Ricardo Jorge Santos da Cruz

Interbot Mobile Robot: Human-Robot Interaction Modules

Thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Electrical and Computer Engineering

September, 2017

Universidade de Coimbra

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Interbot Mobile Robot: Human-Robot Interaction Modules

Ricardo Jorge Santos da Cruz

Coimbra, September 2017

Interbot Mobile Robot: Human-Robot Interaction Modules

Supervisor:
Professor Doutor Urbano José Carreira Nunes

Co-Supervisor:
Professora Doutora Ana Cristina Barata Pires Lopes

Jury:
Prof. Dr. Rui Alexandre de Matos Araújo
Prof. Dr. Jorge Nuno de Almeida e Sousa Almada Lobo
Prof. Dr. Urbano José Carreira Nunes

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

Coimbra, September 2017

Acknowledgments

This dissertation could not have been completed without the guidance and assistance of many people to whom I would like to express my appreciation and gratitude. I am very grateful to my advisors, Professor Doutor Urbano Nunes and Professora Doutora Ana Lopes, for giving me the chance to work on a subject that I enjoy, for offering me support in a great number of ways, and for the guidance and suggestions provided along the completion of this work.

I am also grateful to all my laboratory colleagues that helped me along the way, as well as the camaraderie between everyone which resulted in a creative, dynamic and enjoyable workplace. To Daniel Almeida and Tiago Barros, for their patience and constant availability towards helping with the technical aspects of Interbot platform.

A special mention to Luis Garrote, for always providing me with great advice, fruitful and insightful discussions, by keeping a sense of humor when I had almost lost mine, and for motivating me towards fulfilling my objectives.

I thank ISR for providing me with excellent conditions and resources, that allowed me to accomplish this important milestone. This work has been supported by Fundação para a Ciência e Tecnologia (FCT) through project "AMS-HMI12:RECI/EEI-AUT/0181/2012", and programs COMPETE 2020 / PORTUGAL 2020 through project grant "UID/EEA/00048/2013".

I am very grateful to all my friends, with a special mention to the ones from Orxestra Pitagorica, for their support and the good moments spent along this journey. Without them I would not have been able to stay the course, and I will always keep with me the memories and experiences lived during this academic journey.

Finally, I am deeply grateful to my family and their continuous support, specially to my parents, that provided me with everything necessary so that I could achieve my masters degree.

To everyone mentioned here, and to many others that are not... Thank you!

Resumo

A tecnologia robótica tem sido alvo de constantes avanços, e no entanto os robôs móveis ainda não estão completamente presentes no nosso dia-a-dia, locais de trabalho ou casas. Para que esses robôs façam parte de um ambiente humano, ainda é necessário um desenvolvimento significativo da tecnologia *Human-Robot Interaction* (HRI), de modo a que os robôs saibam como se comportar ao interagir com humanos, de modo a serem socialmente aceites. Quando aplicada a robôs móveis, a HRI pode envolver um grande espectro tecnológico, que vai desde interfaces de teleoperação e as suas limitações, o comportamento de cada humano e robô numa HRI, e como o diálogo entre as duas partes é conduzido.

Esta dissertação de Mestrado tem como objectivo principal desenvolver e implementar várias funcionalidades para o *Interbot-Social Robot*, um robô móvel desenvolvido no Instituto de Sistemas e Robótica (ISR). Essas novas funcionalidades permitem que o Interbot interaja com seres humanos e execute determinadas tarefas. Com esse objectivo em mente, foram adicionados dois novos componentes: uma estação de controlo remoto e um tablet, actuando ambos como interfaces homem-robô. A arquitectura do software desenvolvido foi projectada para ser modular e versátil, de forma a que seja fácil de usar e entender pelos futuros utilizadores e investigadores. Para desenvolver o software necessário para os novos componentes, as *frameworks* ROS (*Robot Operating System*) e Qt Creator foram utilizadas como as plataformas de desenvolvimento, teste e detecção de erros dos métodos e algoritmos que integram o novo sistema do Interbot. Diferentes tipos de controlo e comportamentos nas HRI foram analisados e avaliados para esta plataforma.

