thrun and Montemerlo, 2005], with generally good results for their specific robots, environments and requirements. Because it has been the focus of many researchers, several methods for performing SLAM are already published and known to the research community. Approaches to the SLAM solution depend on the type of sensors the robot is equipped with, type of the environmental constrains, task requirements and other restrictions.

Performing SLAM with a robot has not an obvious solution, because in order to localize the robot, a map is previously needed, and the localization of the robot is required to map the environment. This is often referred as the “Chicken-Egg problem” and it is also known as the problem of “concurrent

Table 2.1: Recent wheelchair platforms.

Institution	Main Robotic Technologies	Shared-Control Type and user intention	HMI
University of technology of Sydney [Patel et al., 2012]	Montecarlo localization Topological mapping.	Hierarchical Hidden Markov Model framework that predicts both the short term (local) and long term (navigational) goals of the user.	Joystick.
VAHM (LASC, University Paul Verlaine-Metz) [Grasse et al., 2010]	Particle filtering approach to implement the recognition of the most frequent paths according to an offline-constructed topological map.	Provides assistance to the user during navigation by proposing the direction to be taken when a path has been recognized.	Joystick.
SHARIOTO (Katholieke Universiteit Leuven) [Vanhooydonck et al., 2010]	Dynamic window approach for obstacle avoidance.	Shared-control with user intention prediction based on a Bayesian network.	Joystick.
INRIA [Rios-Martinez et al., 2011, Escobedo et al., 2012]	Motion planner based on the RiskRRT; Map of the environment is built using a LIDAR mounted on the top of the wheelchair; Important goals in the map are set by hand.	A Bayesian network is used to estimate the user intent. Generation of human friendly paths based on the application of a social filter, which includes constraints inspired by social conventions.	Face tracking and voice recognition.
LURCH Politecnico di Milano [Bonarini et al., 2012]	Localization based on odometry. Odometry correction is performed based in the detection of passive markers placed in the environment using vision. Trajectory planning based on the fast planner SPIKE (Spike plans In Known Environments).	Control module based on a fuzzy behavior management system, where a set of reactive behaviors, which will be carried out by the robot, are implemented as a set of fuzzy rules. Two set of rules were established: one implementing trajectory following, and the another one implementing obstacle avoidance.	Joystick, touch-screen, electro miographic interface, and Brain Computer Interface (BCI).
University of Michigan [Park et al., 2012]	A static occupancy grid map obtained via SLAM. Global topological map. Position and velocity estimation of new obstacles in the environment based on a Kalman filter.	Model Predictive Equilibrium Point Control (MPEPC) framework, which allows the navigation of a wheelchair in dynamic, uncertain, structured scenarios with multiple pedestrians.	Joystick.
RobChair Institute for Systems and Robotics, University of Coimbra [Lopes et al., 2011, Lopes et al., 2012, Lopes et al., 2013b]	A priori metric map; Markov localization based on laser scan matching; Global planner based on the A*algorithm and local planning for obstacle avoidance.	Two-layer collaborative controller that depends on the user's ability steering the wheelchair with the BCI (the user selects among a set of discrete steering commands); Intent matching algorithm that matches user intents with machine steering proposals.	Synchronous BCI; scanning interface with single/multiple switch.
University of Zaragoza [Iturrate et al., 2009]	A binary occupancy grid map is used to model the static obstacles and free space. A set of extended Kalman filters was chosen to track moving objects around the robot. The final motion of the vehicle was computed using the nearness diagram (ND) technique.	Control of real wheelchair and simulated wheelchair in virtual environment (selection of local surrounding points).	Synchronous BCI.

Table 2.2: Common Robot Frameworks of today

Framework	Description
CARMEN	The Carnegie Mellon Robot Navigation Toolkit, CARMEN, is an open-source collection of software for mobile robot control. CARMEN is modular software designed to provide basic navigation primitives including: base and sensor control, logging, obstacle avoidance, localization, path planning, and mapping [CARMEN-Team, 2000].
MRDS	Microsoft Robotics Developer Studio, provides a wide range of support to develop robot applications. The latest RDS 4 includes a programming model that helps make it easy to develop asynchronous, state-driven applications. RDS 4 provides a common programming framework that can be applied to support a wide variety of robots, enabling code and skill transfer [Microsoft, 2012].
MOOS	MOOS, originally from "Mission Oriented Operating Suite", is a C++ cross platform middle ware for robotics research. It is helpful to think about it as a set of layers [MOOS-Team, 2013].
Orca	Orca grew out of Orocos EU funded project at KTH, and it is now an open-source framework for developing component-based robotic systems. It provides the means for defining and developing the building-blocks which can be pieced together to form arbitrarily complex robotic systems, from single vehicles to distributed sensor networks [Orca-Team, 2009].
OROCOS	OROCOS, the Open Robot Control Software, is a project that aim to develop a general-purpose, free software, and modular framework for robot and machine control [OROCOS-Team, 2007].
Player	Player provides a network interface to a variety of robot and sensor hardware. Player's client/server model allows robot control programs to be written in any programming language and to run on any computer with a network connection to the robot. [Player-Team, 2010].
ROS	ROS, the Robot Operating System, is an open-source, meta-operating system for robots maintained by [WillowGarage-Team, 2013]. It provides the services expected from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers [Quigley et al., 2009, ROS-Team, 2013].
YARP	YARP, for Yet Another Robot Platform presents a set of libraries, protocols, and tools to keep modules and devices cleanly decoupled. [YARP-Team, 2013].

mapping and localization", or CML [Thrun et al., 2005]. Figure 2.1 shows where SLAM is inserted, (data processing module) between the fundamental modules of a mobile robot project. SLAM is integrated in the data processing module, where upon receiving the sensorial information from the data acquisition module, allows the implementation of decision methods in the decision module, such as navigation, making use of the current computed location and mapping to do so.

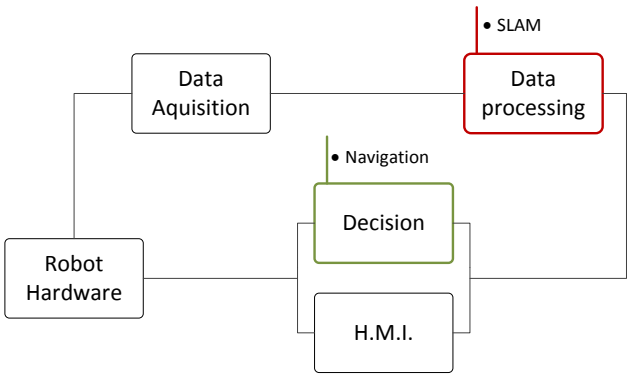


Figure 2.1: Common modules of a mobile robot, evidencing SLAM and Navigation

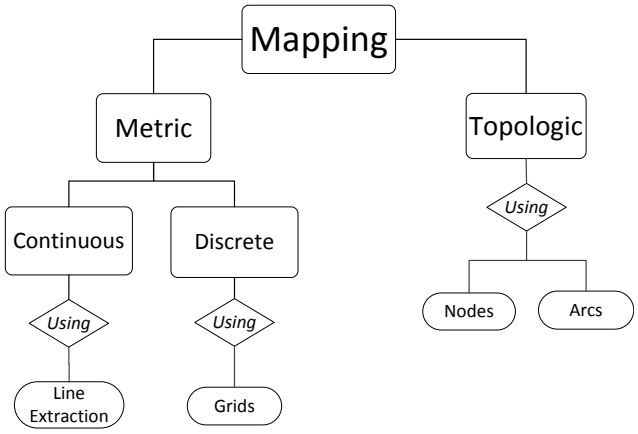


Figure 2.2: Most common mapping techniques and their representations

Therefore, to better understand the SLAM problem, it is necessary to split it into two distinct parts; localization and mapping:

Localization: Mobile robot localization is the problem of determining the pose of a robot in its surrounding environment, using the provided sensory information, and it is often called "pose estimation". It can also be seen as a problem of establishing correspondence between the map coordinate system and the robot's local coordinate system [Thrun et al., 2005]. In [Thrun et al., 2001], localization methods are classified in two major groups: position tracking, and global localization. Position tracking assumes that the initial robot pose is known and the robot only has to compensate small odometry errors occurring during robot navigation. Typically, position tracking methods are not able to recover when they lose track of the robot's pose. In global localization, the robot is placed somewhere in its environment and the initial pose of the robot is unknown. Global localization is more complex to solve than position tracking [Thrun et al., 2005].

Mapping: There are two major types of maps that can be generated and maintained by mobile robots: metric and topological maps [Thrun, 1998]. There are also other solutions, but these two are the most popular ones, as seen in Fig. 2.2.

Metric maps are the most simple and easy to understand by humans, as they represent the environment directly from sensorial information [Dudek and Jenkin, 2000]. These maps can be continuous or discrete. The continuous map is an interpolation of the real world from discrete measures. One example is line extraction, where best-fit lines are extracted from the provided sensorial data [Siegwart and Nourbakhsh, 2004]. Discrete maps, for example occupancy grid maps, are represented by a matrix of cells, where each cell has attached to it a occupancy value indicating the type of environment in that precise place, for example, a cell can represent empty space, an obstacle, unknown space, or just a value of the probability of that area being occupied. However, the most common mapping techniques just have a binary occupancy value, indicating occupied or free states for the cell. An occupancy grid map can divide the space into finitely many and small grid cells, making it possible to have a detailed map, but with a critical computational cost, so a trade-off between number of cells and cell size for each map must be performed [Thrun et al., 2005]. In Fig. 2.3 there is an example of a 2D metric map to the right,

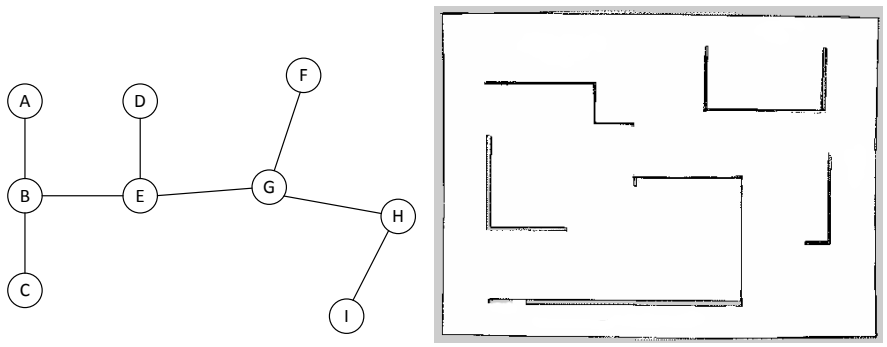


Figure 2.3: Examples of a topological map (left) and a discrete metric map (right)

also called occupancy grid map.

The other type, topological maps, describe the connectivity of different places, making use of graphs to represent the environments. Topological maps represent environments as a list of significant places or nodes, that are connected via arcs. Arcs are usually annotated with information on how to navigate from one place to another. The nodes in a topological map represent places, landmarks or other distinct situations. These maps are much more easy to store and use by a computer than metric maps, improving performance greatly in large environments, but difficult to keep consistent [Thrun, 1998]. In Fig. 2.3, on the left, there's an example of a small topological map of a corridor (B, E, G, H) and some rooms (A, C, D, F, I).

Solutions to perform SLAM rely heavily in probabilistic assumptions. Both the estimation of maps and robot localization are performed by gathering information from sensorial data, making use of probabilistic methods that have the capability of noise modeling and representing the uncertainty associated with sensorial data and estimation processes.

Most of the probabilistic models used in SLAM techniques rely on the Bayes rule, which determines the conditional probability of quantity x based on measurement data y:

$$p(x|\mathrm{y})=\frac{p(y|x)p(x)}{p(y)} \tag{2.1}$$

If x is a quantity that needs to be inferred from y, the probability $p(x)$ will be referred to as *prior probability distribution*, and y is called the data (e.g., a sensor measurement). The distribution $p(x)$ summarizes the knowledge regarding X prior to incorporating the data y (X here denotes a random variable and x denotes a specific value that X might assume). The probability $p(x|y)$ is called the *posterior probability distribution* over X.

Given that a mobile platform takes measurements over time, an extension of the Bayes rule, called the Bayes Filter is used. The Bayes filter algorithm has two essential steps: the control update (or prediction), and the measurement update. This is well detailed in Chapter I of [Thrun et al., 2005]. Most recursive state estimators are implementations of the Bayes filter, such as the Gaussian filter, Kalman Filter, Extended Kalman Filter, Histogram Filter, Particle Filter and others.

Generally, a SLAM method is composed by the same main modules, that together solve the SLAM problem, as shown in Fig. 2.4. The acquisition module prepares the sensor data for processing. The data processing module is where the main submodules of a SLAM algorithm are located. Three

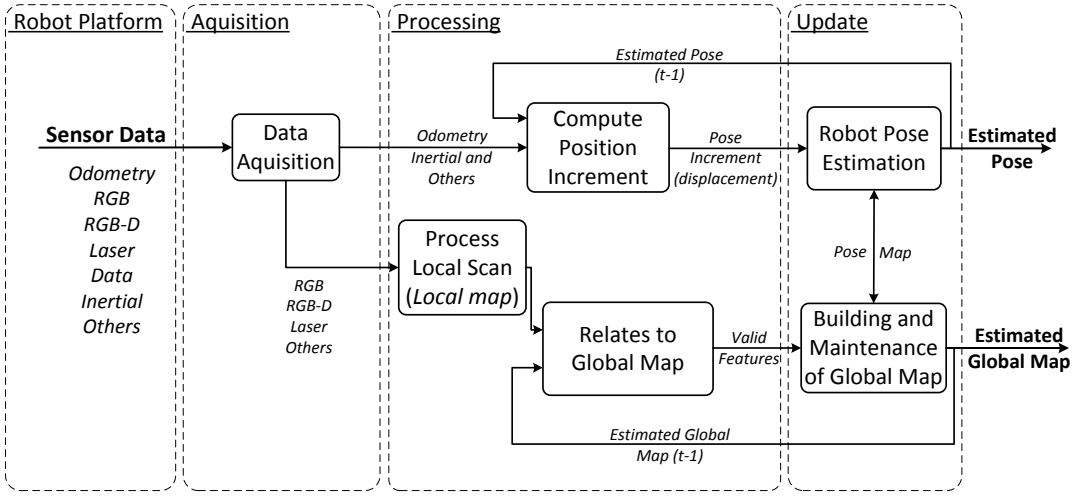


Figure 2.4: Generic SLAM algorithm

submodules can be observed in the processing module of Fig. 2.4. "Compute the position increment" submodule calculates the pose increment from the last known pose, using data such as odometry, to compute the displacement of the robot. The "Process Local Scan" submodule evaluates sensor data (such as laser range data) to perceive the robot surroundings at that instant, in the form of a local map. The third module uses the perceived local map and matches it with the global map, searching for a matching with a previous located space or, if not found, just prepares the local map to be further added to the global one. The update module is where the global pose of the robot is updated given the displacement provided by the processing module, and the global map is updated based on the local map.

2.3.1 EKF SLAM

EKF SLAM algorithm is based on the Extended Kalman Filter, or EKF, an algorithm similar to the Kalman Filter, where the linear predictions are replaced by their non-linear generalizations. This means that with the Extended Kalman Filter it is possible to solve non-linear problems by linearizing the non-linear function around the current estimate. This process is detailed in [Mughal, 2004].

This method approaches the SLAM problem using a geometric interpretation of the observation and motion model, shown in Fig. 2.5. A series of metric measurements $\mathbf{Z}_i$ are taken from locations $\mathbf{X}_i$ to features $\mathbf{m}_i$, typically in the form of range, bearing or a combination. The location of the features $\mathbf{m}_i$ with respect to the previously visited discrete locations $\mathbf{X}_i$ can then be determined in continuous geometric space. [Maddern et al., 2012]. The EKF is used to estimate the robot position through data from odometry and the observation of spacial references. A covariance matrix of landmark localizations and a pose vector are used to represent the computed robot location by the algorithm.

The EKF SLAM algorithm applies the EKF to online SLAM, using maximum likelihood data association. In doing so, EKF SLAM is subject to a number of approximations and limiting assumptions:

Feature-based maps: Maps, in the EKF, are composed of point landmarks. For computational reasons, the number of point landmarks is usually small (e.g., smaller than 1000). Further, the EKF approach performs better with distinctive landmarks. For this reason, EKF SLAM requires

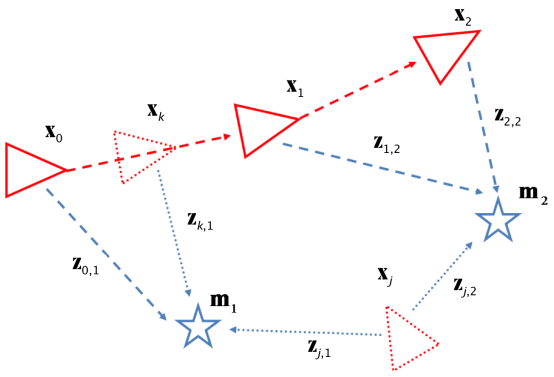


Figure 2.5: Geometric SLAM interpretation as shown in [Maddern et al., 2012]

significant engineering of feature detectors, sometimes using artificial beacons or landmarks as features.

Gaussian noise: EKF SLAM makes a Gaussian noise assumption for the robot motion and the perception. The amount of uncertainty must be relatively small, since otherwise the linearization in EKFs tends to introduce intolerable errors.

Positive measurements: The EKF SLAM algorithm, can only process positive sightings of landmarks. It cannot process negative information that arises from the absence of landmarks in a sensor measurements. This is a direct consequence of the Gaussian belief representation.

EKF SLAM has been applied with considerable success in a number of robotic mapping problems. The EKF-SLAM solution has some limitations, such as computational complexity, data association and non-linearities. Namely, the correction stage computational complexity grows quadratically with the number of landmarks, which is usually a problem in practical applications. Data association problem is very difficult when references are re-observed by very distinct observation points [Thrun et al., 2005].

2.3.2 FastSLAM

A popular SLAM algorithm that also makes use of the geometric solution to the SLAM problem, shown in Fig. 2.5 is FastSLAM, proposed in [Montemerlo et al., 2002], which uses a Rao-Blackwellized particle filter and various schemes for particle resampling. By storing many different location and map hypotheses as individual particles and assigning weights to those particles based on how well they match observations, particle filter SLAM avoids both the linearization and computational complexity issues of EKF SLAM.

In a particle filter, a given particle s is associated with an occupancy grid map $m^{[s]}$ and a topological map $\mathcal{G}^{[s]}$, both of which are updated while the robot is performing the exploration task. In the topological map $\mathcal{G}^{[s]}$, the vertices (or nodes, depending on interpretation) represent positions visited by the robot. The edges represent the trajectory corresponding to the particle s. To construct the topological map, an initial node is considered as the starting pose of the robot. Let $x_t^{[s]}$ be the pose of particle s at the current time step t. A new node is added at $x_t^{[s]}$ to the topological map $\mathcal{G}^{[s]}$, if the distance between $x_t^{[s]}$ and the all other nodes in $\mathcal{G}^{[s]}$ exceeds a threshold ($dist_{m^{[s]}}(x_t^{[s]}, n)$), or if none of the other nodes in $\mathcal{G}^{[s]}$ is visible from $x_t^{[s]}$ ($not\_visible_{m[s]}(x_t^{[s]}, n)$). This is described by:

$$\forall n \in nodes(\mathcal{G}^{[s]}) : [dist_{m^{[s]}}(x_t^{[s]}, n) > c \vee not\_visible_{m[s]}(x_t^{[s]}, n)] \tag{2.2}$$

Whenever a new node is created, an edge is also added from this node to the most recently visited node [Stachniss et al., 2005]. To each particle is assigned a weight w that is related to the importance or reliability of that particle. All weights are normalized to sum to unity. The particles are then resampled with replacement, where the probability of selection is proportional to the weight w. Remaining particles are then updated. This process allows the particle filter to store multiple hypotheses and switch between them as required, but it can suffer from "particle deprivation" if there are no particles near the correct hypothesis [van der Merwe et al., 2001, Maddern et al., 2012].

According to [Computer, 1999], the key idea of the Rao-Blackwellized particle filter for SLAM is to estimate the joint posterior $p(x_{1:t}, m|z_{1:t}, u_{1:t-1})$ about the map m and the trajectory $x_{1:t} = x_1, ..., x_t$ of the robot. This estimation is performed given the observations $z_{1:t} = z_1, ..., z_t$ and the odometry measurements $u_{1:t-1} = u_1, ..., u_{t-1}$ obtained by the mobile robot. The Rao-Blackwellized particle filter for SLAM makes use of the following factorization:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t-1}) = p(m|x_{1:t}, z_{1:t}) \cdot p(x_{1:t}|z_{1:t}, u_{1:t-1}) \tag{2.3}$$

this factorization allows to first estimate only the trajectory of the robot and then to compute the map given that trajectory. Since the map strongly depends on the pose estimate of t the robot, this approach offers an efficient computation [Grisetti et al., 2006].

This method has known advantages and disadvantages such as:

Advantages:

- Scales favorably to a large number of landmarks;
- Capable of multi-hypothesis data association (robustness);
- It has no linearization of non-linear motion models;
- Works well with diversified environments and landmarks.

Disadvantages:

- Computational heavy if particle number to high;
- Has an over optimistic robot pose estimate;
- Makes it harder to close the loop.

2.3.3 GraphSLAM

GraphSLAM is an algorithm that uses sparse information matrices produced by generating a graph of observation interdependencies, that is, two observations are related if they contain data about the same observed landmark.

Figure 2.6 helps illustrate the GraphSLAM algorithm. Shown there is the graph that GraphSLAM extracts from four poses labeled $x_1, ..., x_4$, and two map features m_1, m_2. Arcs in this graph come in

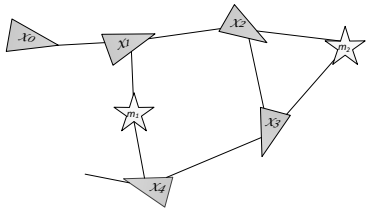


Figure 2.6: Example of robot poses, landmarks and connection links performed in GraphSLAM [Thrun and Montemerlo, 2005]

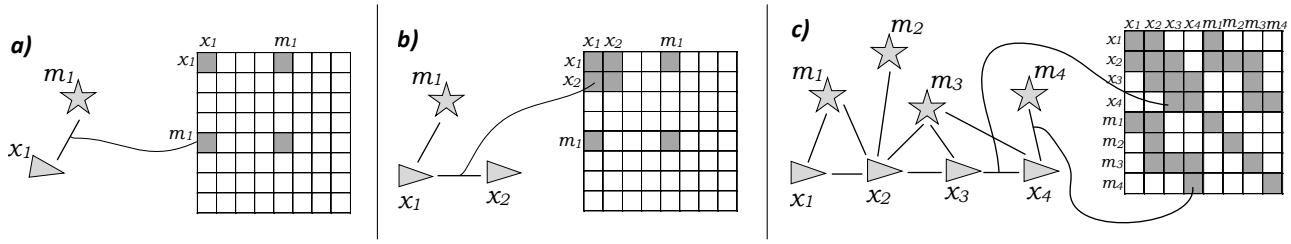


Figure 2.7: Acquisition of the information matrix in GraphSLAM [Thrun and Montemerlo, 2005]

two types: motion arcs and measurement arcs. Motion arcs link any two consecutive robot poses, and measurement arcs link poses to features that were measured there. Each edge in the graph corresponds to a nonlinear constraint. These constraints represent the negative log likelihood of the measurement and the motion models, hence are best thought of as "information constraints". Adding such a constraint to the graph is trivial for GraphSLAM; it involves no significant computation. The sum of all constraints results in a nonlinear "least squares problem". To compute a map posterior, GraphSLAM linearizes the set of constraints. The result of linearization is a sparse information matrix, seen in Fig. 2.7, and an information vector. The sparseness of this matrix enables GraphSLAM to apply the variable elimination algorithm, thereby transforming the graph into a much smaller one only defined over robot poses. The path posterior map is then calculated using standard inference techniques. GraphSLAM also computes a map and certain marginal posteriors over the map; the full map posterior is of course quadratic in the size of the map and hence is usually not recovered [Thrun and Montemerlo, 2005].

Figure 2.7 shows three steps of how the information matrix is acquired *(a)*, *b)* and *c)*). For each step, the left diagram shows the dependence graph, and to the right is the information matrix.

2.3.4 6D SLAM

6D SLAM, or six degree of freedom SLAM, is a technique that maps the environment and locates a robot in the 3D space. Maps are 3D models, acquired directly from 3D laser sensors, stereo cameras, RGB-D sensors, such as the Kinect, or even by using 2D images to infer a 3D space. The pose information is given in x, y and z coordinates, as well as *roll*, *pitch* and *yaw* rotations, $(x, y, z, \gamma, \beta, \alpha)$, and the method keeps track of these using visual odometry, that basically, at each image sample computes the robot dislocation.

A fast variant of the Iterative Closest Points (ICP) algorithm registers the 3D scans in a common coordinate system, also called the point cloud system. From here, as shown in previous SLAM methods, features, or landmarks, are extracted from the 3D scans and used to estimate the necessary information. 6D SLAM requires much more computer resources than the previously mentioned algorithms [Surmann et al., 2004].

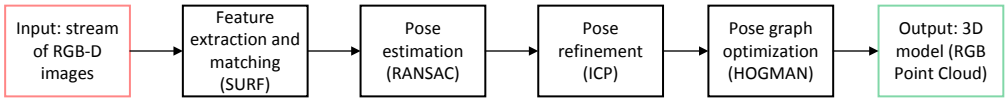


Figure 2.8: Processing steps of an RGBD SLAM approach from [Engelhard et al., 2011]

Another method, similar to the generic 6D SLAM, called RGB-D SLAM [Endres et al., 2012], has become popular in recent years, because it makes use of RGB image sensors with depth sensing capabilities, such as the Kinect sensor. In Fig. 2.8 is represented an approach using and RGB-D sensor. First, SURF features are extracted from the incoming color images. Then, these features are matched against features from the previous images. By evaluating the depth images at the locations of these feature points, a set of point-wise 3D correspondences between any two frames are obtained. Based on these correspondences, the relative transformation between the frames is estimated using RANSAC ("RANdom SAmple Consensus", is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers). The third step is to improve this initial estimate using the ICP algorithm. As the pair-wise pose estimates between frames are not necessarily globally consistent, the resulting pose graph in the fourth step is optimized using a pose graph solver. The output of the algorithm is a globally consistent 3D model of the perceived environment, represented as a point cloud [Engelhard et al., 2011].

2.4 Loop closure

Loop closure can be described as the task of deciding whether or not a robot has returned to a previously visited area. During the SLAM mapping process, the robot may come to a place that it has been before, and to solve the loop closing problem the robot must be able to recognize the previously visited places. Such knowledge is crucial for enhancing the robustness of both topological and metric SLAM algorithms, to increase the precision of the actual pose estimate, recognize previously mapped locations, or even for recovering from a kidnapping (when a robot is moved by something it does not control)[Filliat, 2009, Cummins and Newman, 2007]. According to [Newman and Ho, 2005], the hard part about loop closing is not asserting the presence of a loop but detecting when loop closure is even a possibility.

Correct loop closure detection can help the robot to reduce the uncertainty associated with the system states and amend the accumulated errors during the mapping process, while incorrect claim of loop closure can be catastrophic as it will either introduce redundancy or force an incorrect update to the map, which can finally lead to mapping errors, as seen in Fig. 2.9, the red path before performing loop closure, while the blue path has the loop closed and the map correctly joined. Loop closure detection is then a subject of increasing attention in SLAM research [Liu and Zhang, 2012].

Some relevant methods for performing loop closure are described in the next subsections.

2.4.1 Particle filter based loop closure

In the FastSLAM method, [Stachniss et al., 2005] loop closing means actively re-entering a known terrain for the robot and following a previously traversed path, ability that is given by the Particle Filter itself, as described before, because every particle is associated to an occupancy grid map and a topological map, both of which are continuously updated.

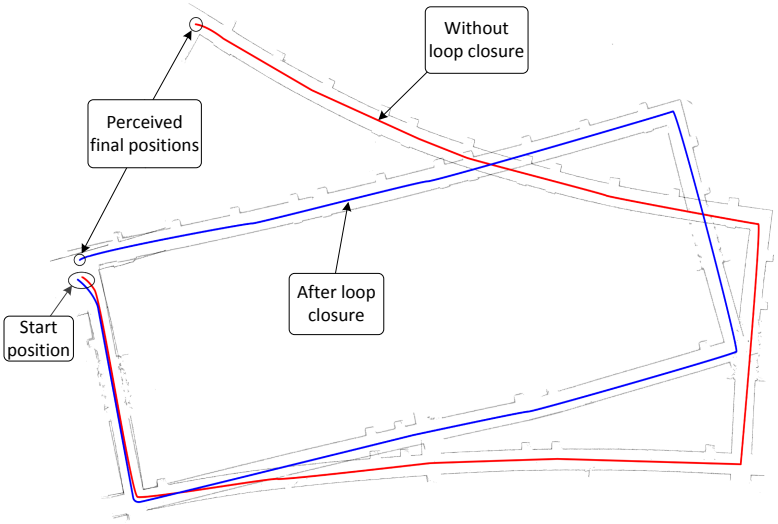


Figure 2.9: Example with successful loop closing (blue) and without (red) at ISR ground floor

In Fig. 2.10 it is shown in the two left images, that the robot traveled through unknown terrain, so that the uncertainty increased. In the right image, the robot re-entered known terrain so that samples representing unlikely trajectories vanished.

To determine whether or not a loop can be closed, for each sample s the set $\mathcal{I}(s)$ of positions of interest is computed which contains all nodes that are close to the current pose $x_t^{[s]}$ of particle s based on the grid map $m^{[s]}$, but are far away given the topological map $\mathcal{G}^{[s]}$ of s. $\mathcal{I}(s)$ is calculated by the formula:

$$\mathcal{I}(s) = \{x_t^{[s]} \in nodes(\mathcal{G}^{[s]}) \mid dist_{m^{[s]}}(x_{t1}^{[s]}, x_{t2}^{[s]}) < c_1 \wedge dist_{\mathcal{G}^{[s]}}(x_{t1}^{[s]}, x_{t2}^{[s]}) < c_2\} \tag{2.4}$$

Here, $dist_{m^{[s]}}(x_{t1}^{[s]}, x_{t2}^{[s]})$ is the length of the shortest path from x_1 to x_2 given the representation m. The distance between two nodes in $\mathcal{G}^{[s]}$ is given by the length of the shortest path between both nodes, whereas the length of a path is computed by the sum over the lengths of the traversed edges. Depending on the number of nodes in $\mathcal{I}(s)$, this distance information can be efficiently computed using either the $A*$ algorithm or Dijkstra's algorithm (both are pathfinding algorithms). The terms c_1 and c_2 are constants that must satisfy the constraint $c_1 < c_2$.

If one or more solutions for $\mathcal{I}(s)$ are found, it means that there are existent shortcuts from the current pose $x_t^{[s]}$ represented by the corresponding particle to the positions in $\mathcal{I}(s)$. These shortcuts

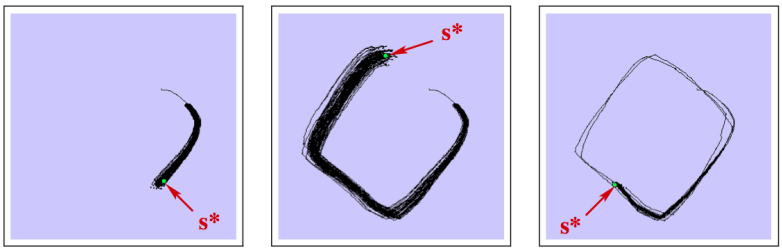


Figure 2.10: Evolution of a particle set and the topological map of the particle s^* at three different time steps, from [Stachniss et al., 2005]

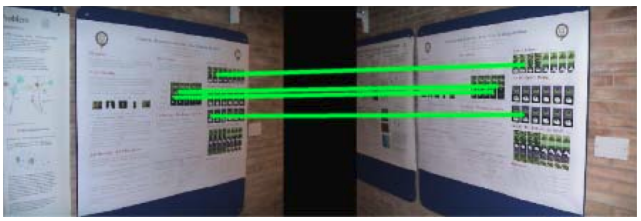


Figure 2.11: Correspondences found between features in very different views of a poster using MSER regions and SIFT descriptors, from [Newman and Ho, 2005]

represent edges that would close a loop in the topological map $\mathcal{G}^{[s]}$. Then, particle samples with unlikely trajectories (without solutions for $\mathcal{I}(s)$) are deleted in the resampling process, as seen in the right part of Fig. 2.10.

Although its efficiency is proven in [Stachniss et al., 2005], this can be time consuming in certain circumstances, such as big loops, because of the processing time required for a high number of particles.

2.4.2 Loop closure with visual features

Due to the popularity of visual sensors (such as monocular and others like depth and stereo sensors), researchers have introduced appearance-based methods to SLAM. Appearance-based SLAM maintains a topological representation of the environment where locations are described by images taken by the robot camera in a continuous manner [Liu and Zhang, 2012].

In [Newman and Ho, 2005] it is illustrated how visual features, extracted with computer vision algorithms such as MSERs (Maximally Stable Extremal Regions) and SIFT (Scale-invariant feature transform) descriptors, used in conjunction with laser scanning, can be used to a great advantage in the Loop Closing problem, therefore contributing to a long term robust SLAM. Also, to detect the possibility of having a loop closure, it is necessary to decide when and where to look. Searching only in the neighborhood of the vehicle is not robust in the face of gross vehicle error. The method proposes that to aid the Loop Closing task, the features must be salient, wide-baseline-stable and descriptive, as the ones in Fig. 2.11.

A method that makes use of visual features for closing the loop with Graphical SLAM, was achieved by [Folkesson and Christensen, 2007] and they find sets of features from both ends of the loop that can give the constraint around the loop. Then, they use the information in the graph to estimate the energy gain from closing the loop. If there is a gain, then they can make the match and close the loop by correcting the map and robot pose. This can be achieved with a stereo vision setup or simply a RGBD sensor, as the Kinect.

2.4.2.1 FAB-MAP

A method developed by [Cummins and Newman, 2007] called FAB-MAP is an approach to appearance-based place recognition. FAB-MAP compares images of locations that have been visited and determines the probability of re-visiting a location, as well as providing a measure of the probability of being at a new location. This information is used to detect loop closure occurrences, as the ones depicted in Fig. 2.12, where the images from the places indicated with circles are actually from the same location in

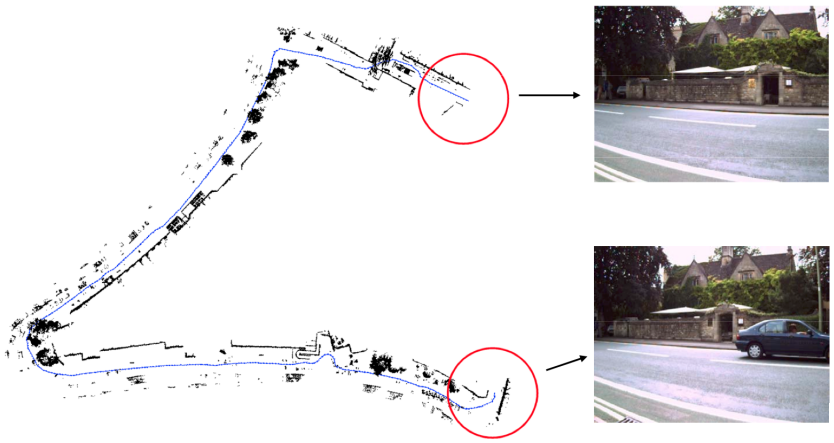


Figure 2.12: Example of loop closure detection using openFABMAP algorithm in an outdoor environment.

the world, meaning that a loop closure situation was detected.

This method adopts the Bag of Words technique, detailed in Appendix A, to represent images and match the appearance between the current visual scene and a past location, using it to compute not only the similarity of two given observations, but also the probability that they are originated from the same location. The probabilistic approach that is developed allows to explicitly account for the perceptual aliasing in the environment, where identical but indistinctive observations receive a low probability of being from the same place.

An open and modifiable source code that implements the FAB-MAP algorithm, based on the published work of [Cummins and Newman, 2007] was made by [Glover et al., 2012a] and it is available in [Glover et al., 2012b].

Both FAB-MAP and consequently openFABMAP require training data (e.g. a collection of images from a similar but not identical environment) to construct a visual vocabulary for the visual bag of words model, along with a Chow-Liu tree representation of feature likelihood. Chow-Liu trees method is proposed by [Chow and Liu, 1968] and it consists on approximating the joint distribution of a set of discrete random variables using a product of second-order distributions (the first-order tree dependence). Basically this method maps dependence relations between data in a tree.

Chapter 3

RobChair platform

In this chapter, the RobChair platform is presented on a technical level, along with the changes made in the restructuring process that are part of the new version, the RobChair 2.0. Changes were made both to the hardware architecture level and abstraction layer software.

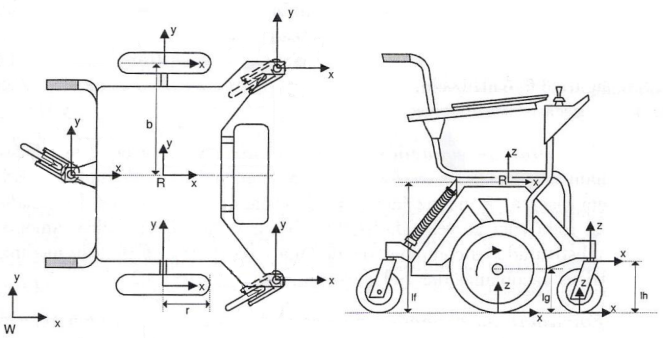
In the restructuring process, many components of the RobChair such as the steel frame, motors, encoders and a few others remained on the new platform, therefore a review of the previous architecture is carried out.

3.1 Previous architecture

The RobChair is composed by two motorized wheels with two casters in the front. There is also a fifth rear wheel connected to the back of the wheelchair with a damper used for stability. Figure 3.1 shows the RobChair mechanical structure and the associated system coordinates. Figure 3.2 presents a block diagram of the previous hardware control architecture.

The wheelchair is powered by two 12 V batteries feeding two permanent magnet DC motors with 24 V input voltage. These motors are coupled to two gearboxes with factor 1:10 (one complete wheel revolution corresponds to 10 complete motor revolutions). With the aid of these gearboxes, each wheel may have a nominal torque of 29,3 Nm. There were two power drivers to guarantee the independent and direct control of the motors. Two encoders have been coupled to the motor axis, providing the velocity feedback of each motor. One Hokuyo URG-04LX laser range finder with the capability to scan 240 degrees, is also integrated on the platform.

The overall RobChair communication system was constituted by communication modules based on TCP/IP communication protocol, and other communication modules based on fieldbuses, namely Controller Area Network (CAN) and Universal Serial Bus (USB), as depicted in Fig. 3.2. CAN was used for data transfer of small critical messages between devices, while USB is mainly used for devices that send or receive large amounts of data. A custom communication protocol, based on the



High	94,5 cm
Width	66 cm
Depth	120 cm
Wheel radius (r)	17 cm
Wheel to center (b)	30 cm
Maximum motor speed	1000 RPM
Gear Box reduction factor	1:10
Encoder pulses per revolution	500

Figure 3.1: Mechanical structure, system coordinates and relevant data of the RobChair

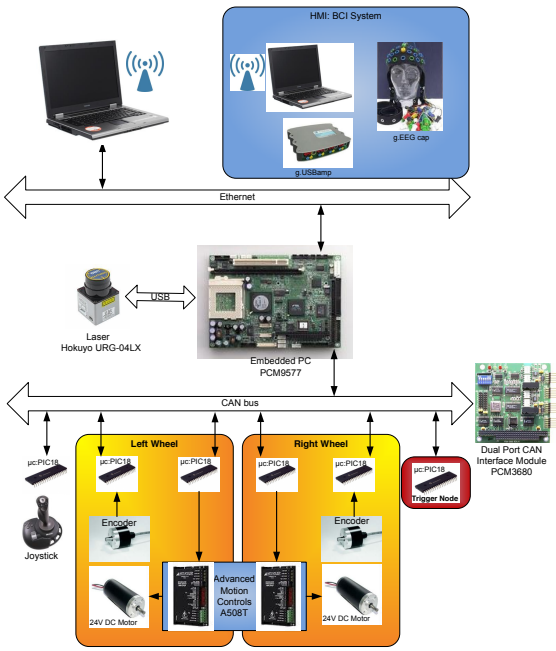


Figure 3.2: Previous RobChair system setup overview

Time-Triggered CAN (TT-CAN) protocol paradigm, was designed and implemented [Nunes et al., 2003, Silva et al., 2005]. All events were synchronized by a message sent from a trigger node based on a Microchip PIC18F258 micro-controller that synchronized all other micro-controller units. An Embedded-PC was responsible for giving some degree of intelligence to the robot. This computer was connected to distributed devices through CAN.

3.2 RobChair 2.0

The previous architecture of the RobChair had the advantage of being modular, with the CAN bus allowing new devices to be connected to the system, as seen in Fig. 3.2. However, this architecture became complex, due to several micro-controllers accessing the CAN bus, which contributed to a higher failure rate of the system, also making the learning curve of a new researcher quite high. Also, in order to add a new low level device to the RobChair, a new micro-controller board needed to be made and the trigger node reprogrammed, making it a more difficult and complex task that it should be. Additionally, the embedded computer was becoming obsolete, being too bulky and slow for todays standards, along with the old software system that resides in it.

So, with the objective of providing the RobChair with a simpler architecture, making it easier and fast to add new low level devices, while giving it more up to date and flexible components, (because the RobChair is mainly a research platform), the restructuring of the old RobChair took place to create the RobChair 2.0.