Uma série de testes e experiências foram feitos para medir o desempenho dos novos componentes do Interbot, através de diversos cenários de teste com diferentes configurações. As novas interfaces homem-robô provaram ser uma forma acessível para os utilizadores interagirem e controlarem o Interbot, participarem numa visita guiada pelo piso 0 do ISR, ou para delegar tarefas como entrega de mensagens ou objectos ao Interbot. Verificou-se que a estação remota era capaz de planear e delegar tarefas sequencialmente, enquanto exibia as informações do robõ de forma perceptível. Finalmente, foi realizada uma análise aos efeitos da latência na performance do sistema e foram apresentadas possíveis soluções para futuras melhorias.

Palavras chave: Interacção Homem-Robô, Teleoperação, Controlo Supervisionado, Interfaces Homem-Robô, Comportamentos Homem-Robô, *Interbot*

Abstract

Robotic technology has had many advances, and yet mobile robot are still not completely present in our daily environments, workplaces or homes. For these robots to make part of a human environment, a significant development of Human-Robot Interaction (HRI) technology is still required, so that robots know how to behave while interacting with humans towards being socially accepted. When applied to mobile robots, HRI can cover a large technological spectrum ranging from teleoperation interfaces and their limitations, human and robot roles in a HRI, and how the dialogue between both parties is conducted.

The main goal of this Master's dissertation is to develop and implement several functionalities for Interbot- Social Robot, a mobile robot being developed at the Institute for Systems and Robotics (ISR). These new functionalities allow Interbot to interact with humans and to perform given tasks. With that goal in mind, two new components were added: a remote control station and a tablet, acting both as human-robot interfaces. The architecture of the developed software is designed to be modular and versatile, in a way that it is easy to use and understand by future users and researchers. To develop the required software for the new components, the ROS (Robot Operating Systems) and Qt Creator frameworks were used as development, test and debugging platforms of the methods and algorithms integrating the final Interbot system. Different types of control and HRI roles were analyzed and evaluated for this platform.

A series of tests and experiments was done to measure the performance of the new Interbot components, using different test scenarios and control types. The developed human-robot interfaces provided a reliable way for users to interact and directly control Interbot, to be escorted during a guided tour of ISR floor 0, or to order Interbot to perform tasks such as delivering messages or items between laboratories and rooms. The remote station was found to be capable of planning tasks sequentially, while displaying the robot information in a perceivable way. Finally, a latency analysis was performed by examining its effects on the system performance, and possible solutions for future improvements are presented.

Key words: Human-Robot Interaction, Teleoperation, Supervisory Control, Human-Robot Interfaces, Human-Robot Roles, Interbot

"It's not when you get there, it's always the climb."

Contents

List of Acronyms

AMCL	Augmented Monte Carlo Localization
BMS	Battery Management System
CMU	Carnegie Mellon University
D-DWA	Double-Dynamic Window Approach
DWA	Dynamic Window Approach
EKF	Extended Kalman Filter
FOV	Field Of View
GUI	Graphical User Interface
HRI	Human-Robot Interaction
HCI	Human-Computer Interaction
HMI	Human-Machine Interaction
HMP	Hybrid Motion Planner
IMU	Inertial Measurement Unit
ISR	Institute of Systems and Robotics
JSON	JavaScript Object Notation
LED	Light-Emitting Diode
LCD	Liquid-Crystal Display
MCL	Monte Carlo Localization
ROS	Robot Operating System
SLAM	Simultaneous Localization And Mapping
TCP	Transmission Control Protocol
UI	User Interface
USB	Universal Serial Bus

List of Figures

List of Tables

Chapter 1

Introduction

This chapter presents the main motivations that lead to this dissertation, the context of the developed work, as well as the main goals and key contributions.

1.1 Motivation and Context

The ultimate goal of robot designers is to develop robots that can perform tasks autonomously. Even so, there are situations where the robot's capabilities can be insufficient for the task at hand, and yet the only thing needed to complete the task would be advice or help from a human. Robots have an incredible processing power that proves to be useful in structured environments, allowing the use of computer algorithms or well-defined processes. However when they are required to make an unstructured decision, or when common sense is required, a human can typically outperform the robot. Therefore, as portrayed in [1], the combination of autonomous task execution, remote collaborative or direct control, and the capability to instantly request human collaboration during task performance may be an appropriate solution for a mobile robot working in a populated environment.