The new configuration is composed by the following hardware, numbered in Fig. 3.3:

- Two batteries of 12V each, (numbers 9 and 10);
- Two permanent magnet 24V DC motors, coupled to gearboxes with 10:1 reduction (ten turns of motor shaft for one wheel turn), (numbers 1 and 2);

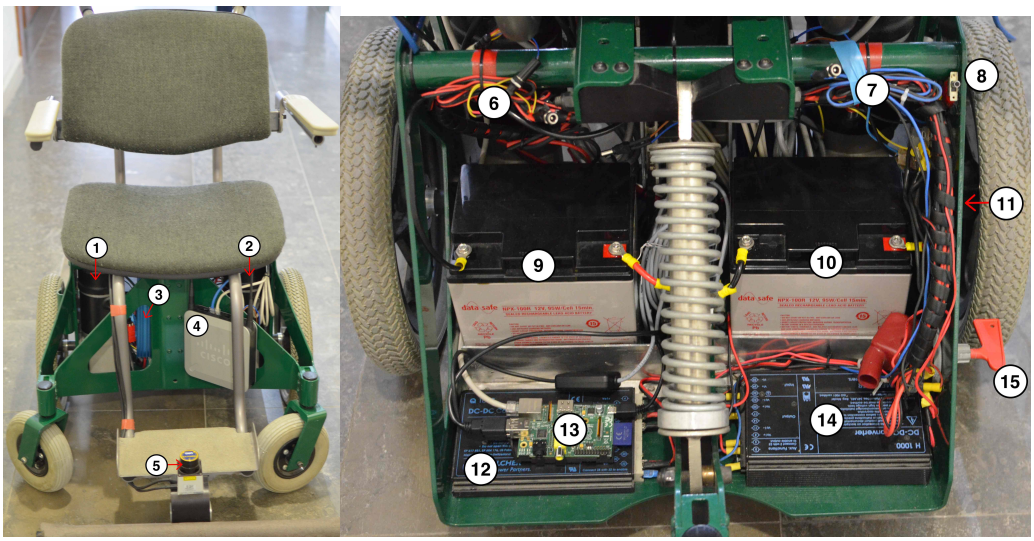


Figure 3.3: The restructured RobChair 2.0, front and back

- Two encoders, 500 pulses by turn, (coupled to motors, number 1 and 2);
- RoboteQ HDC2450 dual channel motor controller, (number 3);
- One 24V DC to 12V DC converter, (number 14);
- One 24V DC to 5V DC converter, (number 12);
- A micro-computer, the Raspberry Pi, Model B (ARM architecture), (number 13)
- One Ethernet/Wireless Cisco Router, (number 4);
- One Hokuyo URG-04LX laser range finder with four meters of range, mounted horizontally, (number 5);
- One Xsens Inertial sensor, (positioned in the middle point of the wheel axis).

From the previous list is clear that some components like the motors remained the same as before. Figure 3.4 shows the RobChair 2.0 system setup, where the software is now centralized in one low level micro-computer system.

The HDC2450 motor controller from RoboteQ is a commercially available controller that has a dual channel setup, for two independent motors, capable of providing 150A for each motor, at 50V maximum. It has also four types of operation modes: open loop, closed loop position control with a PID controller, closed loop speed control with a PID controller, and closed loop torque control with a PID controller. This controller is fully configurable and allows access to a large array of parameters, such as battery voltages, current being consumed by the motors, temperatures, encoder feedback and more, using the API defined by the manufacturer in [RoboteQ, 2012].

The controller has analog and digital inputs that allows devices to be connected directly, (e.g. a joystick), and used to drive the motors without additional electronics. The controller also receives and sends data through RS232, USB and CAN Bus. In this particular setup the USB connection was used to directly connect to the micro-computer. The encoders are directly powered by the 5V outputs of the HDC2450 controller and connected to the specific encoder channel ports.

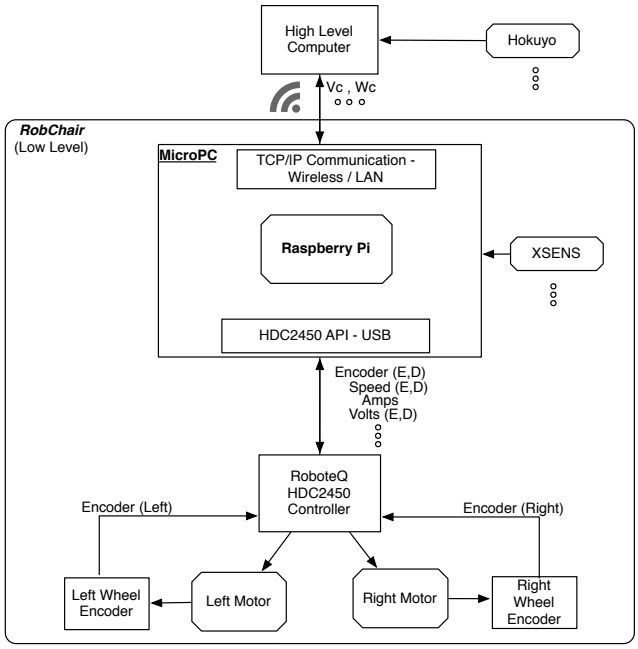


Figure 3.4: RobChair 2.0 system overview

Because of the DC to DC converters, the RobChair 2.0 includes three ready to use voltage outputs, 24V, 12V and 5V. The 24V are used to directly feed the HDC2450 controller. An additional 12V output is used to power on the controller. This 12V output also powers the Cisco router and has connections ready for other devices, such as a Kinect RGB-D sensor, later used (12V connections located at number 7, in Fig. 3.3). The 5V output powers the laser range finder and the micro-computer, with more available connections for other sensors (5V connections located at number 6, in Fig. 3.3). All this connections are easily expandable, being a matter of coupling the right connector to the power lines. Figure 3.5 shows the power connections between devices.

The micro-computer, a Raspberry Pi, is a credit card sized, and inexpensive ARM-based single-board-computer, running Linux, depicted in Fig. 3.6. Although the Raspberry Pi has less computational power than current PCs, it is still better than the previous RobChair embedded PC, being fast enough for a wide field of applications. The Raspberry Pi Model B is being used in the RobChair, and it has 512MB of RAM, 10/100 MBit Ethernet connection, two USB ports, a HDMI output, a composite video output, one 3.5mm audio jack output, a SD card connector and several I/O connectors, called the GPIOs, including SPI, I2C and RS232 connections, with reconfigurable pins for integrating other communication protocols if needed.

New devices can connect directly to the Raspberry Pi, or to a high level PC, a laptop for example if needed. Custom software developed for the RobChair is implemented in the Raspberry Pi, and it manages the connections and the operability of the system. Since this software is being written in standard C++, if in the future a more powerful micro-computer is needed, it could be changed just by replacing it and compiling the code.

Finally, RobChair 2.0 is equipped with three main switches. A main power switch, that cuts power from the batteries (number 15 in Fig. 3.3), a switch for the micro-computer, the Raspberry Pi (number 11 in Fig. 3.3), and a switch for the motor controller (number 8 in Fig. 3.3).

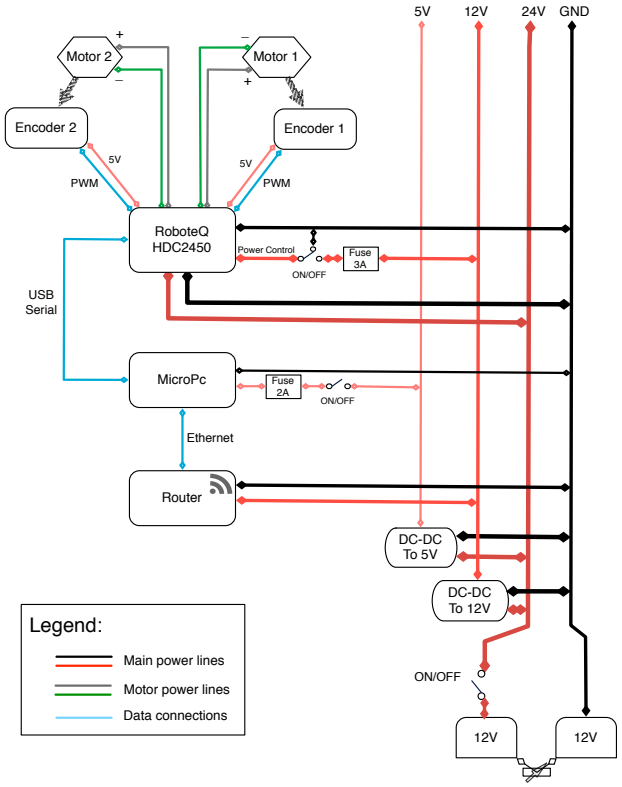


Figure 3.5: Power connections of the RobChair 2.0 devices

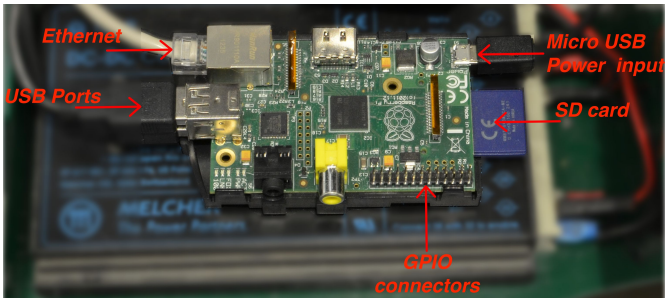


Figure 3.6: Raspberry Pi installed in the RobChair

3.3 PID controller

The PID controller, acronym for "Proportional, Integral and Derivative" controller, was first published in 1922 by Nicolas Minorsky [Minorsky, 1922], and since then it became one of the most used type of controller to a wide range of control applications. The fundamentals of the PID controller, including the steps necessary for this work, are described in Appendix B.

3.3.1 PI tuning

In order to have a working PI controller to our application, in this case to control the RobChair motors with the RoboteQ controller, the proper proportional and integral gains must be obtained. First a model of the system ($G(s)$) was obtained. The model of the system can be obtained through several techniques, being the most common:

1. Using a DC motor model that considers the internal parameters of the motor like the inertia of the rotor, the viscous friction constant, torque, electric resistance and inductance, and with these parameters simulate the motor response and obtain its transfer function;

2. Lifting the RobChair, in a way that the wheels turn freely and then retrieving the motors response to an applied command, denominated as a step response;

3. Moving the RobChair in the floor, retrieving the motors response to an applied step.

The first two methods allow a more realistic modelation of the motors response, and the friction forces must be inserted as an external disturbance. But in the case of the RobChair motors, the internal parameters are not fully known, so this method is not an option.

Because the third method already has the friction forces included in the motor response, and it also takes into account the internal behavior of the RoboteQ controller, it was the one used to obtain the speed model.

The input command is applied by the controller and the output is the speed values of the wheels in RPM. To experimentally obtain the transfer function of the motor response, a step corresponding to 10% of the max speed was applied to the motor controller, in open loop mode, with the RobChair in the floor. The acquisition rate was 100Hz. The raw data of the speed value evolution through time from the readings of the right and left encoders are shown in Fig. 3.7.

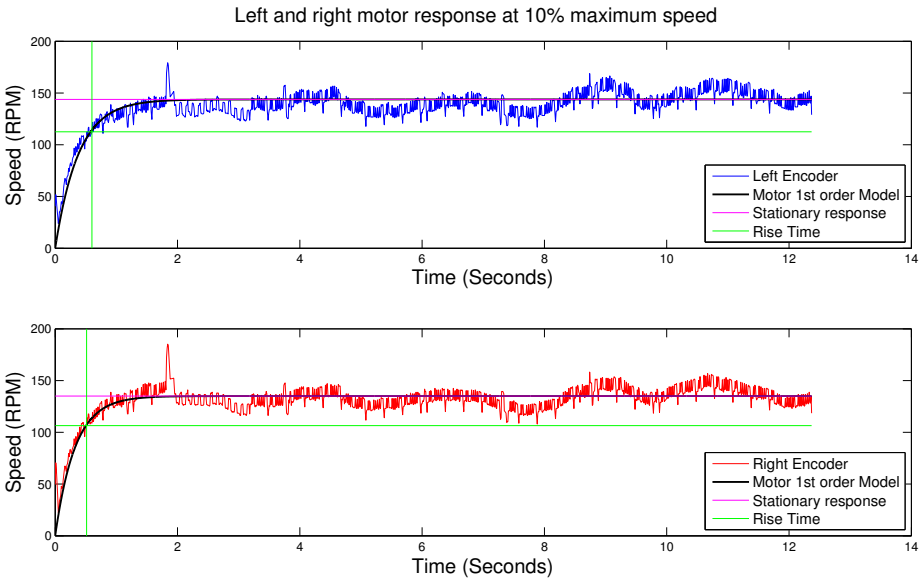


Figure 3.7: Motor response speed model

Processing the encoder data using MATLAB to obtain the rise time and the stationary response, as illustrated in Fig. 3.7, and knowing the input command, with (B.2) and (B.3), the following values were obtained:

$$\tau_S = 0.2681$$

$$G_{DC} = 1.4519$$

Assuming that both the left and right motors have similar responses, these values are valid for both. This way, all the variables for the first order model in (B.1) are known. The first order transfer function was then simulated in Simulink and tested for the same step, with the result shown in Fig. 3.7, the black curve, showing that the model follows accurately the real motor response for the same input command.

The next step is to determine the PI gains, with the process detailed in Appendix B for a generic full PID controller, where the PID transfer function ($G_s(s)$) is calculated, to obtain the second order system function (B.15) and the PID gain equations (B.17), (B.18) and (B.19).

In this case the derivative gain is set to zero ($K_D = 0$), making this a PI controller, with no derivative gain. Accordingly to the required system response, the values ζ and ω_n must be chosen. Since the RobChair is a system for assisting people, it should not make sudden movements. For a smooth acceleration, a under-damped system should be designed, and to do that ζ must be below 1, in this case the value 0.9 was chosen.

For the natural frequency, a value of $2\,rads/sec$ was set. Considering these values and equations (B.17) and (B.18), the final gains were obtained:

$$K_P = 1.7907 \tag{3.1}$$

$$K_I = 2.7549 \tag{3.2}$$

3.3.2 PI validation

Using the first order model of the motor and the obtained PI gains in Simulink (Fig. B.2), the system response time can be observed in Fig. 3.8as a black line, for an input reference of 50 RPM and then 100 RPM, where it is possible to observe a rise time of about half a second. Note that 50 RPM on the motor shaft will correspond to a slow speed of 5 RPM on the wheels, and $0,5\,secs$ of rise time for this speed represents a very "soft" response, as desired.

To validate the obtained PID gain values, a test was performed on the RobChair 2.0 platform. The values were set to the RoboteQ motor controller and the control scheme changed from open loop mode to close loop speed control mode. Next a speed of 50 RPM and then 100 RPM was set, similarly to the simulation. The results are shown in Fig. 3.8, where it is possible to verify that the motor response corresponds to the simulated one, with a rise time of about half a second.

3.4 Software architecture

To provide the RobChair with an abstraction layer, a software for its internal micro-computer, the Raspberry PI, was built. The main goal was to provide the high level devices, a way to access the RobChair internal parameters and low level sensor readings. It also gives the high level devices the capability of setting the internal parameters of the RobChair controller and its devices. In Fig. 3.9 it is shown how devices interact in the new RobChair, and which corresponding protocols are being used. Devices depicted in green establish the RobChair network, using the standard sockets protocol to exchange information between them, using either a ethernet cable or a wireless connection. A

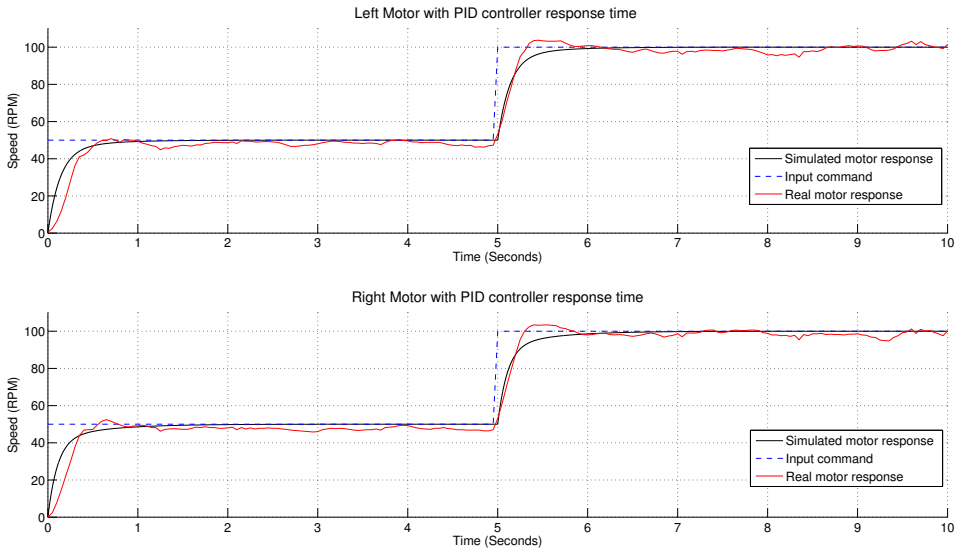


Figure 3.8: PID validation results

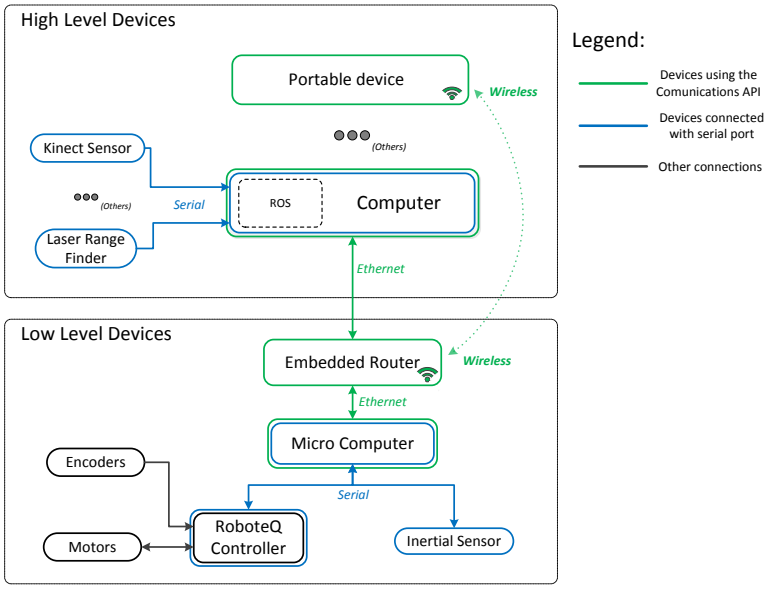


Figure 3.9: Types of device connections in the RobChair

communications API (Application Programming Interface) is build into the abstraction layer of the RobChair.

This abstraction layer should be simple and robust, while being trouble free, to keep the RobChair from having unexpected behavior. This software is called the "RobChair Framework".

3.4.1 The RobChair framework

To understand the requirements of this abstraction layer and device management, a requirement analysis was performed, as follows:

- Modular software system;

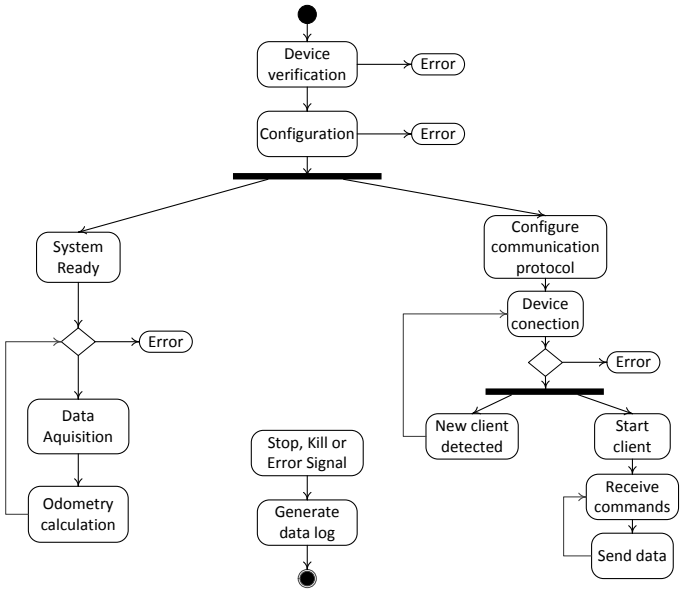


Figure 3.10: RobChair Framework prototype

- Written in standard C++, using standard system libraries and protocols;
- Capable of connecting to several devices using *Ethernet/Wireless;*
- Using *threads* to manage new connections and the communication framework;
- Mutual exclusion access to devices, in this case the RoboteQ HDC2450 controller and the Xsens inertial sensor;
- Targeted to run on a GNU/Linux distribution;
- Using POSIX *semaphores* and *threads*.

A prototype of the framework, developed according to the presented requirements, is shown in Fig. 3.10. This project is similar to a previous work carried out on another ISR robot platform, the IS-Robot. Therefore, the following framework proposal is based on the ISRobot 2.0 Framework, developed together with Luís Garrote [Garrote, 2013].

The RobChair 2.0 Framework integrates two active main modules, or tasks. The Main Task and the Communication Management Task (see Fig. C.1 and C.2 from Appendix C). The third task shown in Fig. C.3, for Client Communication, is only activated when a new device is connected and actively communicating. The Main Task is responsible for initializing the RoboteQ HDC2450 controller class, that makes use of the internal API of the controller. This allows the framework to exchange information with the controller. Then the Communication Management Task starts on a separate *thread*, and after that, it starts logging internal parameters from the Controller. A routine that continuously acquires and computes odometry and other sensorial data, with a controlled frequency of 20Hz (every $50ms$), is then started. Here the direct kinematics of the RobChair, a differential robot, are also computed. If the Main Task stops running, all other tasks are closed and the activity log is written to the disk before terminating.