This dissertation work plans to develop and implement functionalities for "Interbot-Social Robot". This robot was developed in ISR thanks to the previous work of several researchers. At the start of this work, Interbot was already able to navigate autonomously [2]. Towards improving the social aspect of Interbot, two new components have to be added to the platform: a remote control station and a tablet acting as human-robot interfaces. Software for these two new components has to be developed using a modular approach so that they can be decoupled from the system. The resulting system must be reliable and allow for direct and supervisory control, as well as be capable of having dialogue capabilities so that information is properly conveyed to the users, and requests for assistance to be possible.

1.2 Goals

The main purpose of this work is to extend the features of Interbot, by adding a human-robot interface and an acessible remote control station. Since it is intended to navigate Interbot in human environments, it is of utmost importance to extend the previous Interbot architecture to allow for a robust human-robot interaction.
The main goals are as follows:

1. To research different types of robots operating in populated environments, giving particular attention to user interaction systems: human-robot interface and remote controllers.

2. To improve the previous Interbot architecture, by adding two new components: a Base Station for remote control/supervision, and a Tablet to allow a friendly interface for human interaction.

3. To develop software for the new components with a high degree of modularity, so that any modules developed may be used to control multiple robots in a networked human-robot environment.

4. To perform several experimental tests to evaluate the performance of the whole system, and of the new components.

When developing software, there are certain requirements that must be defined previously [3]. For the purpose of this work, the most crucial demands for the new implemented software components were:

1. The new software components must be decoupled from the previous existing architecture, and multiple-platform compatible.

2. Both the Base Station and Tablet must contain a Graphical User Interface (GUI) that requires little to no training to allow a rapid comprehension of the current robot state.

3. In terms of communication, the connection between the Base Station and the Tablet must have a well-defined transport protocol and use human-readable text for data transfer.

4. Security measures have to be implemented to prevent any material or human damage in the event of a bad or loss of connection.

1.3 Implementations and key contributions

Figure 1.2 shows the two new components added to Interbot's previous platform, Tablet and Base Station, and the purpose of each software module implemented in both components. The following main implementation and contributions are described in this dissertation:

Interbot Platform (Chapter 4)

- Description of Interbot's architecture, represented in Fig.1.1, giving particular attention to the two new components: a remote Base Station for monitoring and supervision, and a Tablet interface for human-robot interaction.

- Overview of Interbot's distributed software architecture, with a concise description of the software modules integrated in the three main components.

Software Design: Requirements (Chapter 5)

- Description of the proposed solutions to comply with the established requirements, as well as the rationale behind the chosen solutions.