The Communication Management simultaneously creates a socket connection and starts a 10Hz routine where it checks for new devices requesting a connection. Upon a new detected connection, a

Table 3.1: RobChair 2.0 communications API

Description	Command
Ask to send data measurements continuously	**!MEAS**
Ask to send pose measurements continuously	**!POSE**
Stop sending data measurements	**~MEAS**
Stop sending pose measurements	**~POSE**
Close all connections	**CLOSE**
Close all connections and exit the system	**EXIT**
Reset pose	**RESET**
Send velocities to robot	**VEL=V;W**
Correct pose info	**CPOSE=X;Y;Teta**
Send raw speed values to motor controller	**M=M1;M2**

thread for answering the client requests is created, and the Communication Management task continues searching for new clients to connect. If the socket connection stops, for example due to a network failure, the task terminates. The Client Communication Task grants the access of the system resources to the client, and communicates with it using a defined API, shown in Table 3.1, while the connection is active. Although the motors can be independently controlled using the command: **M=M1;M2**, this is not the intuitive way the RobChair is intended to be controlled, but instead using the linear v and angular w velocity commands: **VEL=V;W**. So, the software also implements a conversion between these commands and the motor commands to the controller, using:

$$w_{right} = \frac{v + b * w}{r} \qquad w_{left} = \frac{v - b * w}{r} \tag{3.3}$$

where b is half the distance between the two wheels, r the wheel radius and w_{right}, w_{left} are the individual speeds of the wheels in rad/s. The speed is then converted to RPM. Also, the software sends odometry data, extracted from the encoder readings and sent in linear and angular speed values. First, the linear displacement for each wheel is given by:

$$\triangle_{wheel} = \frac{E}{C} 2\pi r \tag{3.4}$$

where E is the encoder pulses increment, C the total encoder pulses by wheel revolution. Then the linear displacement of the RobChair is given:

$$\triangle_{linear} = \frac{(\triangle_{Left\,wheel} + \triangle_{Right\,wheel})}{2} \qquad \triangle_{angular} = \frac{(\triangle_{Right\,wheel} - \triangle_{Left\,wheel})}{d} \tag{3.5}$$

If a client asks the robot pose directly at instant k, (x_k, y_k, θ_k), the software also computes the kinematics of the RobChair [Garrote, 2013, Lopes et al., 2007], considering the differential robot model as follows:

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + \begin{bmatrix} \cos(\theta_k) \\ \sin(\theta_K) \\ 0 \end{bmatrix} \triangle_k + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \triangle\theta_k \tag{3.6}$$

where $\triangle_k$ and $\triangle\theta_k$ are the linear and angular displacements.

Chapter 4

ROS integration

ROS is an open-source, meta-operating system that that stands out comparatively with others presented in Table 2.2, because it has been greatly adopted by the community (research groups, commercial companies, governmental organizations), and consequently, tools and algorithms for ROS are added frequently. Also, comparatively to others, ROS is well documented and there are many ready tutorials to help the user. In sum, ROS was found suitable to integrate the RobChair 2.0 due to several factors, namely: the available documentation, broad adoption, the developer community and ROS experience gathered in the past. A technical overview of ROS is provided in the appendix D.

4.1 Simulation

Testing existent and developed nodes on the RobChair itself can be dangerous due to several reasons, such as damaging the robot, collisions, or even injure people. It can be time consuming as well, because of preparing the robot, the workspace, reseting positions, etc. Additionally, control over the workspace can be very limited, for example in ISR ground floor, to test a SLAM algorithm without people or objects in day time is almost impossible.

Due to this real world restrictions comes the necessity of using a simulator. One that provides a way to simulate the robot, sensor readings, robot behavior and even control the workspace, grating that the tested algorithms will perform similar to the real world. Bearing this in mind, from the available simulators integrated in ROS, being the most common ones Stage and Gazebo, the Gazebo simulator was chosen, because of its capability of simulating a 3D world and having a realistic physics engine, that simulates real world physical interactions. Stage, on the other hand, only provides simulation in a 2D world and provides limited physical interaction. Gazebo was originally designed to aid in the development process of algorithms for robotic platforms. By realistically simulating robots and environments, the code for a physical robot can be executed on an artificial version. It is capable of simulating a population of robots, sensors and objects, in a 3D world. It generates both realistic sensor feedback and physically plausible interactions between objects (it includes an accurate simulation of rigid-body physics). A set of plugins create a seamless interface between Gazebo and ROS, allowing developers to easily switch between hardware and simulation. Gazebo is under active development at the Open Source Robotics Foundation [Robotics-Community, 2013].

4.1.1 Virtual workspaces

To perform tests in defined and controlled environments and also in places that resemble the real scenarios that the robot has to go through, two 3D models were created for gazebo using the Google SketchUp 3D modeling software [Google, 2013]. First, a small model, seen to the left of Fig. 4.1 shows an enclosed small area with several characteristics, such as corridors, open and closed areas, and small

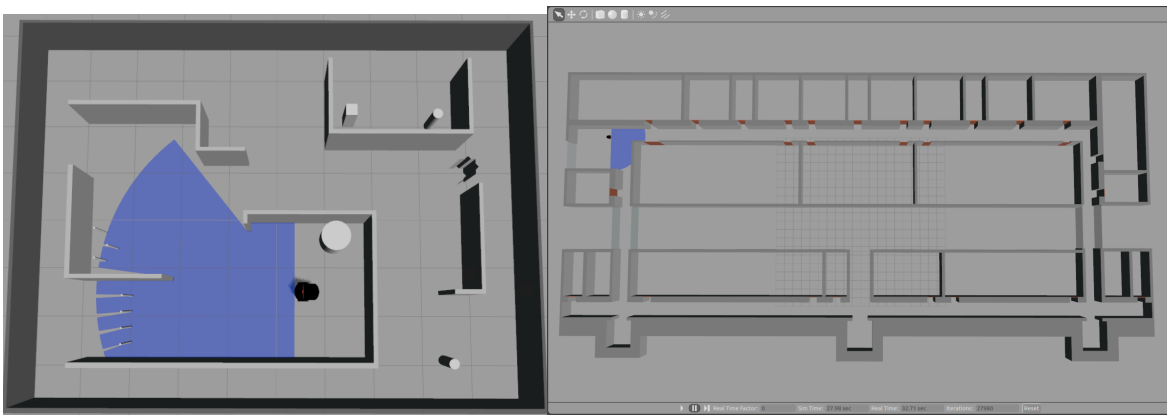


Figure 4.1: Small test field 3D model in Gazebo simulator to the left and a 3D model of ISR ground floor in Gazebo simulator to the right

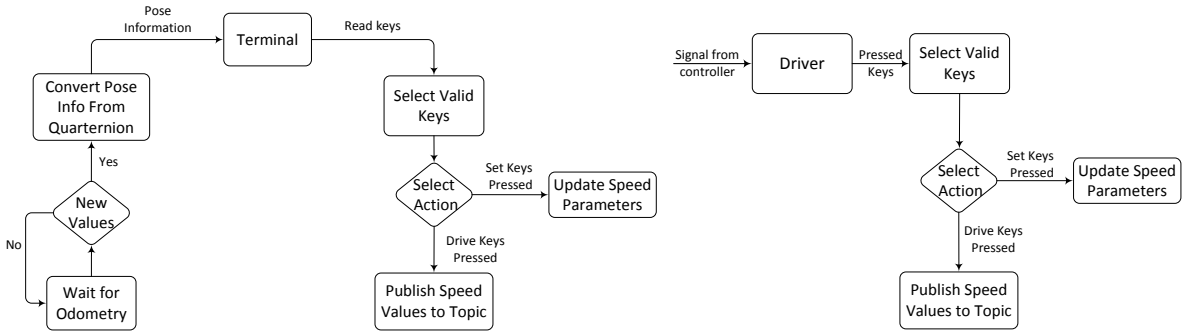


Figure 4.2: Keyboard (left) and controller (right) teleoperation ROS nodes

obstacles, with the goal of testing several methods. Second, a 3D representation of ISR ground floor was created, as seen to the right of Fig 4.1, because that is the RobChair workspace, where algorithms need to perform, and with this virtual version it is possible to test it before using the real platform, with the advantage of having a controlled environment, as opposed to the real rooms and corridors. This ISR model was created based on real measurements.

4.2 Teleoperation nodes

To remotely control the simulated robot, and the real RobChair, two nodes in ROS for teleoperation, were written. The first one allows to control the robot with a keyboard and the second one with a game control pad, both by sending speed commands. These commands are published to the *command_velocity* topic, the standard message topic for controlling robot velocities in ROS. Users can set both the desired linear and angular speed for the robot, and then drive it, using the arrow keys, at the set speeds.

The keyboard teleoperation node (*keyboard_teleop*), depicted on the left part of Fig. 4.2, reads the keyboard information directly from the terminal. It has also the ability to receive odometry feedback from the robot, so the user can directly see the updated pose on of the robot on the screen while it is moving. The game control pad node fulfills the same purpose as the previous keyboard node, but without the odometry feedback information. Although, internally it needs to load a driver for the

controller keys to be properly detected by the system. This node, called *gamepad_ teleop* is depicted in the right part of Fig. 4.2.

4.3 The RobChair ROS driver node

To integrate the ROS platform with the RobChair, a "driver" or bridge between the RobChair framework and the ROS framework needed to be developed. In the previous Chapter, the RobChair framework implemented in the micro-computer is able to communicate to the outside using a socket protocol (using TCP/IP), either by ethernet or wireless connection (Fig. 3.9). The commands are exchanged by a previously defined communication API (Tab. 3.1). Giving these requirements, a node for ROS that communicates with the RobChair (called *wheelchair_ talker*), acting as a client for the RobChair Framework, needs to implement the bridge between the socket protocol and the ROS message topics using the predefined API, as seen in Fig. 4.3.

To perform this, the socket connection is carried out making use of the POSIX system socket functions (*sys/socket.h*), that allows information to be transmitted over the TCP layer using the IP address of the server, in this case the IP address of the RobChair micro-computer. After a successful connection, the node instructs the RobChair that wants to keep receiving data measurements from the devices connected to the micro-computer. Once data is received, if it is from the inertial sensor, it generates a ROS message with the information, and publishes it. If the data is from the odometry, it computes the displacements between this reading at t_k and the previous at t_{k-1}, in pose values, x, y and the angle θ (also known as direct kinematics):

$$\Delta t = t_k - t_{k-1} \tag{4.1}$$

$$\Delta x = (V_x * \cos\theta) * \Delta t \tag{4.2}$$

$$\Delta y = (V_x * \sin\theta) * \Delta t \tag{4.3}$$

$$\Delta\theta = V_\theta * \triangle t \tag{4.4}$$

The variables $\triangle x$, $\triangle y$ and $\triangle\theta$ are used to calculate the robot new position, by updating the transformation matrix between the start point (called *odom* link in ROS) to the base of the RobChair (called *base_ link*). It also computes the transformation matrix from the base of the RobChair and the position of the laser range finder (called *laser* link), using known measurements from the RobChair structure. It publishes these transformations to the *tf* ROS message topic, that keeps track of multiple coordinate frames over time, by maintaining the relationship between coordinate frames in a tree structure buffered in time. At the same time that this information is being published, the node watches the command velocity message topic in ROS for new commands, and reroutes them to the RobChair using the "VEL" defined command in the micro-computer API. The concurrent waiting for new velocity commands, is performed by making use of a ROS callback. A callback in ROS is a subroutine that is executed asynchronously when triggered by some event, in this case a new message

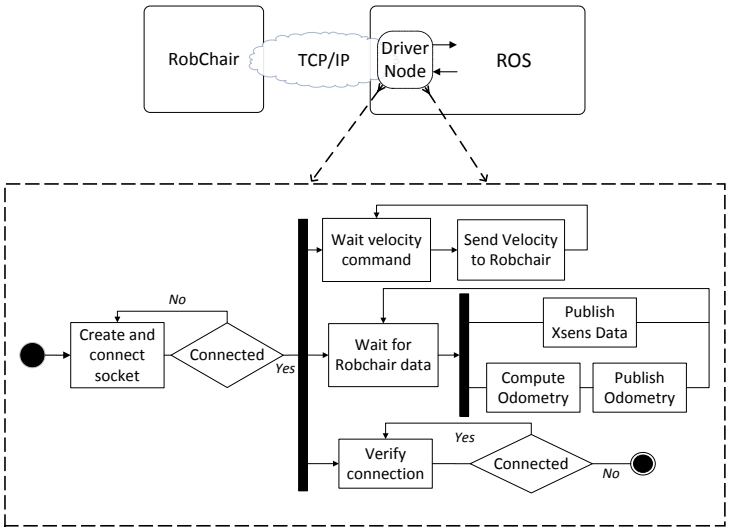


Figure 4.3: RobChair Driver Node for interoperability with ROS

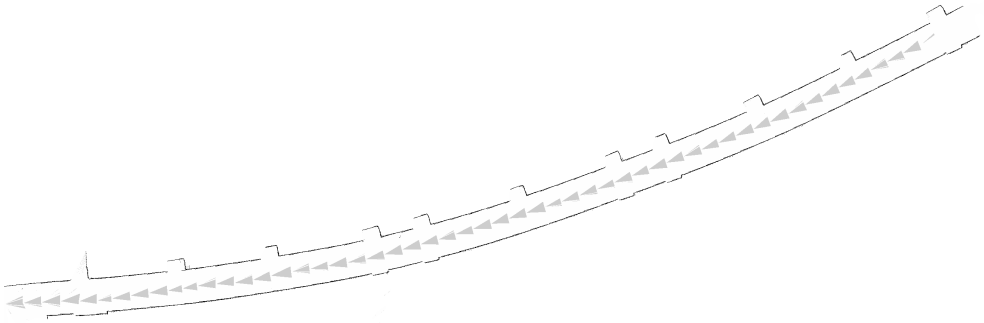


Figure 4.4: Straight corridor mapped with uncalibrated odometry

in the *command_velocity* topic.

This works while the socket connection is valid and the node is still running. Data from the laser or camera in the RobChair do not go through this node, that is because they connect directly to the high level computer, and ROS provides ready to use drivers for these sensors, which publish the relative data topics to the ROS system.

4.4 Validation

With the *wheelchair_talker* and teleoperation nodes, it is now possible to make use of ROS tools to visualize and store data from the real platform. Early tests confirmed that the nodes are working as intended. Making use of the RVIZ, a tool to visualize several topics being published, it is possible to see the the evolution of the RobChair pose, based on odometry readings. By plotting the laser readings according to the pose evolution provided by the odometry, it is possible to conclude that the odometry readings do not represent the true RobChair motion, as shown in Fig. 4.4, that corresponds to the RobChair motion in a straight corridor. This means that the odometry information is not properly calibrated, and before continuing to implement higher level methods, such as SLAM, some techniques can be applied to mitigate this problem.

4.4.1 Odometry error study

Odometry is the most widely used method for determining the pose of a mobile robot. In most practical applications odometry provides easily accessible real-time positioning information in-between periodic absolute position measurements. The frequency at which the (usually costly and/or time-consuming) absolute measurements must be performed depends to a large degree on the accuracy of the odometry system.

Odometry errors can be non-systematic or systematic. Non-Systematic odometry errors are the ones caused by interaction of the robot with unpredictable features of the environment. For example, irregularities of the floor surface, such as bumps, cracks, or debris, will cause a wheel to rotate more than predicted. Systematic errors are vehicle-specific and don't usually change during a run. Thus, odometry can be improved significantly by measuring the individual contribution of the most dominant error sources, and then counter-acting their effect in software [Borenstein and Feng, 1996]. Determining the systematic odometry errors of a mobile robot is essential to accurately compute the robot pose, which is determined based on encoder data. With the error information it is possible to improve reliability by reducing the error that builds up between the real robot position and the estimated position using the encoder data. It can be very important during the process of map building because having an accurate robot pose estimation, it can reduce the mapping errors and the difficulty of detecting loop closure, representing a significant performance improvement.

Some odometry calibration techniques were already studied in [Rekleitis, 2003, Borenstein, 1994, Larsen et al., 1998, Borenstein and Feng, 1996], but the most standard one is still the often called "method of the square" by [Borenstein and Feng, 1996]. Briefly, in this method, a differential drive robot is instructed to go on a pre-programmed square path of 4x4 meters, that consists on straight paths and 90 degree turns. The cumulative error is then extracted by observing a reference wall and measuring the final robot pose.

Another method, from [Rekleitis, 2003], performs a different approach to the problem, relying on laser measurements and a few landmarks to estimate the rotational and translational odometry error. Also, less space is needed to perform this study.

4.4.1.1 Method from Rekleitis

In this method, odometry errors are modeled by performing a series of controlled tests with the robot, by separating rotational movement errors from translational ones. In both cases, the robot is placed in a 'C' shaped enclosure, consisting of four walls, as seen in Fig. 4.5, that shows in green the laser readings with the RobChair in the start pose, $(0, 0, 0)$. The intersections of the four walls provide three geometric landmarks (highlighted by circles in Fig. 4.5), and then the orientation of the four walls in world coordinates should vary by the same amount of the robot's rotation, as depicted in the example shown in Fig. 4.5. The detection of the required landmarks was performed using MATLAB.

Lines are approached using (4.5), where the required parameters are extracted by computing the points provided by the laser data.

$$y = mx + b \tag{4.5}$$

The three corner landmarks correspond to the occurrence of intersections between the detected lines

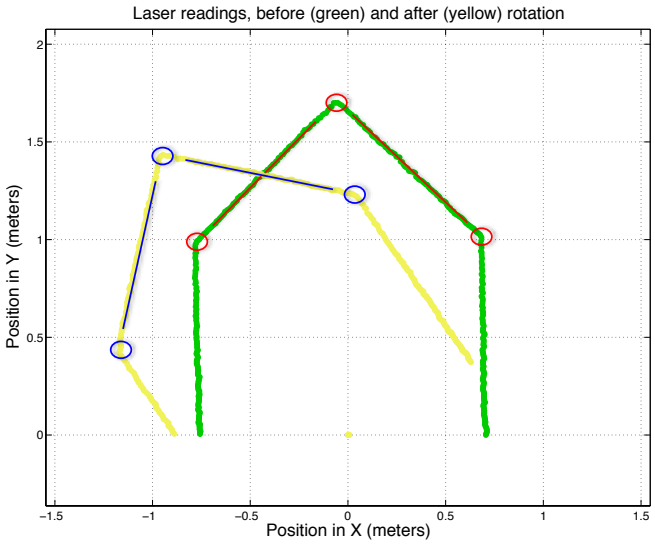


Figure 4.5: Laser readings of the four walls providing five landmarks, shown in blue and red. Initial position in green and yellow after rotation.

that provide three estimates (indicated by the red and blue circles). And the two top walls provide two more estimates (indicated by the red and blue lines). These five landmarks are used for both rotation and translation analysis, because these landmarks are always in the laser field of view. To estimate the errors, the five landmarks are first detected for the initial pose of the robot (the ones in red), and then after the robot motion, the five landmarks are detected again (the ones in blue), as it is possible to see in Fig. 4.5, in this case, for rotational movements.

To compute the rotational error, a similar procedure is repeated using different motion parameters, namely speed and rotation angle. The rotational error was determined using six rotational angles (-50º, -30º, -10º, 10º, 30º and 50º), at three different speeds ($0.1m/s$, $0.2m/s$ and $0.4m/s$). Although, these predefined angle values are difficult to obtain with the real robot. So, errors will be compared with the ground truth laser readings. For example, a desired rotation of -50º is actually translated in a "reported rotation" of -55.2344 degrees, given by the odometry, and of -53.0241 degrees of "real rotation" given by the laser readings. This situation happens because there was a delay between the RobChair stop command, and the time it actually stopped.

Results are shown in Fig. 4.6, with the rotation angles in the X-axis adjusted to the average laser reported values, and the odometry errors in the Y-axis. The 'o', 'x' and '+' stand for the three different test speeds. The dashed lines are the average errors for each speed. It is clear that for larger rotation angles there are larger errors.

By calculating a constant correction factor and then multiplying it by the received rotation measurement (angular value of the received displacement) from the RobChair, it is possible to mitigate the cumulative errors of the odometry readings in real time.

To obtain the constant K, first, an average of the measured errors for each rotation angle, and for each speed was performed. From these values, it is possible to calculate the correction factor, as follows:

$$K_{rot} = \left| \frac{Reported\,rotation}{Real\,rotation} \right| \tag{4.6}$$

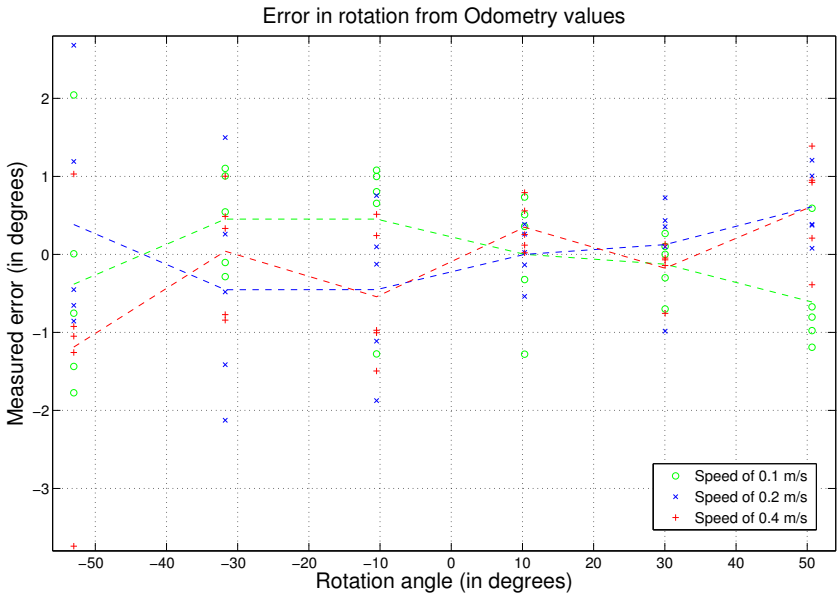


Figure 4.6: Odometry rotational errors

Because the factor can not be changed in real time on the RobChair driver node for each speed and rotation displacement, an average of all K was used:

$$K_{rot\_avg} = 1.0269 \tag{4.7}$$

The correction is made on the ROS driver node, before computing the odometry transformations, as follows:

$$V_{\theta\_new} = V_{\theta} * K_{rot\_avg} \tag{4.8}$$

Where V_{θ} is the received angular displacement from the RobChair and the V_{θ_new} is the new displacement value to be added to the odometry transformation matrix in ROS. For the translational error, the same setup was used. The robot moved about one meter at the same three different speeds as before. Each test was repeated ten times. Like before, the errors are given by relating the "reported rotation" from the RobChair to the "real rotation" obtained based on the laser data.