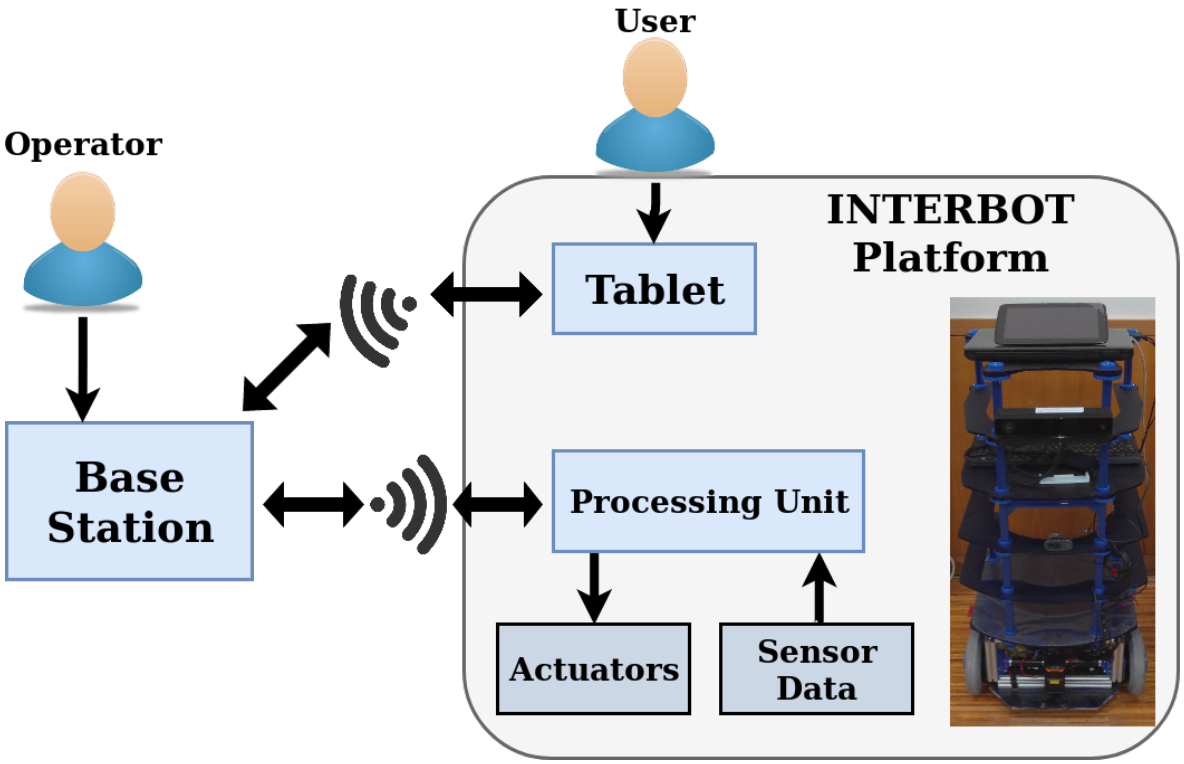


Figure 1.1: Overview of Interbot platform, with the integration of the two new components.

Software Design: Development (Chapter 5)

- **Communication method**: A client/server relationship using Transmission Control Protocol (TCP) sockets was used for communication between Base Station and the Tablet, containing JavaScript object Notation (JSON) encapsulated messages.
- **Safety**: Different timers were added to prevent malfunctions in the event of communication failures.
- **User Interface (UI)**: UI's for each of the new components were developed with the targeted goal of being accessible to any user, simple to understand and to quickly comprehend Interbot's state.
- **Teleoperation**: Two methods were implemented to directly control Interbot: a virtual joystick and a keyboard/arrowpad controller.
- **Human-Robot Interaction**: Using each of the UI's, a user can request or assign tasks to Interbot, as well as being requested to assist towards a completion of a task.
- **Path Planning**: A path planning algorithm applied to be used together with a topological map was designed.
- **Supervisory Control**: A supervisor function was implemented, that constantly checks the robot state, displays the information, handles communication, plans and orders all the tasks.

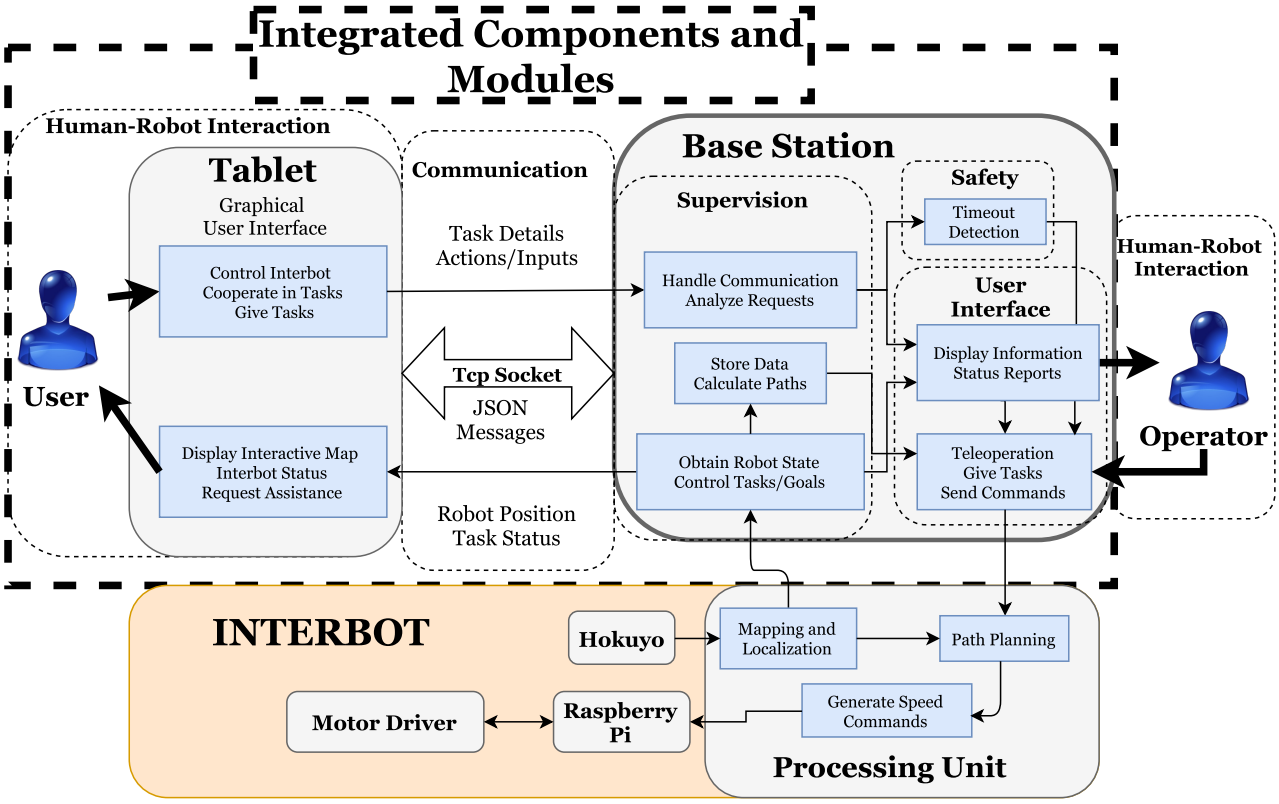


Figure 1.2: Integrated components corresponding to the key contributions and respective modules.

Chapter 2

State of the art

2.1 Indoor Mobile Robots

In our modern society, there is an ever-growing need to use autonomous robots to execute different types of tasks that can easily be replaced without disturbing their integrated environment. There is a wide variety of robots such as guide robots, or interaction and service robots, that were developed to help people overcome their difficulties while performing a multitude of different tasks such as: (a) perform messenger tasks, improve productivity of employees, escort visitors or carry objects in office environments [4]; (b) carry heavier resources of components over long distances in industrial facilities; (c) perform guided tours explaining the attractions, answering questions and conducting satisfaction surveys in museus; (d) monitoring and helping patients in hospitals.
Figure 2.1 displays a typical architecture of an indoor mobile robot. These robots must be able to localize, navigate and perceive their surrounding environment so that they are capable of performing these tasks while avoiding obstacles, in a dynamic changing environment and operating during long-term deployments with or without supervision. Since these robots typically are integrated in populated environments, their social aspect is paramount. To be socially accepted, they must be able to know how to behave and interact with humans and present a friendly, accessible interface.