In Fig. 4.7, the distance errors are shown for the three different speeds, with the dashed line representing the average. It is clear that at a larger speed, a larger error is obtained. This happens because of a significant wheel slippage that occurs when breaking at a higher speed. Because of this, the $0.4m/s$ speed values were discarded.

Now, a constant K_{trans_avg} is also calculated, the same way as before, and applied to the linear displacement received from the RobChair, as follows:

$$V_{lin\_new} = V_{lin} * K_{trans\_avg} \tag{4.9}$$

By observing both translational errors in X and Y it is possible to see an emerging pattern, that should be corrected by the applied constants. The reported positions are shown in Fig. 4.8, with the uncalibrated data in the right, and the laser measured positions in the left (also called ground truth).

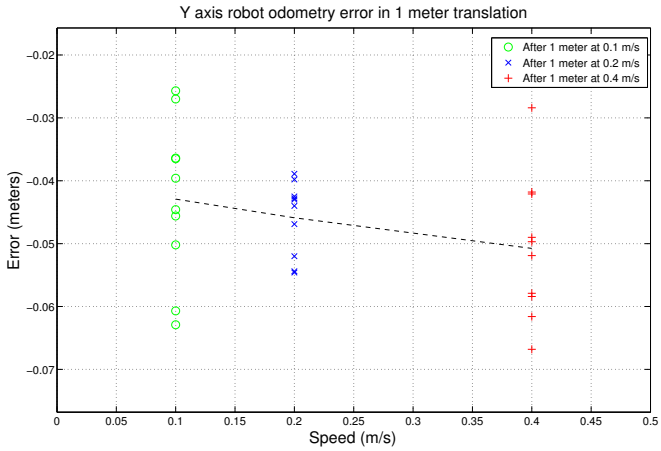


Figure 4.7: Odometry translational errors

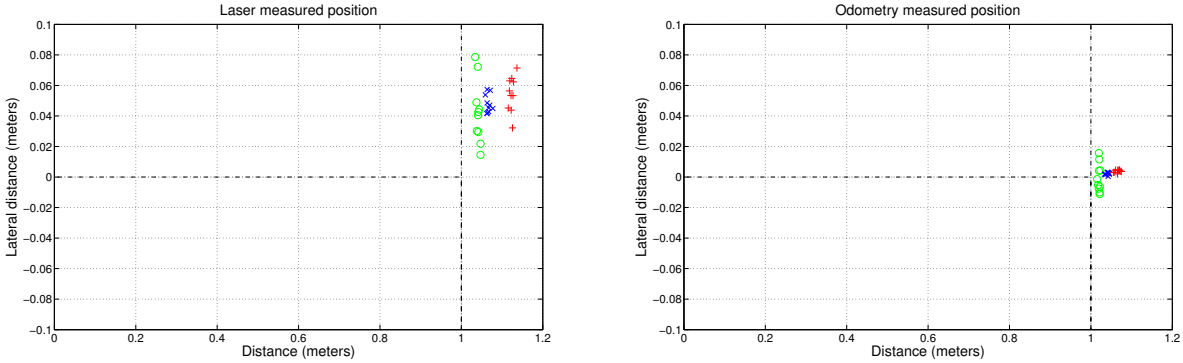


Figure 4.8: RobChair position according to ground truth data (left) and uncalibrated odometry data (right)

Analyzing both, it easy to observe why the mapped corridor of Fig. 4.4 was bent to the left. On the left image of Fig.4.8, it is possible to observe the RobChair true position going to the right (positive values in the Y axis indicate real displacement to the right and vice versa). When going on a corridor, the user, or a navigation method, will pull the RobChair back to the center (left), making the RobChair odometry register a false bend to the left, generating the observed error in Fig. 4.4.

Figure 4.9 shows the mapping of an ISR corridor after the odometry calibration. As it can be observed, the bend effect was eliminated.

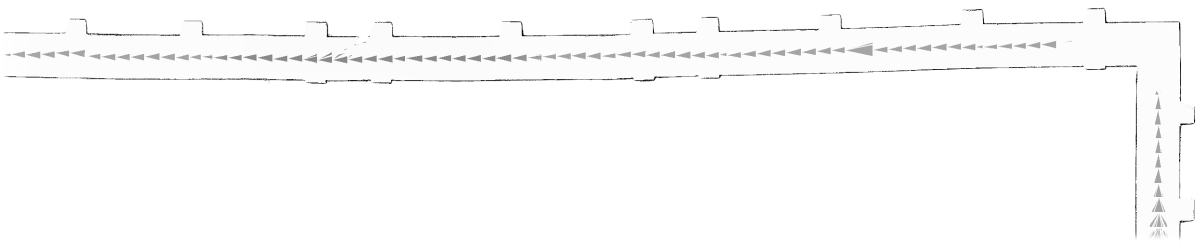


Figure 4.9: Test in ISR corridor after odometry correction

Chapter 5

SLAM benchmarking

In this Chapter several SLAM methods available in ROS are described and evaluated, in order to find the most appropriate one to implement in the RobChair.

5.1 Evaluated SLAM methods

The ROS repositories contain several contributions from different researchers, with many implementations of SLAM algorithms already available to the community. But only a few are widely used and regularly supported. Methods like *Gmapping*, *Hector SLAM*, *Karto* and *RGBD-SLAM* (names of the stacks in ROS) are under constant updates and are supported in the latest ROS versions. Since they are very different, these four methods were chosen.

5.1.1 Gmapping

GMapping implements the FastSLAM method (previously described in Chapter 2), with a highly efficient version of the Rao-Blackwellized particle filer to learn grid maps from laser range data, and it is an open project at the OpenSLAM community [Grisetti et al., 2013] coded in C++, that has been wrapped into a ROS package.

This implementation was proposed in [Grisetti et al., 2007], presenting an approach to compute an accurate proposal distribution that takes into account not only the robot motion but also the most recent observation. This drastically decreases the uncertainty about the robot's pose in the prediction step of the filter. Furthermore, in the same work an approach is applied to selectively carry out re-sampling operations which significantly reduces the problem of particle depletion.

Gmapping has a great ROS support and documentation in comparison to the rest of the algorithms tested in ROS. Its functional requirements are the odometry information of the robot platform and a horizontally mounted laser range finder. This version is optimized for long-range laser scanners like SICK LMS or PLS scanner. Short range lasers like Hokuyo laser scanner may not be perfectly suitable for working with the standard parameter settings [Grisetti et al., 2005, Gerkey, 2013].

5.1.2 Hector

Hector is for an Heterogeneous Cooperating Team of Robots, and it is a flexible and scalable SLAM algorithm for different search and rescue scenarios, mainly used in USAR (Urban Search And Rescue) robots. This Method can be used without odometry information from the mobile platform and it is commonly used in platforms that exhibit *roll-pitch-yaw* motion, such as quadcopters. It leverages from the high update rate of modern LIDAR systems like the Hokuyo UTM-30LX. Hector SLAM fuses sensor information using an Extended Kalman Filter [Kohlbrecher et al., 2012].

The created map is represented by a 2D grid, holding the probability of cell occupancy. It accesses map data on non-integer coordinates using bilinear filtering, which is an approximate method, even though it is fast. It also caches recently accessed grid points. This method does not provide explicit loop closing ability, and it is available in ROS since 2010.

5.1.3 Karto

The Karto SDK is currently an advanced development project within the AI Center at SRI International, and both a commercial and a free LGPL (GNU Lesser General Public License) version are available. But, these newer versions are not embedded in ROS anymore, only a older version is available in ROS [KARTO-Team, 2013].

Karto implements the GraphSLAM algorithm, which relates two observations if they hold information about the same observed landmark, by making use of sparse information matrices, as described in Chapter 2. Karto is known for being fast and lightweight, making use of odometry and laser range finder data. This mapping library also contains all important building blocks for 2D navigation: a scan matcher, pose graph, loop detection, and occupancy grid construction. Unfortunately, being a commercial software, the source code of the latest versions is not provided and little to no information is given about its internal configuration.

5.1.4 RGBD-SLAM

This method is included in ROS as a package that can be used to register the point clouds from RGB-D sensors such as the kinect or stereo cameras to perform 3D mapping of environments, and it is also an open project at the OpenSLAM community [Endres et al., 2013].

It implements the 6D SLAM algorithm approached in Chapter 2, and it allows to quickly acquire colored 3D models of objects and indoor scenes with a hand-held Kinect-style camera. It provides a SLAM front-end based on visual features such as SURF or SIFT (user can choose) to match pairs of acquired images, and uses RANSAC to robustly estimate the 3D transformation between them. The resulting camera pose graph is then optimized with the SLAM back-end HOG-Man to reduce the accumulated pose errors (Fig. 2.8) [Engelhard et al., 2011].

Tests show that this method requires a huge computational power and works best with CUDA GPU acceleration technology. If parameters set accordingly, it can perform on real-time on an average computer but with less accurate results.

5.2 Evaluation metrics

Even with dozens of different techniques to tackle the SLAM problem have been presented, there is no gold standard for comparing the results of different SLAM algorithms. In the community of feature-based estimation techniques, researchers often measure and compare the estimated distance between a landmark location and the true robot location (ground truth), analyzing the error between both. As illustrated in [Kummerle et al., 2009], comparing results based on an absolute reference frame can have shortcomings. In the area of grid-based estimation techniques, people often use visual inspection to compare maps or overlays with blueprints of buildings. This type of evaluation becomes more and

more difficult as new SLAM approaches show increasing capabilities and thus large scale environments are needed for evaluation [Kummerle et al., 2009].

Two methods are used in [Sturm et al., 2012], to evaluate RGB-D SLAM: the Relative Pose Error (RPE) and the Absolute Trajectory Error (ATE). RPE measures the local accuracy of the trajectory over a fixed time interval Δ, corresponding to the drift of the trajectory. The ATE, is where trajectories are aligned and compared with the absolute distances between the estimated and the ground truth trajectory.

In [Balaguer et al., 2007] SLAM algorithms (including Gmapping) are evaluated visually, by observing the map results of simulation and real world test, performed at the same time, with the same control input.

With all these various attempts to properly evaluate a SLAM method, a standard method does not exist, mainly because of the diverse nature of the output data of all these methods. That is, some SLAM approaches create a 3D map, while others a 2D one, and even in the 2D map representation, there are several possibilities. The same diversity applies for trajectory/pose data. So, the best way to evaluate a SLAM algorithm always depends on the nature of the algorithms and data available, and often custom solutions are made, like the one implemented in MATLAB for this work, which is described in the next section.

5.2.1 Evaluating SLAM maps

With the goal of testing some of the described SLAM methods in order to find the most suitable one for the RobChair, a method for comparing them was required. Giving that a simple way of obtaining a ground truth data with the RobChair was not easily accessible, the method was tested on a simulated environment, described before in section 4.2 of Chapter 4. Although good pose estimation is important in SLAM, its evaluation was not specifically performed, because of the lack of ground truth data and also because a good pose estimation helps to reduce the error in the generated map of the environment. Giving that ground truth data is easy to obtain in simulation, because of the environment models, the implemented method depicted in Fig. 5.1 performs a comparison of the final map given by the SLAM method with the 2D version of the simulated environment model.

The method for comparing map images was created with MATLAB, and it has with two ways of performing the comparison. One way is by doing point to point matching of the two maps, and the other is by detect corners on the map, using the corner feature extractor in MATLAB, from the Image Processing Toolbox, and then match the existent corners of both ground truth and the map originated from the SLAM algorithm. This method can be divided in three main steps: image preparation, image overlapping and image matching.

Image preparation: The images are loaded in black and white, converted in a 2D matrix, followed by the creation of another image with extracted corners. In the case of the ground truth image, only the edges of the obstacles are important, because in the SLAM map only the contours of the obstacles are mapped. To accurately match the two images, an edge extraction method was performed first. The method, is available in the MATLAB Image Processing Toolbox. After this step, the 2D matrices containing the edges of the obstacles, and the images of the detected corners are available (as shown in Fig. 5.2).

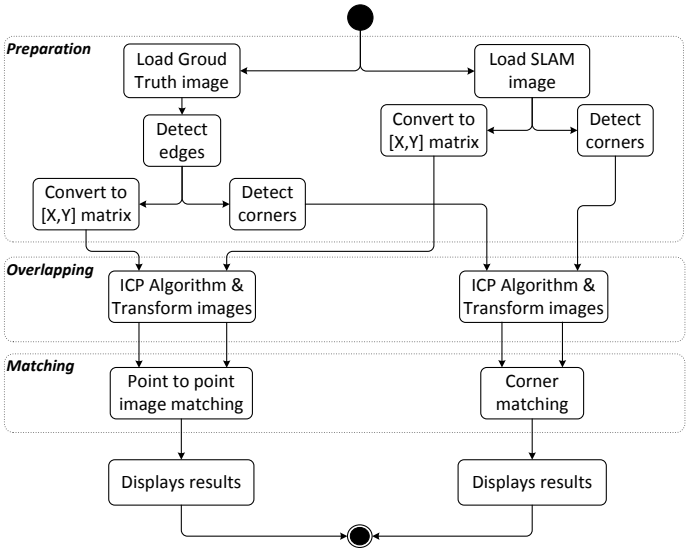


Figure 5.1: Proposed map evaluation method

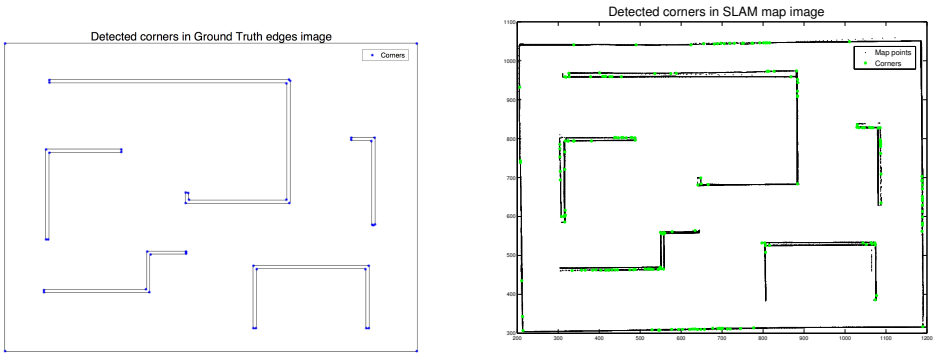


Figure 5.2: Detected corners in the Ground Truth edges image (left) and in the SLAM map (right)

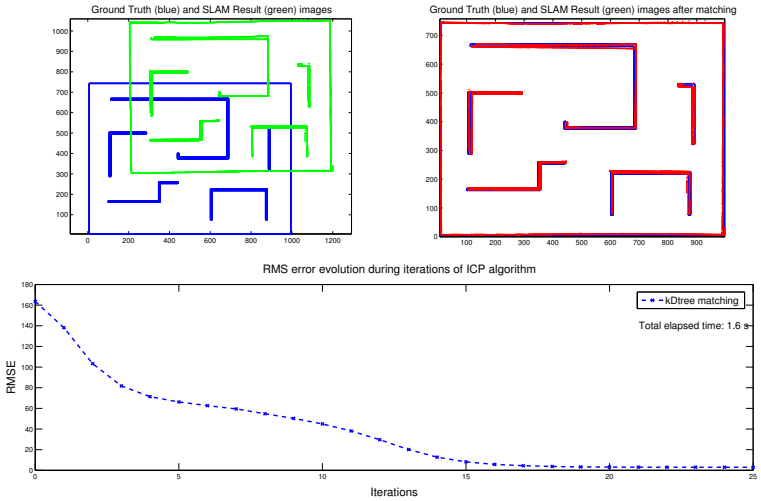


Figure 5.3: Results of the ICP algorithm, before, after and RMSE evolution

Image overlapping: An ICP (Iterative Closest Points) algorithm based on the implementation of [Kjer and Wilm, 2010] for 3D point clouds, was adapted for 2D data and used to match the two images, returning a translation and a rotation matrix that are then applied to the SLAM image. Also it returns the Root Mean Square Error (RMSE) of the obtained fitting (Root Mean Square Error (RMSE) is a frequently used measure of the differences between values predicted by a model or an estimator and the values actually observed). It iterates until it reaches the limit of iterations or the RMSE value stops decreasing, as seen in Fig. 5.3.

This method allows two types of point matching: *kDtree* and brute force. A *kDtree* (or "k-D tree"), for k-dimensional tree, is a data structure used in computer science for organizing some number of points in a space with k dimensions. It is a binary search tree with other constraints imposed on it. K-d trees are very useful for range and nearest neighbor searches, used here. Brute force method tries all possible combinations incrementally, and despite giving the most accurate results, it is impractical, as it takes too long to operate. The SLAM images are then transformed to overlap with the ground truth images (the edge images and the corner points images), resulting in as seen on the top right at Fig. 5.3:

$$New\,Image = R_{rotation} * Old\,Image + T_{translation} \tag{5.1}$$

Image matching: Two distinct matchings are performed, for latter comparison. Point to point image matching and corner matching. In the first, for each point of the SLAM image it verifies if a corresponding point exists in the ground truth image, within a radius that is equal to the RMS error from the image overlapping. The matching points are registered and saved. The result of this process can be observed in the left image of Fig. 5.4, with the red points being the matched ones. In MATLAB, a percentage of the matching is also computed and displayed.

In case of corner matching, for each detected corner in the ground truth image, a corresponding corner is searched in the SLAM image, within a defined radius equal to the RMS error of the ICP method, plus a small threshold. This indicates if the features (in this case corners) of the

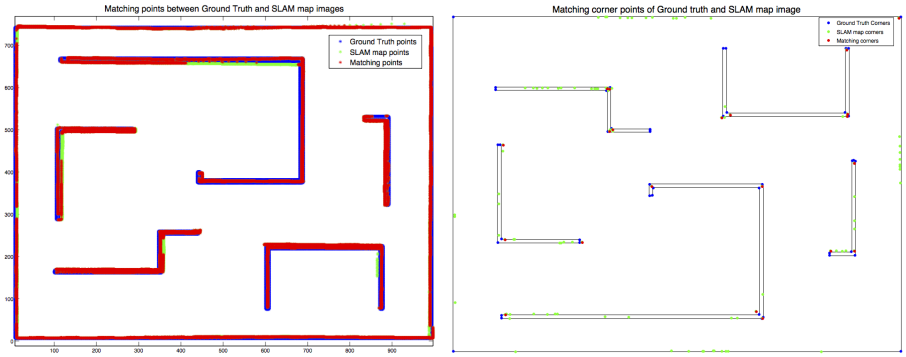


Figure 5.4: Matching points to the left and matching corners to the righ, represented in red

map are present in the SLAM map. However, with a very "noisy" or imperfect SLAM map, the number of false corners can be very high. So, the number of extra corners in the SLAM map that do not correspond to the ground truth, can give a good estimate of how smooth the walls and obstacles are represented in the map. In other words, this corner detection and matching can give a good estimation of how smoothly the map is created by the SLAM method. In the right image of Fig. 5.4, it is possible to observe that many detected corners in the SLAM map do not match with any corner represented in the ground truth map. Those mismatches represent imperfections in the SLAM map. If the number of mismatches (imperfections) is known, then a percentage of "smoothness" of the SLAM map can be calculated. Here a smoothness of 100% represents zero mismatches, meaning that all the mapped obstacles were perfectly represented in the map. However, this value can only be interpreted together with the corner matching value, because not all obstacles need to be mapped for them to be correct.

5.3 Test results

From the SLAM methods highlighted before (*Gmapping*, *Hector SLAM*, *Karto* and *RGBD-SLAM)*, only *Gmapping* and *Karto* are suitable to perform with the RobChair, because *Hector SLAM* relies heavily on inertial sensors, ignoring encoder data, and works best with high frame rate and long range lasers. *RGBD-SLAM* is not suitable as well, because it creates 3D maps, ignoring 2D laser data and it is very computational demanding. So, both *Gmapping* and *Karto* were chosen to be evaluated by the benchmarking process, in a simulated environment.

To ensure that the methods were perform every time under the exact same conditions, a single run was performed on a defined area, and all the data was stored using the *rosbag* tool in ROS. This tool is able to record and play back data exchanged in ROS topics, meaning that it is possible to record all the activity in ROS for latter reproduction. In this case, the information provided from the simulator Gazebo was recorded, upon which the SLAM method will perform.

The results shown in Fig. 5.5 and 5.6, were obtained by running the simulations five times for each method, and then running the evaluation method for each resultant map. These results are organized in three classes: the point to point image matching percentage (see Fig. 5.5), the corner matching percentage (see left chart in Fig. 5.6), and the percentage of the map "smoothness" (see right chart in Fig. 5.6). By analyzing the data, it is possible to conclude that *Gmapping* performed better in all the tests. Also, in Table 5.1 the average results for both SLAM methods are presented. *Karto* had

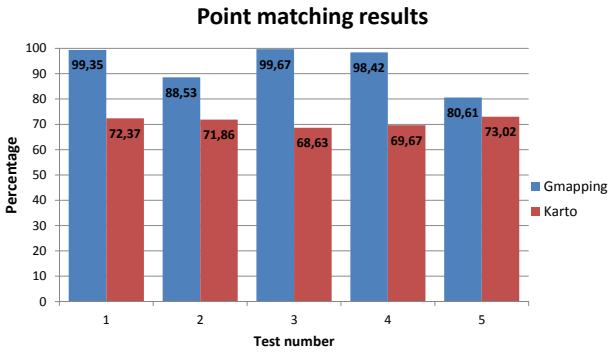


Figure 5.5: Chart with the point to point matching results for Gmapping and Karto

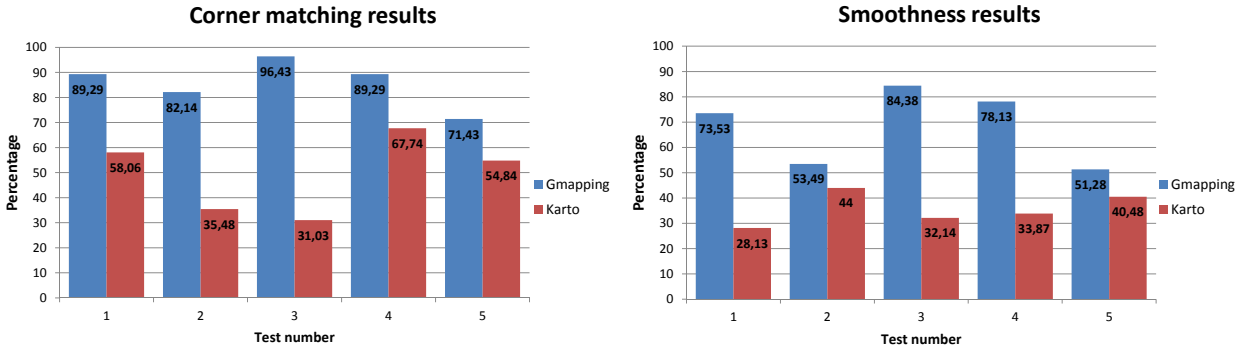


Figure 5.6: Charts with the corner matching and smoothness results for *Gmapping* and *Karto* SLAM methods

significant problems in keeping a smooth map, where in average, only 49,43% of the corners from the original map were found, and only 35,72% of the detected corners in the *Karto* map match the original, indicating that more false positives where detected in comparison to *Gmapping*.

5.4 SLAM with Gmapping

After concluding that *Gmapping* is a better choice for the RobChair, both from the results from Table 5.8 and by discarding other non practical methods, tests on the real platform were made at the ISR ground floor using *Gmapping*, depicted in Fig. 5.7.

The resultant map in Fig. 5.8 is an example of a situation that occurred frequently in more tests. Although details like the room entrance doors and 90^{o} corners are well replied on the map, clearly it is not representative of the real corridor. It is easy to see that the loop was not closed, because in the map, the end point does not correspond to the start point of the RobChair, and that was not true on the real run. These results show that the loop closure ability of *Gmapping* is having trouble identifying loops on big environments, leading to greater map imperfections on the long run. Several reasons are causing this, such as the non existence of a long range laser sensor and mainly the incorrect parameterization of the Gmapping method. Because of that, changes to the default Gmapping need to be performed, along with the introduction of new methods to mitigate the problem.

Table 5.1: Average results from benchmarking tests

	Point matching	Corner matching	Smoothness
Gmapping	93,32%	85,72%	68,16%
Karto	71,11%	49,43%	35,72%

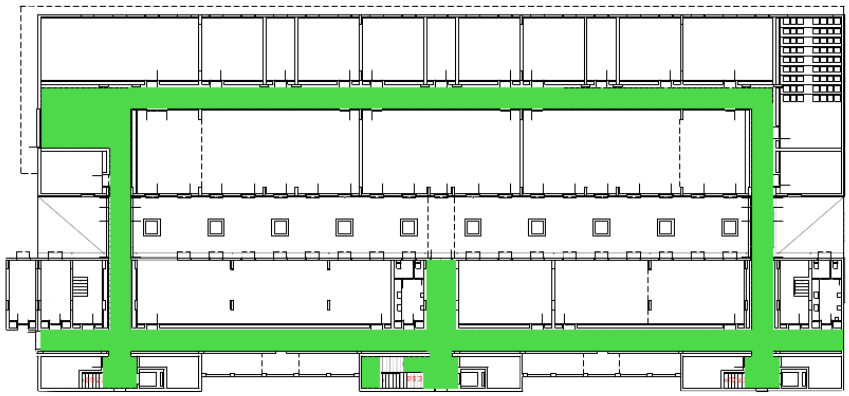


Figure 5.7: Plant of the ISR ground floor, with the corridor indicated in green.

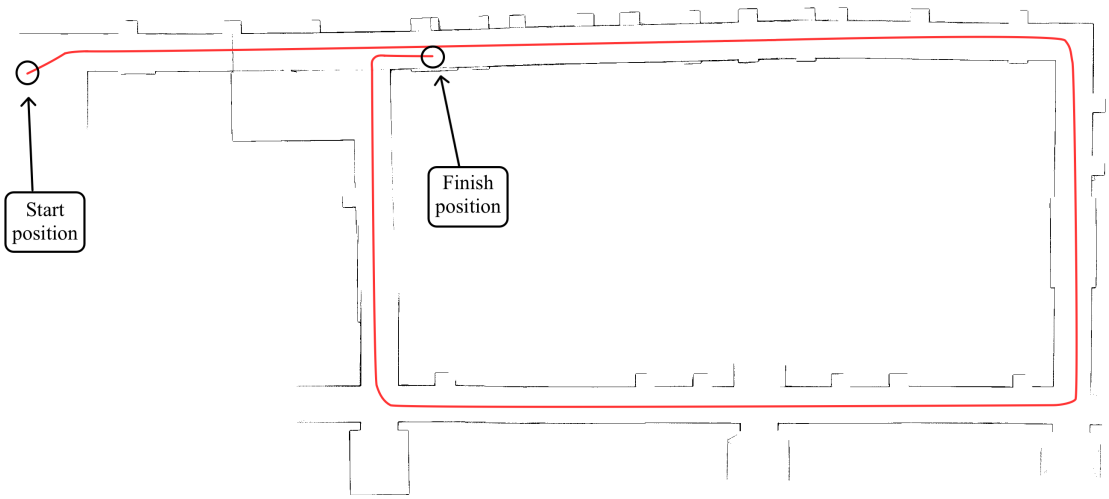


Figure 5.8: Testing *Gmapping* at the ISR corridor.

Chapter 6

Two stage loop closure detection

Following the results of the previous chapter, where loop closure detection failure lead to serious mapping problems, changes to the SLAM algorithm and the introduction of new methods were performed to mitigate these problems. Here, loop closure detection is addressed, making use of modifications to the standard *Gmapping*, and also by applying the FAB-MAP algorithm. A new ROS node is developed to manage both methods. These are further described in this chapter.

6.1 RobChair Gmapping

The loop closure problem happened because the default settings of *Gmapping* were not ready for the RobChair setup and environment particularities. Three internal modules/routines from *Gmapping* might benefit from adjusting their internal parameters. These modules/routines are the "particle resampling", "scan matching" and the motion model prediction.

Particle resampling: This is a routine where lower ranked particles from the particle filter are deleted. This process is constantly evaluating the quality of all particles, by attributing a probability value to each one of them, that represents the probability of the particle representing the true position and motion of the robot. The particle with the highest probability value at time k is the one that is used for the map displayed at time $k + 1$. When some particles drop bellow a threshold value, opening room for new particles to be created from highly ranked ones. Although this is a very important routine to keep the quality of the particles, in certain cases, namely during the mapping of large corridors, good particles can be wrongly eliminated. The particle resampling should occur after a loop closure situation is detected, if possible. Particle resampling should be prevented during the first loop, by lowering the threshold value. Regarding the particle filter, the default number of particles in the particle filter was low (30 particles). By raising the number of particles, it is possible to keep track of a higher number of possible robot locations and map variations, but with an associated increase of the computational cost.

Scan matching: The scan matching procedure observes the laser readings and matches the actual scan with previous scans. If a correspondence is found with a high degree of certainty, given by a standard deviation parameter, the pose of the robot is updated to the previous matched location. This is important for correcting the robot pose when all the map is available, although if incorrectly set, it can lead to large pose estimation errors, and consequently mapping errors. Ideally, when exploring a new part of the environment, scan matching should be kept to a minimum, by decreasing the standard deviation of the matching process and increasing the number of iterations the method performs, while finding a matching sample, to ensure that scan matching only occurs if the method has a big certainty of the matching. But once the environment is mapped, it should be set with higher standard deviation and with a lower number of refinement

steps. This would provide leverage to the method in situations such as odometry errors, laser occlusion and robot kidnapping after a loop is completed. Using a laser sensor with a low range also makes the scan matching process more inaccurate.

For example, in the resultant map shown in Fig. 5.8, the lower corridor is smaller than the upper corridor, because of incorrect pose updates given by the scan matching process, leading to a large final displacement between initial and final positions. This happened because the lower corridors do not have significant distinct features, causing the perception of the laser readings as two straight lines. When moving along the corridor, the sensor readings remain unchanged, and the scan matching process keeps updating the robot pose to the beginning of the corridor (the initial part that was actually mapped), resulting in smaller corridors in the map.

Motion model: *Gmapping* has an embedded motion model of the robot that processes the odometry information and computes a gaussian noise model for the predicted movement. This noise model is important for representing the odometry errors of a robot and take then into account when estimating the robot pose. But, with higher standard deviation values for the model, the uncertainty can rapidly increase, leading to more sparse particles in the particle filter. But in the case of the RobChair, with a calibrated odometry, this standard deviation parameters should be reduced to more accurately model the correct behavior of the RobChair. In sum, this will increase the odometry readings confidence.

Giving all this internal routines of *Gmapping*, it is possible to conclude that the mapping procedure can be divided in two parts, the one where the RobChair is uncovering new terrain, in the case of the ISR corridor this corresponds to the first loop, and a second part, where the RobChair is maintaining and adding detail to the obtained map, after loop closure detection. For each part, a set of adapted configurations to the particle resampling, scan matching and odometry confidence can provide benefits to the SLAM method both in mapping and pose estimation.

Therefore, there are two problems to be addressed in order to perform the right adjustments, being them: reformulate Gmapping to accept internal parameter changes in run time, and having the RobChair to automatically and reliably detect a loop closure. For *Gmapping*, a new version called the *RobChair Gmapping*, was designed to accept changes to the internal routines parameters in run time, using a ROS callback that listens to a ROS message topic, receiving new settings when required. Upon start, the new version of Gmapping loads the initial configuration settings from a launch file, that are adjusted for generating the map with the RobChair before achieving loop closure (see Fig. 6.1).

6.2 Loop closure detection

For the problem of the loop closure detection, the openFABMAP method ([Glover et al., 2012a]) described in chapter 2, that makes use of the bag of words technique, was applied in ROS. This method requires an a priori training to be performed in an different environment from where the robot is usually found, but with similar types of features to be detected, in this case an indoor location. This dataset was performed on a distinct corridor, located in the Department of Electrical Engineering of University of Coimbra. The resultant dataset is then used to compute appearance matches during an online procedure, and detect cases such as the one illustrated in Fig. 2.12 for an outdoor environment,

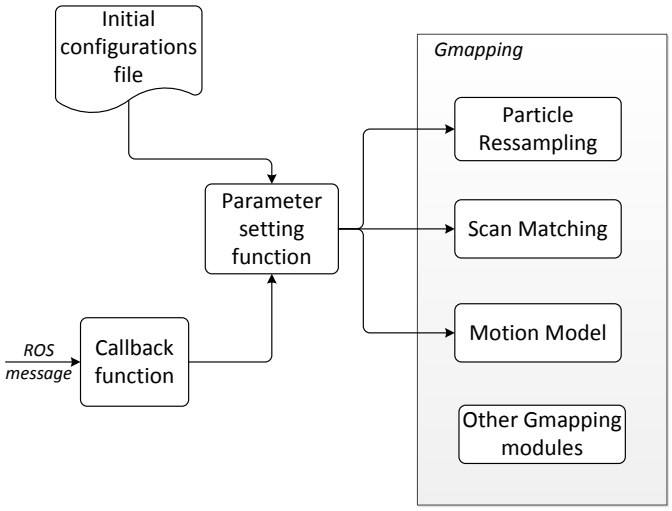


Figure 6.1: New parameter setting method for some routines of *Gmapping*.

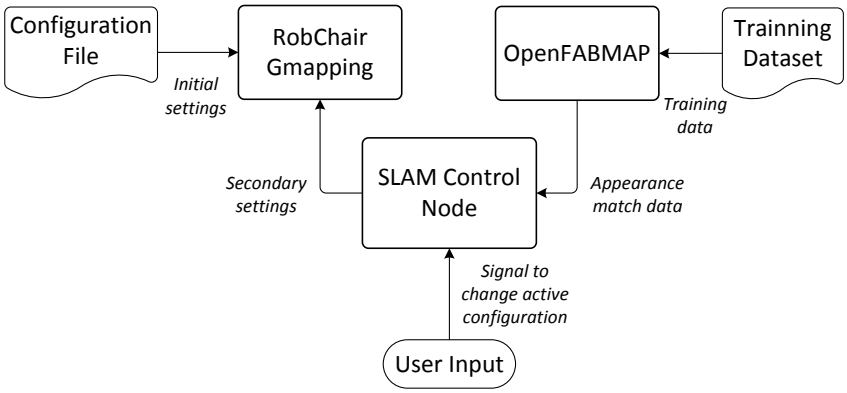


Figure 6.2: Interaction between nodes for dynamic adjustment of Gmapping.

where the images from the places indicated with circles are actually from the same location in the world, meaning that a loop closure was detected.

The method publishes the appearance match data to a ROS message topic. This message carries the actual image index, an older image index and the probability of matching between the two. The method is configured to output match data only for high probability loop closure situations, in this case, it only sends a message if the probability of matching is above 98%. This means that openFABMAP has more than 98% confidence that the robot has already been in that place, indicating a possible loop closure situation.

With this information given in real time, it is possible to inform the *RobChair Gmapping* of the loop closure detection, in order for the loop closure to be performed by the internal particle filter of *Gmapping*.

6.3 Implementation

Given the loop closure detection information from the openFABMAP algorithm, a SLAM control node was developed to set the RobChair Gmapping with the new settings, as depicted in Fig. 6.2.

Table 6.1: The parameters that change in *RobChair Gmapping* for the three scenarios.

	Initial settings	Secondary settings	Default settings
Scan matching standard deviation (σ)	0,01	0,1	0,05
Scan matching routine iterations	7	4	5
Linear noise standard deviation (srr)	0,01	0,075	0,1
Linear noise standard deviation (stt)	0,02	0,1	0,2
Number of particles	50	50	30
Resampling threshold probability	0,025	0,25	0,5
Linear global map and pose update interval (meters)	0,6	0,8	1
Angular global map and pose update interval (radians)	0,3	0,5	0,5
Temporal global map and pose update interval (seconds)	2,0	2,0	1,0

This control node receives data from two sources, one from the openFABMAP and another directly from the user (as depicted in Fig. 6.2). The node has internally the necessary parameters for the three *RobChair Gmapping* scenarios: standard Gmapping settings, initial RobChair Gmapping settings (same as the ones present in the initial configuration file), and the after loop closure detection settings. The user can change between these three settings at any time.

If a message of appearance matching data is received, it further evaluates if the RobChair is in a loop closure situation, by analyzing the indexes of the matched images. If the indexes are numerically too close to each other, this means that openFABMAP is detecting similarity between image frames from the same location, meaning that it is still in the same place, either because the robot is moving slowly or because the local area has no distinct features. If the method detects that the robot has been already there, it sets the "after loop closure" settings, that will allow particle resampling to occur, by setting the scan matching routine with a higher standard deviation and with a lower number of refinement steps. Also, the motion model is set with higher standard deviation values.

With this changes, the *RobChair Gmapping* is now able to close the loop by matching the beginning and end parts of the loop, correcting the map in the process, and performing particle resampling. Also, this will allow for *RobChair Gmapping* to better maintaing and improve the map after the loop closure.

In Table 6.1 the parameters that change in *RobChair Gmapping* for the three scenarios are listed.

6.4 Experimental results

With the new SLAM method properly implemented, tests were performed directly on the RobChair, in the ISR ground floor corridor. The image data required by the openFABMAP algorithm was obtained by extracting the RGB image data from a Kinect sensor mounted on the RobChair. In Fig. 6.3 it is possible to observe in the first two map versions that a pose estimation error is also present, because once again the starting point of the loop does not match exactly with the ending point. Although, with the new settings in the *RobChair Gmapping*, the observed error is significantly lower than results with the standard *Gmapping*. The last map version of Fig. 6.3, shows the map after the openFABMAP method detected that the RobChair was there before (a loop closure possibility), changing the settings in *RobChair Gmapping* and giving it the right parameters for detecting and closing the loop. In the loop closure process, the end parts of the map are joined and the robot pose updated. Some overlapping can occur, because even the highest ranked particle in SLAM algorithm that was chosen to close the

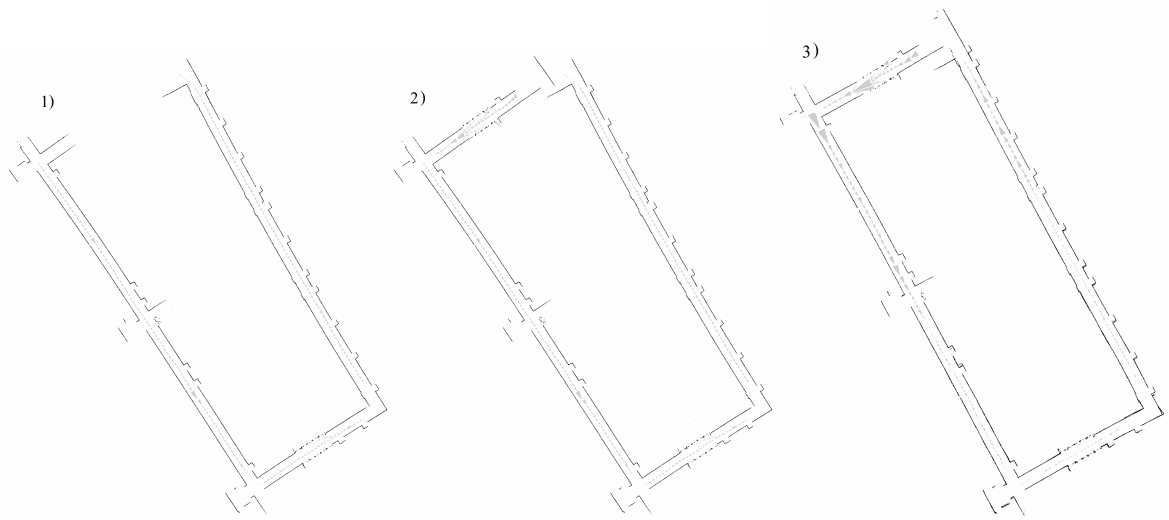


Figure 6.3: Example of successful corridor mapping with loop closure detection, in a single run.

loop (as previously demonstrated in Fig. 2.10) may not represent the exact pose of the robot in world coordinates.

A second example of an obtained result is shown in Fig. 6.4, where laser occlusion was performed (the laser was obstructed for brief periods of time). In this result, after a loop closure possibility had been detected, the SLAM method took more iterations to match the new readings with the old ones, and in image *2)* of Fig. 6.4, the same corridor starts to be mapped again on a different location. But eventually, the scan match detected that new part on the previous map, and corrected the robot pose, and consequently the map, in image *3)*. In image *4)*, the RobChair continued to perform a second loop on the same corridor, showing that parts of the map that are missing or not detailed enough improve on further passages of the RobChair. This is also a positive result towards the long term stability of the method when the RobChair goes through the same environment for a long period of time.

In these results, also the problem of long featureless corridors described in the beginning was addressed, because the RobChair Gmapping generates lower uncertainty on the motion model and the scan matching process does not have enough confidence to correct its pose in the first loop, being able to map the entire featureless part of the corridors.

However, this new implementation have disadvantages as well, such as the problem of robot kidnapping[1] before the loop closure is detected, while it is performing the initial mapping. Also, at the present, the method does not detect when the RobChair enters another loop, meaning that and the configurations do not change back to the initial ones automatically.