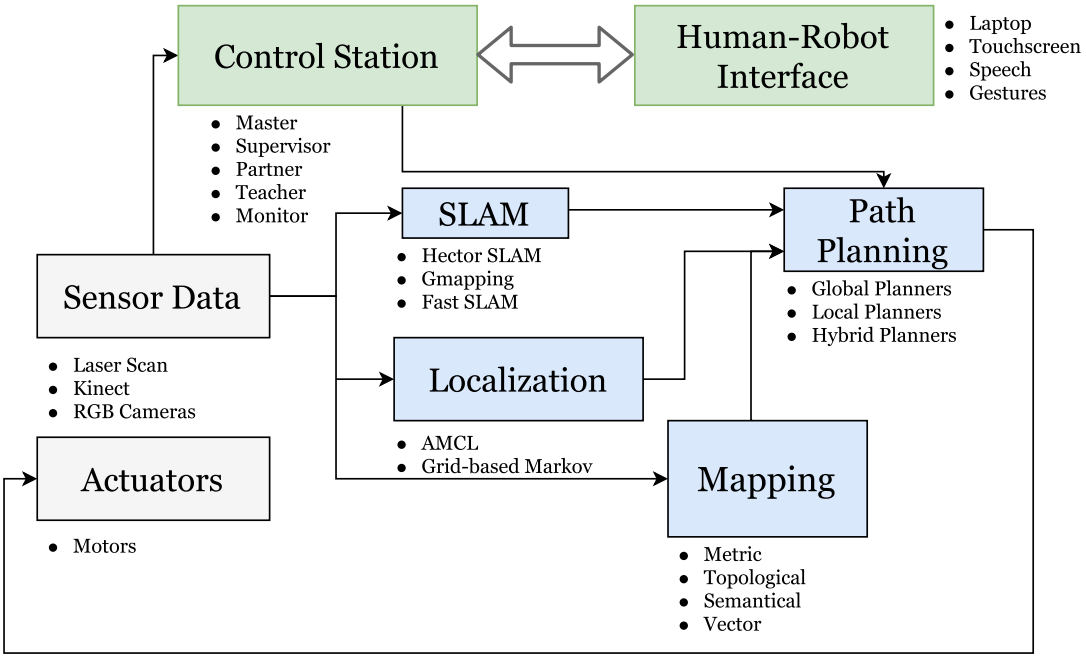


Figure 2.1: Typical basic architecture of an indoor mobile robot, the main modules and some possible choices for the methods used by the implemented modules.

2.1.1 Human-Robot Interaction

Human and robots have interacted for many decades, since the 1940's [5]. While in the beginning this interaction was uni-directional, over time this relationship has developed similar to one between two human beings, as robots became more intelligent. Human-Robot Interaction (HRI) regards the analysis, design, modeling, implementation and evaluation of robots for human use. HRI differs from human-computer interaction (HCI) and human-machine interaction (HMI) because it concerns robots, which have complex and dynamic control systems, exhibiting autonomy and cognition while operating in changing, real-world populated environments.
HRI can occur through physical contact, or be mediated by a user interface [6]. This interface acts as a translator: translating from human input to robot commands, and a way to provide feedback in the displays. If this interaction is separated by a barrier, like distance or time, and information is exchanged through a communication link, then it is called teleoperation.

2.1.1.1 Teleoperation in Robotics

There are important demands that have to be met for a secure robotic teleoperation, as mentioned in [7], such as: (i) reliable navigation; (ii) efficient command generation from human inputs; and (iii) trustworthy sensor data; The human-robot interface must also be as efficient and capable as possible, by providing tools to perceive the remote environment and to allow proper decision-making before commands are generated. The majority of interfaces are designed to minimize training or to be user adaptive, and try to maximize information display while minimizing sensory and cognitive workload. Even if a robot displays an elevated level of autonomy, it still needs to convey to an operator how and what it did during execution of tasks [8]. This is important whenever the robot encounters unexpected problems or fails to complete a task.
Operator interfaces used in robotic teleoperation can be divided in categories, such as:

- **Direct Control** - An operator directly controls the robot using controllers like joysticks, while watching a video feed from a mounted camera on the robot.
- **Supervisory Control** - The operator divides a problem into a sequence of subtasks that are ordered for the robot to execute on its own. This means the robot must be somewhat autonomous, and be capable of achieving goals while staying safe. The interface typically provides tools for task planning, sequence generation of commands, monitoring and diagnosis of the robot's state, allowing for the operator to monitor and identify execution anomalies.
- **Novel** - State of the art interfaces are usually called "novel". This is an ever-changing definition, since present day interfaces such as touch screen devices, web-based or gesture capture were once called novel, and yet are now quite common. Nowadays, there is mostly focus on haptic interfaces using virtual realities for teleoperation.

2.1.1.2 Limitations of Teleoperation

Teleoperation can be a challenging task since the operator has to operate the robots at a remote distance through sensor feedback while maintaining situational awareness, depth perception and obstacle detection. There are also many factors that can compromise teleoperation inside office environments, according to [9], such as:

- **Field of View (FOV)**: A limited FOV can lead to erroneous speed and distance judgments, peripheral vision loss and degraded remote driving.
- **Orientation**: The operator needs to be situationally orientated, both globally and locally, so that areas of interest such as corners or obstacles can be relatively localized.
- **Video Frame Rate**: The communication channel between the operator and the robot can be overloaded with other sensor information, which leads to a decrease in the quality of the video feed.
- **Time Delays**: If the delay between operator commands and visual feedback is variable and unpredictable, it can lead to overactuating and repeated command-issuing.