[1]Moving the robot to another place in the environment without giving it any indication of the performed dislocation or new the pose

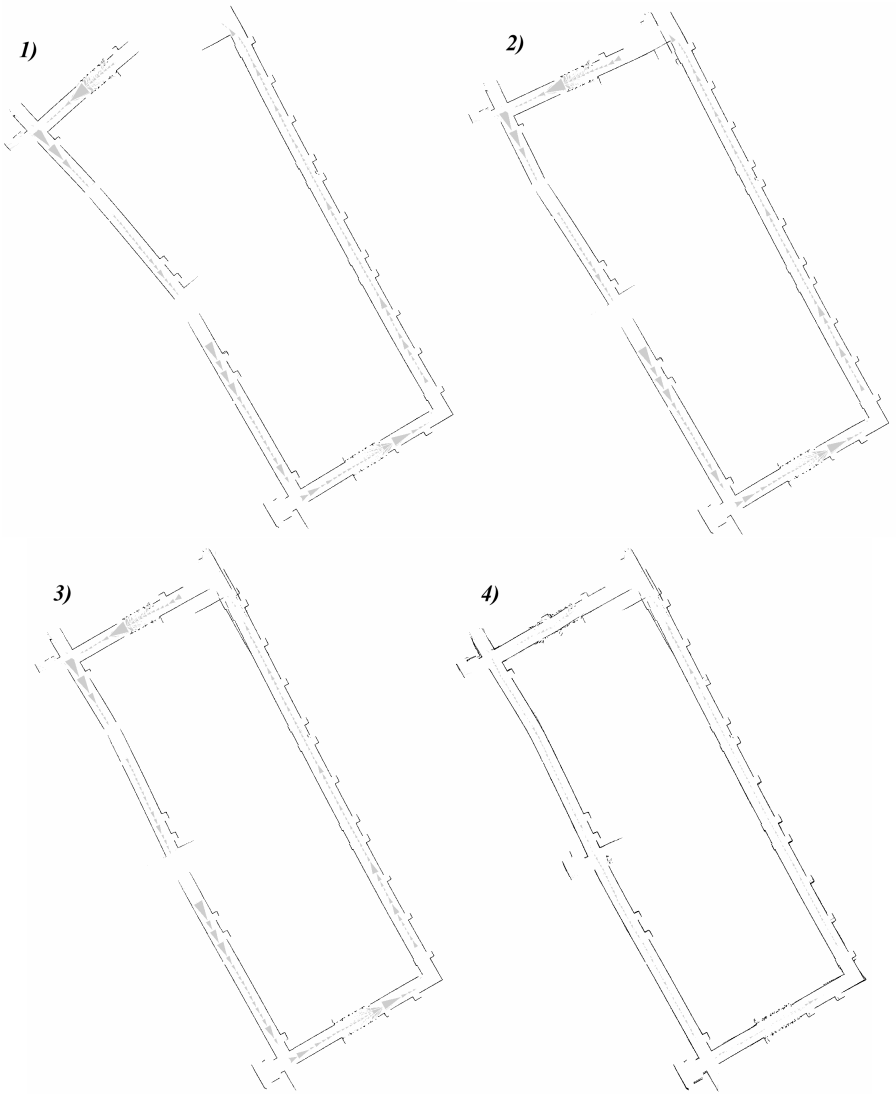


Figure 6.4: Example of successful loop closure detection and map improvement, in a single run.

Chapter 7

Conclusion and future work

7.1 Conclusion

In this dissertation a study of the most relevant SLAM techniques were performed and presented together with some loop closure detection methods that were integrated to achieve a better SLAM solution.

The goal of upgrading the RobChair to a simpler and up to date platform was successfully achieved, proving to be reliable during all the performed tests carried out during the execution of the work reported in this dissertation. The new integrated software is capable of providing an easy and fast way of connecting new devices and testing diversified algorithms, making it an ideal setup for researchers. Also, ROS proved to be a very useful tool for developing new methods for the RobChair, as well as testing them in a simulated environment. These tools allowed for the research and implementation of a reliable SLAM technique.

With the future goal of researching navigation techniques for a semi-autonomous robotic wheelchair, SLAM techniques that are able to generate maps and compute the pose of the robotic wheelchair were implemented. For that purpose, several SLAM techniques were evaluated, and some of them were benchmarked, in a new proposed method for evaluating 2D grid maps from SLAM algorithms. A final SLAM technique was researched and implemented by combining the information of an adjusted SLAM method and a visual appearance method, that together solved the loop closure problem, achieving the goal of having a robust SLAM algorithm.

7.2 Future work

To continue the current work on the RobChair platform, the SLAM techniques must be tested in oder scenarios. Further tests concerning long life support, environment changes and robot kidnapping should be performed to the new SLAM method, to better test the implementation and improving the necessary aspects of the solution accordingly.

The main task to be performed next is to provide the RobChair with semi-autonomous capabilities, integrating a navigation system on the current RobChair solution. Problems such as local and global planning and obstacle avoidance should be addressed. Also, new human machine interface systems, such as the Brain Computer Interface, must be integrated in the platform.

Bibliography

[Angeli et al., 2008] Angeli, A., Filliat, D., Doncieux, S., and Meyer, J.-A. (2008). A fast and incremental method for loop-closure detection using bags of visual words. *IEEE Transactions on Robotics, Special Issue in Visual SLAM*.

[Araki, 2003] Araki, M. (2003). Control systems, robotics and automation - pid control. Kyoto University, Japan.

[Balaguer et al., 2007] Balaguer, B., Carpin, S., and Balakirsky, S. (2007). Towards quantitative comparisons of robot algorithms: Experiences with slam in simulation and real world systems. *IROS 2007 Workshop*.

[Bay et al., 2006] Bay, H., Tuytelaars, T., , and Gool, L. V. (2006). Surf: Speeded up robust features. *In 9th Europ. Conf. on Computer Vision*.

[Bonarini et al., 2012] Bonarini, A., Ceriani, S., Fontana, G., and Matteucci, M. (2012). Introducing lurch: a shared autonomy robotic wheelchair with multimodal interfaces. *IROS 2012 Workshop on Progress, challenges and future perspectives in navigation and manipulation assistance for robotic wheelchairs*.

[Borenstein, 1994] Borenstein, J. (1994). Internal correction of dead-reckoning errors with the smart encoder trailer. *International Conference on Intelligent Robots and Systems*.

[Borenstein and Feng, 1996] Borenstein, J. and Feng, L. (1996). Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on Robotics and Automation*, 12(5).

[Callmer et al., 2008] Callmer, J., Granstrom, K., Nieto, J., and Ramos, F. (2008). Tree of words for visual loop closure detection in urban slam.

[CARMEN-Team, 2000] CARMEN-Team. Carmen robot navigation toolkit [online]. (2000). Available from: `http://carmen.sourceforge.net/intro.html`.

[Chow and Liu, 1968] Chow, C. K. and Liu, C. N. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*.

[Coleman, 2013] Coleman, D. T. Ros concepts [online]. (2013) [cited 2013-06-27]. Available from: `http://ros.org/wiki/ROS/Concepts`.

[Computer, 1999] Computer, K. P. M. (1999). Bayesian map learning in dynamic environments. *Neural Info. Proc. Systems (NIPS)*.

[Cummins and Newman, 2007] Cummins, M. and Newman, P. (2007). Probabilistic appearance based navigation and loop closing. *Robotics and Automation, 2007 IEEE International Conference*, pages 2042–2048.

[Dudek and Jenkin, 2000] Dudek, G. and Jenkin, M. (2000). Computational principles of mobile robotics. *Second Edition, Cambridge University Press*.

[Endres et al., 2013] Endres, F., Hess, J., Engelhard, N., Sturm, J., and Burgard, W. Rgbdslam - 6dof slam for kinect-style cameras [online]. (2013). Available from: `http://openslam.org/rgbdslam.html`.

[Endres et al., 2012] Endres, F., Hess, J., Engelhard, N., Sturm, J., Cremers, D., and Burgard, W. (2012). An evaluation of the rgb-d slam system. *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*.

[Engelhard et al., 2011] Engelhard, N., Endres, F., Hess, J., Sturm, J., and Burgard, W. (2011). Real-time 3d visual slam with a hand-held rgb-d camera. *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum*.

[Escobedo et al., 2012] Escobedo, A., Rios-Martinez, J., Spalanzani, A., and Laugier, C. (2012). Context-based face control of a robotic wheelchair. *IROS 2012 Workshop on Progress, challenges and future perspectives in navigation and manipulation assistance for robotic wheelchairs*.

[Filliat, 2009] Filliat, D. (2009). Robotics and computer vision - loop closure detection. Available from: `http://cogrob.ensta-paristech.fr/loopclosure.html`.

[Folkesson and Christensen, 2007] Folkesson, J. and Christensen, H. I. (2007). Closing the loop with graphical slam. *IEEE Transactions on Robotics*, 23(4).

[Garrote, 2013] Garrote, L. C. A. S. (2013). *ISRobot 2.0 - Controlo Por Computador*.

[Gerkey, 2013] Gerkey, B. Gmapping - ros package summary [online]. (2013). Available from: `http://www.ros.org/wiki/gmapping`.

[Glover et al., 2012a] Glover, A., Maddern, W., Warren, M., tephanie Reid, Milford, M., and Wyeth, G. (2012a). Openfabmap: An open source toolbox for appearance-based loop closure detection. *International Conference on Robotics and Automation*, 14-18.

[Glover et al., 2012b] Glover, A., Maddern, W., Warren, M., tephanie Reid, Milford, M., and Wyeth, G. Openfabmap documentation webpage [online]. (2012). Available from: `http://docs.opencv.org/trunk/modules/contrib/doc/openfabmap.html`.

[Google, 2013] Google. Google sketchup 3d modeling software [online]. (2013) [cited 2013]. Available from: `http://www.sketchup.com`.

[Grasse et al., 2010] Grasse, R., Morere, Y., and Pruski, A. (2010). Assisted navigation for persons with reduced mobility: path recognition through particle filtering (condensation algorithm). *Journal of Intelligent and Robotic Systems*, (60):19–57.

[Grisetti et al., 2005] Grisetti, G., Stachniss, C., and Burgard, W. (2005). Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. pages 2432–2437.

[Grisetti et al., 2006] Grisetti, G., Stachniss, C., and Burgard, W. (2006). Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 32:16–25.

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

[Grisetti et al., 2013] Grisetti, G., Stachniss, C., and Burgard, W. Gmapping - openslam [online]. (2013). Available from: `http://www.openslam.org/gmapping.html`.

[Ho and Newman, 2006] Ho, K. and Newman, P. (2006). Combining visual and spatial appearance for loop closure detection in slam. *Robotics and Autonomous Systems - RaS*, 54(9):740–749.

[Iturrate et al., 2009] Iturrate, I., Antelis, J., Kubler, A., and Minguez, J. (2009). A noninvasive brain-actuated wheelchair based on a p300 neurophysiological protocol and automated navigation. *IEEE Transactions on Robotics*.

[KARTO-Team, 2013] KARTO-Team. Karto - software for robots on the move [online]. (2013). Available from: `http://www.kartorobotics.com`.

[Kjer and Wilm, 2010] Kjer, M. and Wilm, J. Iterative closest point algorithm on three dimensional point [online]. (2010). Available from: `http://www.mathworks.com/matlabcentral/fileexchange/27804-iterative-closest-point`.

[Kohlbrecher et al., 2012] Kohlbrecher, S., Meyer, J., Petersen, K., and Graber, T. (2012). Hector slam for robust mapping in usar environments. *ROS RoboCup Rescue Summer School Graz 2012*. Available from: `http://tedusar.eu/cms/sites/tedusar.eu.cms/files/Hector_SLAM_USAR_Kohlbrecher_RRSS_Graz_2012.pdf`.

[Kummerle et al., 2009] Kummerle, R., Steder, B., Dornhege, C., Ruhnke, M., Grisetti, G., Stachniss, C., and Kleiner, A. (2009). On measuring the accuracy of slam algorithms. *Autonomous Robots*.

[Larsen et al., 1998] Larsen, T., Bak, M., Andersen, N., and Ravn, O. (1998). Location estimation for autonomously guided vehicle using an augmented kalman filter to autocalibrate the odometry. *FUSION98 Spie Conference Las Vegas*.

[Liu and Zhang, 2012] Liu, Y. and Zhang, H. (2012). Indexing visual features: Real-time loop closure detection using a tree structure. *IEEE International Conference on Robotics and Automation*, (14-18).

[Lopes et al., 2012] Lopes, A., Pires, G., and Nunes, U. (2012). Robchair: Experiments evaluating brain-computer interface to steer a semi-autonomous wheelchair. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS12)*.

[Lopes et al., 2013a] Lopes, A., Pires, G., and Nunes, U. (2013a). Assisted navigation for a brain-actuated intelligent wheelchair. *Robotics and Autonomous Systems*.

[Lopes et al., 2011] Lopes, A., Pires, G., Vaz, L., and Nunes, U. (2011). Wheelchair navigation assisted by human-machine shared-control and a p300-based bci. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS11)*.

[Lopes et al., 2007] Lopes, A. C., Moita, F., Nunes, U., and Solea, R. (2007). An outdoor guidepath navigation system for amrs based on robust detection of magnetic markers. *12th IEEE Conference on Emerging Technologies and Factory Automation*.

[Lopes et al., 2013b] Lopes, A. C., Pires, G., and Nunes, U. (2013b). Assisted navigation for a brain-actuated intelligent wheelchair. *International Journal of Robotics and Autonomous Systems*.

[Maddern et al., 2012] Maddern, W., Milford, M., and Wyeth, G. (2012). Cat-slam: Probabilistic localisation and mapping using a continuous appearance-based trajectory. *I. J. Robotic Res*, 31(4):429–451.

[Meckstroth, 2009] Meckstroth, M. (2009). Mobile robotics: Moving robots forward. Technical report, RTC Magazine. Available from: `http://www.rtcmagazine.com/articles/view/101197`.

[Microsoft, 2012] Microsoft. Microsoft robotics developer studio (rds) [online]. (2012). Available from: `http://www.microsoft.com/robotics/#Product`.

[Minorsky, 1922] Minorsky, N. (1922). Directional stability of automatically steered bodies. *Journal of the American Society for Naval Engineers*, 34(2):280–309.

[Montemerlo et al., 2002] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). Fastslam: A factored solution to the simultaneous localization and mapping problem. *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, pages 593–598.

[MOOS-Team, 2013] MOOS-Team. Mission oriented operating suite - robotics framework [online]. (2013). Available from: `http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php/Main/Introduction`.

[Mughal, 2004] Mughal, A. M. (2004). Kalman filter and extended kalman filter. Technical report, University of Arkansas.

[NASA, 2011] NASA. Nasa - what is robotics? [online]. (2011). Available from: `http://www.nasa.gov/audience/foreducators/robotics/home/what_is_robotics_k4.html`.

[Newman and Ho, 2005] Newman, P. and Ho, K. (2005). Slam - loop closing with visually salient features.

[Nieto et al., 2007] Nieto, J., Bailey, T., and Nebot, E. (2007). Recursive scan-matching slam. *Robotica and Autonomous Systems*, 55:39–49.

[Nistér and Stewénius, 2006] Nistér, D. and Stewénius, H. (2006). Scalable recognition with a vocabulary tree. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2:2161–2168.

[Orca-Team, 2009] Orca-Team. Orca: Components for robotics [online]. (2009). Available from: `http://orca-robotics.sourceforge.net/index.html`.

[OROCOS-Team, 2007] OROCOS-Team. The orocos project - open robot control software [online]. (2007). Available from: `http://people.mech.kuleuven.be/~orocos/pub/stable/documentation/rtt/current/doc-xml/orocos-overview.pdf`.

[Oxford, 2013] Oxford. Oxford dictionary - robotics defenition [online]. (2013). Available from: `http://oxforddictionaries.com/definition/english/robotics`.

[Park et al., 2012] Park, J., Johnson, C., and Kuipers, B. (2012). Robot navigation with model predictive equilibrium point control. *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*.

[Patel et al., 2012] Patel, M., Miro, J. V., and Dissanayake, G. (2012). Probabilistic activity models to support activities of daily living for wheelchair users. *IROS 2012 Workshop on Progress, challenges and future perspectives in navigation and manipulation assistance for robotic wheelchairs*.

[Pires and Nunes, 2002] Pires, G. and Nunes, U. (2002). A wheelchair steered through voice commands and assisted by a reactive fuzzy-logic controller. *Journal of Intelligent and Robotic Systems*, 34:301–314.

[Player-Team, 2010] Player-Team. The player project - free software tools for robot and sensor applications [online]. (2010). Available from: `http://playerstage.sourceforge.net/`.

[Quigley et al., 2009] Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. (2009). Ros: an open-source robot operating system. *ICRA Workshop on Open Source Software*.

[Rekleitis, 2003] Rekleitis, I. M. (2003). A particle filter tutorial for mobile robot localization. Master's thesis, Centre for Inteligent Machines, McGill University, Canada.

[R.I.A., 1979] R.I.A. Robot institute of america - definition of a robot. [online]. (1979). Available from: `http://www.cs.cmu.edu/~chuck/robotpg/robofaq/1.html`.

[Rios-Martinez et al., 2011] Rios-Martinez, J., Spalanzani, A., and Laugier, C. (2011). Understanding human interaction for probabilistic autonomous navigation using risk-rrt approach. *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

[RoboteQ, 2012] RoboteQ (2012). *RoboteQ advanced digital motor controllers*. RoboteQ.

[Robotics-Community, 2013] Robotics-Community. Open source robotics foundation [online]. (2013) [cited 2013]. Available from: `http://osrfoundation.org`.

[ROS-Team, 2013] ROS-Team. Ros - robot operating system [online]. (2013). Available from: `http://www.ros.org/wiki/ROS/Introduction`.

[Siegwart and Nourbakhsh, 2004] Siegwart, R. and Nourbakhsh, I. R. (2004). *Introduction to Autonomous Mobile Robots*, volume 169. The MIT Press. Available from: `http://www.amazon.de/Introduction-Autonomous-Mobile-Intelligent-Robotics/dp/0262015358`.

[Sivic and Zisserman, 2003] Sivic, J. and Zisserman, A. (2003). A text retrieval approach to object matching in videos. *Proceedings of the Int. Conf. on Computer Vision*, 2:1470–1477.

[Stachniss et al., 2005] Stachniss, C., Hähnel, D., Burgard, W., and Grisetti, G. (2005). On actively closing loops in grid-based fastslam. *ADVANCED ROBOTICS*.

[Sturm et al., 2012] Sturm, J., Engelhard, N., Endres, F., Burgard, W., and Cremers, D. (2012). A benchmark for the evaluation of rgb-d slam systems. *IROS12*.

[Surmann et al., 2004] Surmann, H., Nuchter, A., Lingemann, K., and Hertzberg, J. (2004). 6d slam — preliminary report on closing the loop in six dimensions.

[Thrun, 1998] Thrun, S. (1998). Learning maps for indoor mobile robot navigation. *Artificial Intelligence*, 99:21–79.

[Thrun et al., 2005] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. MIT Press.

[Thrun et al., 2001] Thrun, S., Fox, D., Bugard, W., and Dellaert, F. (2001). Robust monte carlo localization for mobile robots. *Artificial Inteligence*, 128:99–141.

[Thrun and Montemerlo, 2005] Thrun, S. and Montemerlo, M. (2005). The graph slam algorithm with applications to large-scale mapping of urban structures. *International Journal on Robotics Research*, 25(5/6):403–430.

[van der Merwe et al., 2001] van der Merwe, R., de Freitas, N., Doucet, A., and Wan, E. (2001). The unscented particle filter. *Neural Info. Proc. Systems (NIPS)*.

[Vanhooydonck et al., 2010] Vanhooydonck, D., Demeester, E., Hantemann, A., Philips, J., Vanacker, G., Brussel, H. V., and Nuttin, M. (2010). Adaptable navigational assistance for intelligent wheelchairs by means of an implicit personalized user model. *Robotics and Autonomous Systems*, 58(8):963–977.

[WillowGarage-Team, 2013] WillowGarage-Team. Willow garage home page [online]. (2013). Available from: `https://www.willowgarage.com/pages/about-us`.

[YARP-Team, 2013] YARP-Team. Yarp - yet another robot platform [online]. (2013). Available from: `http://wiki.icub.org/yarpdoc/what_is_yarp.html`.

[Zhang, 2011] Zhang, H. (2011). Borf: Loop-closure detection with scale invariant visual features. *International Conference on Robotics and Automation*, pages 9–13.

[Zhiwei et al., 2012] Zhiwei, L., Xiang, G., Yanyan, C., and Songhao, Z. (2012). A novel loop closure detection method in monocular slam. *Intel Serv Robotics (2013)*, 6.

Appendix

Appendix A

Tree of Words

A class of loop closure techniques explored by many researchers [Cummins and Newman, 2007, Zhiwei et al., 2012, Nistér and Stewénius, 2006, Callmer et al., 2008, Zhang, 2011, Angeli et al., 2008, Ho and Newman, 2006, Glover et al., 2012a] integrates a method called Bag of Words (BoW), or the similar, Tree of Words (ToW).

Tree of Words was first proposed by Nistér and Stewenius [Nistér and Stewénius, 2006] as a way of finding the closest match of an image in a large database. For loop closing detection, Tree of Words was introduced as a hierarchical approach to Bag of Words [Sivic and Zisserman, 2003], which suggests how an image can be represented by predefined features for fast database query. This predefined set of features is found in a codebook that is generated by clustering a large amount of features, extracted from a training dataset, to form a finite list (commonly thousands) of 'general' appearances often encountered in the environment [Glover et al., 2012a].

The differences are that Bag of Words compares images by matching clusters of words from a small vocabulary, i.e. set of predefined words. Tree of Words on the other hand uses a larger vocabulary and no clustering. The latter showed promising results and had a significant computational speed improvement, making it appropriate for loop closure detection in a dense urban environment [Callmer et al., 2008].

More clearly, in order to classify an image using ToW, feature descriptors are first extracted from the image using for example the feature extractor SURF [Bay et al., 2006], SIFT or other. Each descriptor is then compared to a large number of predefined descriptor vectors, called words, using a hierarchical tree search to find its nearest match. If descriptor α is classified as word m, the image is said to contain word m, no matter exactly how well α and m matches. The image is thus compressed into a list of the words it contains. This list can be readily stored and compared to a database of classified images.

There are also several implementations for indoor navigation using the Tree of Words and Bags of Words methods to describe images in monocular vision based loop closure detection, for example, in [Filliat, 2009] they developed a vision-based loop closure detection algorithm that relies on Bayesian filtering for loop closure probability computation, with images encoded according to the incremental bags of visual words scheme. The overall complexity of the designed solution scales linearly with the number of places, making it possible to detect loop closures in real-time conditions. A similar work by [Zhiwei et al., 2012] also uses a bag of visual words approach for building an appearance-based scene model to deal with the loop closure detection problem of monocular SLAM for mobile robots, as seen in Fig. A.1.

Each image can then be represented by a vector of weighted vectors and a Bayesian filter algorithm is applied to update the detection probability and an inverse image retrieval method is employed to eliminate the wrong loop closure results. Figure A.2 illustrates how their process works.

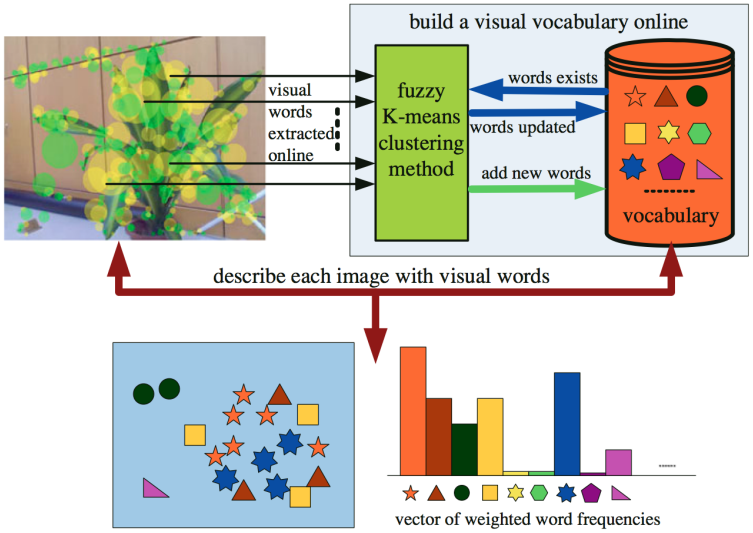


Figure A.1: Image appearance-based modeling process from [Zhiwei et al., 2012]

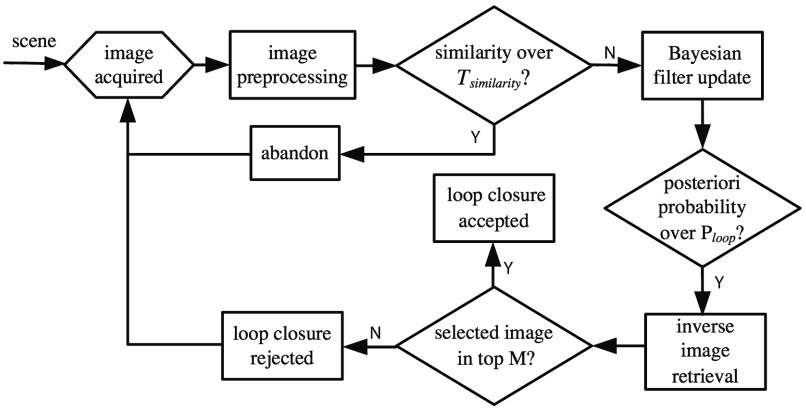


Figure A.2: Loop closure detection process from [Zhiwei et al., 2012]

Appendix B

PID controller

This controller, has three fundamental elements, the proportional part, referred as the P element, the integral part, referred as the I element and the derivative part, referred as the D element.

As seen in Fig. B.1 the PID controller can be described as a transfer function that receives the signal error $E(s)$ and generates an output signal according to its internal parameters, the proportional gain (K_P), the integral gain (K_I), and the derivative gain (K_D).

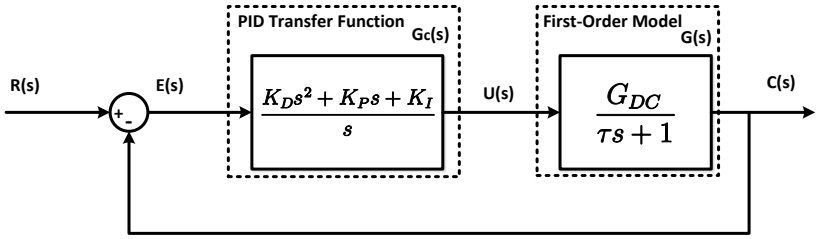


Figure B.1: PID control scheme

The three elements of the PID controller produce outputs with the following nature [Araki, 2003]:

- P element: Proportional to the error at instant t, which is the "present" error.
- I element: Proportional to the integral of the error up to the instant t, which can be interpreted as the accumulations of the "past" error.
- D element: Proportional to the derivate of the error at the instant t, which can be interpreted as the prediction of the "future" error.

To obtain the first a model of a system ($G(s)$), the first order transfer function that models the motor response is:

$$G_S(s) = \frac{G_{DC}}{\tau s + 1} \tag{B.1}$$

with G_{DC} being the DC gain:

$$G_{DC} = \frac{Stacionary\,Response}{Input\,Command} \tag{B.2}$$

and τ_S the rise time, equal to:

$$\tau_S = 0.632(Rise\,Time) \tag{B.3}$$

To obtain the PID transfer function ($G_s(s)$), the following PID controller equation applies:

$$u(t) = K_P e(t) + K_I \int_0^t e(t)dt + K_D \frac{de(t)}{dt} \tag{B.4}$$

Applying the Laplace Transform to equation (B.4), the transfer function is obtained:

$$G_c(s) = \frac{K_D s^2 + K_P s + K_I}{s} \tag{B.5}$$

The closed loop transfer function of the system presented in Fig. B.1 is given by:

$$C(s) = G(s)E(s) \tag{B.6}$$

$$E(s) = R(s) - B(s) \tag{B.7}$$

$$E(s) = R(s) - H(s)C(s) \tag{B.8}$$

$$C(s) = G(s)[R(s) - H(s)C(s)] \tag{B.9}$$

$$\frac{C(s)}{R(s)} = \frac{G(s)}{1 + G(s)H(s)} \tag{B.10}$$

where $R(s)$ is the reference input signal, $C(s)$ is the output, $G(s)$ is the system transfer function, $H(s)$ is the feedback element, $B(s)$ is the feedback signal and $E(s)$ is the error between the reference input signal and the feedback signal. The controller tends to eliminate the $E(s)$ signal over time.

Further, to obtain the required PID gains, the closed loop poles of the system must be obtained:

$$1 + G(s)H(s) = 0 \tag{B.11}$$

with,

$$G(s) = G_c(s)G_S(s) \tag{B.12}$$

$$1 + H(s)G_c(s)G_S(s) = 0 \tag{B.13}$$

Replacing (B.5) and (B.1) in (B.13) with $H(s) = 1$,

$$1 + \frac{K_D s^2 + K_P s + K_I}{s} \frac{G_{DC}}{s\tau + 1} = 0 \tag{B.14}$$

And simplifying,

$$(G_{DC}K_D + \tau)s^2 + (G_{DC}K_P + 1)s + G_{DC}K_I = 0 \tag{B.15}$$

Now it is possible to match (B.15) to a second order system:

$$s^2 + 2\zeta\omega_n s + \omega_n^2 = 0 \tag{B.16}$$

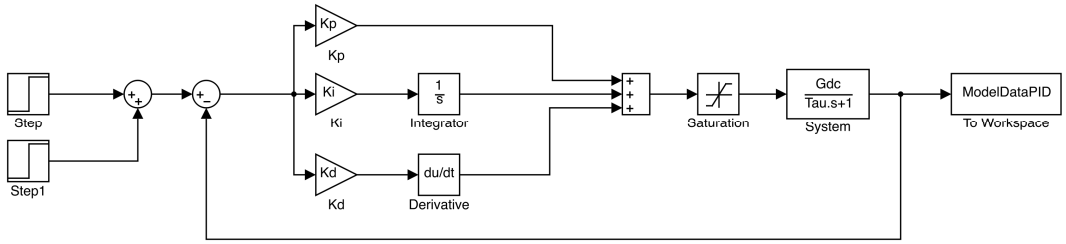


Figure B.2: Simulated PID controller applied to first order model of RobChair motor

where ζ (damping) determines the response shape and ω_n the natural frequency that represent the speed of the response. By matching (B.15) and (B.16), the gains K_P, K_I and K_D are determined:

$$K_P = \frac{2\zeta\omega_n - 1}{G_{DC}} \tag{B.17}$$

$$K_I = \frac{\omega_n^2}{G_{DC}} \tag{B.18}$$

$$K_D = \frac{1 - \tau}{G_{DC}} \tag{B.19}$$

By adding the gain values and the first order function in the system block of the Simulink scheme of Fig. B.2, it is possible to obtain the simulated system response.

Appendix C

The RobChair framework tasks

Here are presented the diagrams of the RobChair framework tasks, or routines, detailed in Chapter 3:

- The main task:

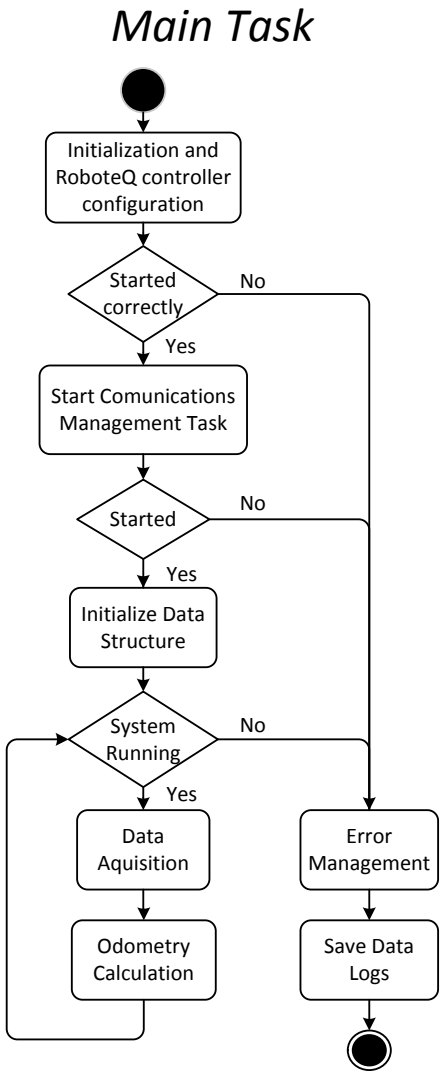


Figure C.1: RobChair Framework main task

- The communications management task:

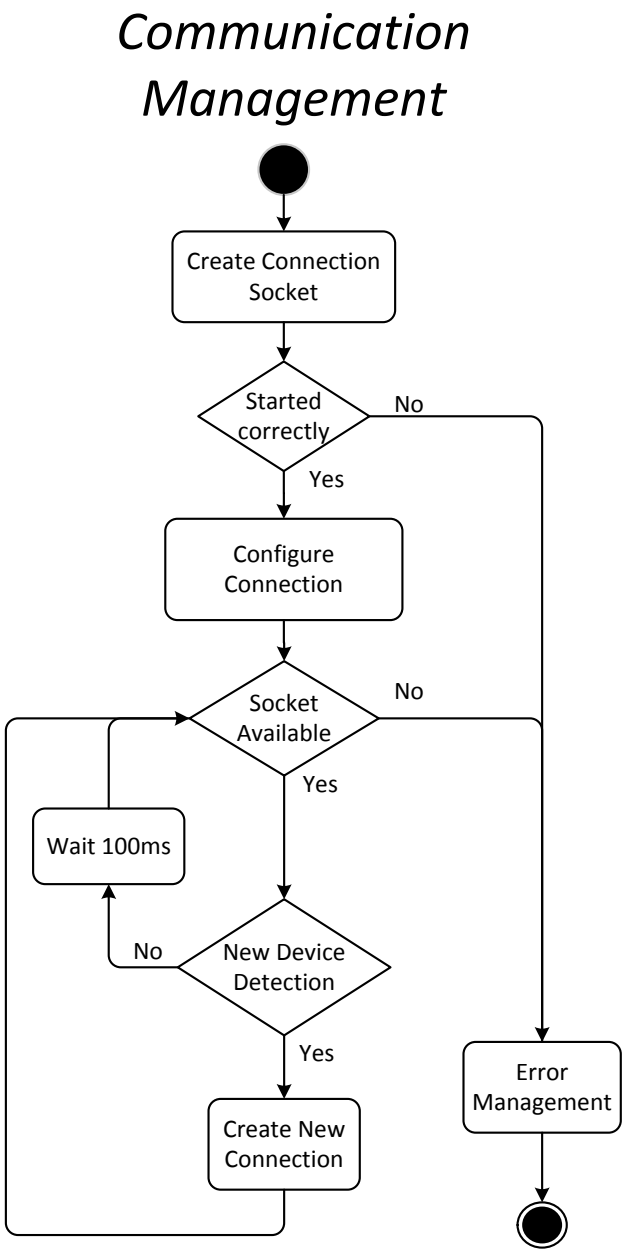


Figure C.2: RobChair Framework communication management task

- The client communication task:

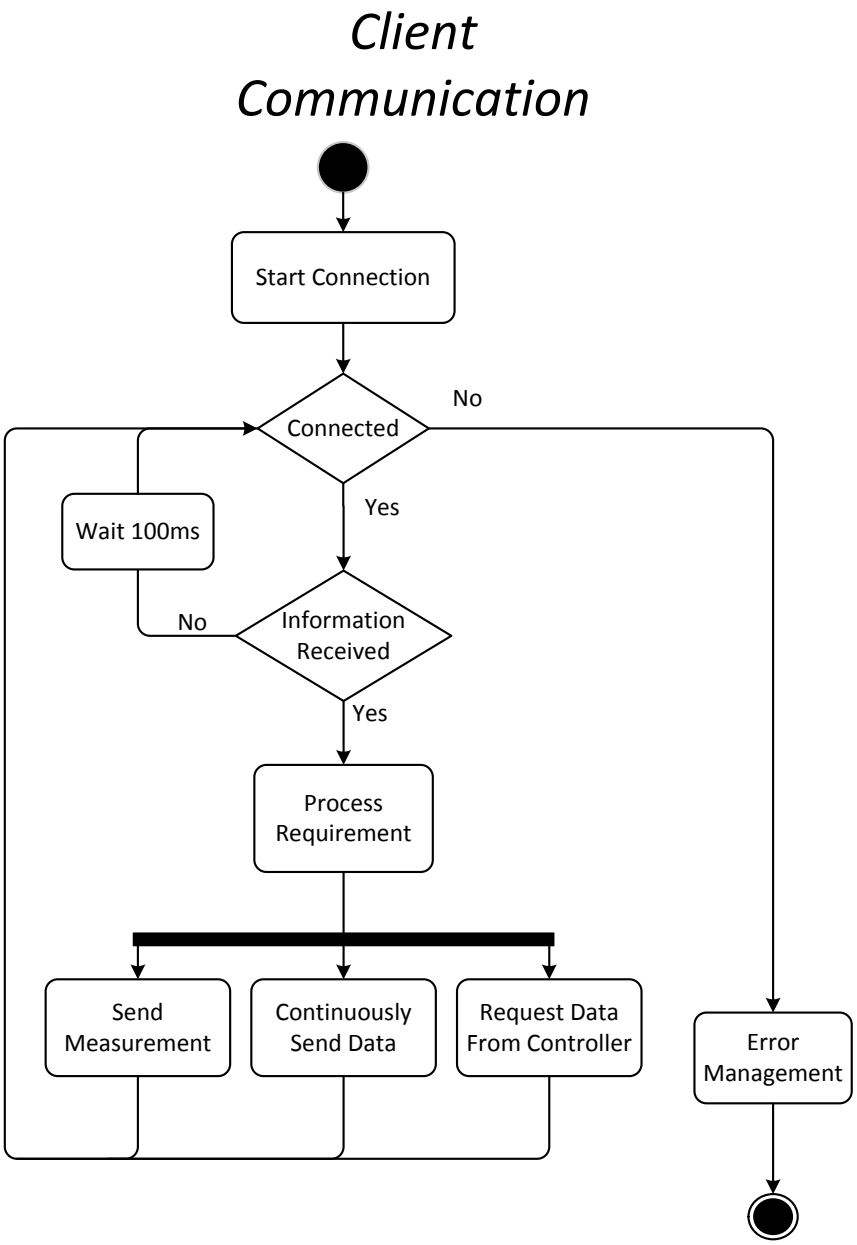


Figure C.3: RobChair Framework client communication task

Appendix D

Technical overview of ROS

A technical overview of ROS was provided in [Coleman, 2013], and the fundamental part is presented in this appendix.

ROS has three levels of concepts: the Filesystem level, the Computation Graph level, and the Community level. These levels and concepts are summarized below:

1. **ROS Filesystem Level**

   The filesystem level concepts are ROS resources that can be encountered on disk, such as:

   - **Packages**: Packages are the main unit for organizing software in ROS. A package may contain ROS runtime processes (nodes), a ROS-dependent library, datasets, configuration files, or anything else that is usefully organized together.
   - **Manifests**: Manifests provide metadata about a package, including its license information and dependencies, as well as language-specific information such as compiler flags.
   - **Stacks**: Stacks are collections of packages that provide aggregate functionality, such as a "navigation stack." Stacks are also how ROS software is released and have associated version numbers.
   - **Stack Manifests**: Stack manifests provide data about a stack, including its license information and its dependencies on other stacks.
   - **Message types:** Message descriptions, define the data structures for messages sent in ROS.
   - **Service types**: Service descriptions, define the request and response data structures for services in ROS.

2. **ROS Computation Graph Level**

   The Computation Graph is a peer-to-peer network of ROS processes that are processing data together. The basic Computation Graph concepts of ROS are: nodes, master, parameter server, messages, services, topics, and bags, all of which provide data to the Graph in different ways.

   - **Nodes**: Nodes are processes that perform computation. ROS is designed to be modular at a fine-grained scale; a robot control system usually comprises many nodes. For example, one node controls a laser range-finder, another one controls the wheel motors, other performs localization, and so on.
   - **Master**: The ROS Master provides name registration and lookup to the rest of the Computation Graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.
   - **Parameter Server**: The Parameter Server allows data to be stored by key in a central location. It is currently part of the Master.

- **Messages**: Nodes communicate with each other through messages. A message is simply a data structure, comprising typed fields.
- **Topics**: Messages are routed via a transport system with publish / subscribe semantics. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each others existence. The idea is to decouple the production of information from its consumption. Logically, one can think of a topic as a strongly typed message bus. Each bus has a name, and anyone can connect to the bus to send or receive messages as long as they are the right type.
- **Services**: The publish / subscribe model is a very flexible communication paradigm, but the many-to-many, one-way transport is not appropriate for request / reply interactions, which are often required in a distributed system. Request / reply is done via services, which are defined by a pair of message structures: one for the request and another one for the reply. A providing node offers a service under a name and a client uses the service by sending the request message and waiting the reply. ROS client libraries generally present this interaction to the programmer as if it were a remote procedure call.
- **Bags**: Bags are a format for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data that can be difficult to collect but is necessary for developing and testing algorithms.

The ROS Master acts as a name-service in the ROS Computation Graph. It stores topics and service registration information for ROS nodes. Nodes communicate with the Master to report their registration information. As these nodes communicate with the Master, they can receive information about other registered nodes and make connections as appropriate. The Master will also make callbacks to these nodes when this registration information changes, which allows nodes to dynamically create connections as new nodes are running.

Nodes connect to other nodes directly; the Master only provides lookup information, much like a DNS server. Nodes that subscribe to a topic will request connections from nodes that publish that topic, and will establish that connection over an agreed protocol (As seen in Fig. D.1). The most common protocol used in ROS is called TCPROS, which uses standard TCP/IP sockets.

3. **ROS Community Level**

The ROS Community Level concepts are ROS resources that enable separate communities to exchange software and knowledge. These resources include:

- Distributions: ROS Distributions are collections of versioned stacks that can be installed. Distributions play a similar role to Linux distributions: they make it easier to install a collection of software, and they also maintain consistent versions across a set of software.
- Repositories: ROS relies on a federated network of code repositories, where different institutions can develop and release their own robot software components.

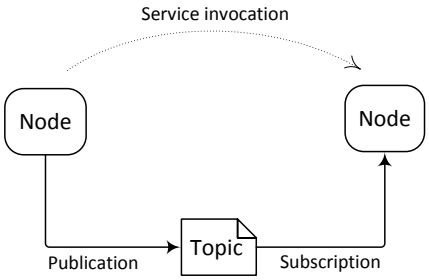


Figure D.1: ROS node working concept

- The ROS Wiki: The ROS community Wiki is the main forum for documenting information about ROS. Anyone can sign up for an account and contribute their own documentation, provide corrections or updates, write tutorials, and more.
- Bug Ticket System: If someone find an issue with ROS or ROS-related software, or wish to request a feature, it can use the issue-tracking system to file a ticket.
- Mailing Lists: The "ROS-users" mailing list is the primary communication channel about new updates to ROS, as well as a forum to ask questions about ROS software.
- ROS Answers: A Q&A site for answering ROS-related questions.