2.1.1.3 Human and Robot Roles in HRI

A robot is, in some occasions, viewed as a device that is only able to perform tasks on command. This is sometimes a misconception , which leads to a limitation on the robot's freedom to act, and in a inability of acting beyond what has been programmed [10] . Therefore, if a robot is constructed without considering human collaboration, it will have no way to ask for assistance to perform a task.
To allow human-robot collaboration, roles have to be defined for both. Relationships between humans and robots depend on these roles [11] , and can be classified as follows:

- **Master-Slave**: Used in typical direct teleoperation systems, as mentioned in Section 2.1.1.1. Human(master) has full control over the robot (slave) and all decision are made by the human.
- **Supervisor-Subordinate**: The robot has the ability to plan and execute intermediate steps,while considering events and situations using minimum human intervention. In the occurrence of a problem that the robot can't solve, the human supervisor is responsible for finding a solution or producing a new plan for the robot.
- **Peer-To-Peer**: Using sliding autonomy [12], the robot has the ability to identify situations it is not capable to solve, and pro actively asks for assistance from humans when needed.
- **Partner-Partner**: The robot is viewed as a work partner, working interactively with the human, allowing both to take advantage and learn from each other's skills, advice and expertise. Whenever a problem occurs, the robot can request or give assistance.
- **Teacher-Learner**: Human has the role of a teacher, while the robot must have sufficient intelligence to learn from the user. When the robot has completely mastered the task, it can replace the human or work together, depending on the context.

- **Fully autonomous**: Robots that can operate without any human intervention once a mission is defined. Human can only monitor and not influence the robot operation.

It then seems clear that there are benefits if humans and robots collaborate. If we treat a robot as a partner, instead of a tool, more meaningful work and better results can be accomplished. However, for collaboration to happen between humans and robots, a dialogue must exist and questions exchanged in order to jointly solve problems.

2.1.1.4 Dialogue between Humans and Robots

When humans and robots communicate, dialogue is typically mediated by an interface [13]. Some interfaces offer great power and adaptability , but contain an associated high-learning cost. Other interfaces containing objects like menus, buttons or labels are easier for beginners, and should require little to no training to comprehend [14]. Interface models for human-robot dialogue are: command languages, natural language (speech/text), question and answer, menus and graphical user interfaces with direct manipulation. Dialogue between humans and robots has to be managed, so that user requests are translated into a language the robot understands and, on the other hand, the robot system outputs are also understood by humans.

2.2 Guide and interaction robots

Mobile robots that provide tour guides while interacting with people, or that perform tasks such as delivering messages or transporting objects in a busy office environment, must have implemented proper safety measures in order to prevent material or human damage, as well as be interactive so that they can be seen as trustful and as a source of help for fellow coworkers, as portrayed in [15]. In the past, one of the main challenges for these robots was the execution of a reliable navigation through crowds while moving at a reasonable speed without colliding with people or objects. With recent technology advances bringing new and more powerful sensors/processors, together with refined mapping, localization and path-planning algorithms, safe navigation has become a standard for guide and interaction robots.
Over the years, another main challenge presented by these types of robots was their social aspect. Since they are supposed to work alongside humans with different ages and skill-sets , they must hold an intuitive, appealing and easy to use interface, which must act as a two-way platform where users can interact with the robot, and vice-versa. A list of tested and documented guide and interaction robots is presented in Table 2.1, with a description of the main innovations, the technologies applied and the HRI interfaces that were used.

Table 2.1: Documented guide and interaction mobile robots.

Robot	**Description/Main Innovations**	**Navigation and Localization Methods**	**HRI Interface**
Rhino (1997) [16]	Acted as a museum tour guide during a six-day deployment period. Able to navigate at high speed through crowds, while avoiding obstacles even if they can't directly be perceived. "Virtual telepresence", using a Web Interface.	Markov localization; Metric environment map; Value iteration planning; μDWA algorithm.	On-board mixed-media interface integrating text, graphics, recorded speech and sound. User input is given through buttons. Web interface used for teleoperation, with camera images, environment map and robot information.
Robox (2001) [17]	Museum guide during Expo 02, guiding nearly a million people over 5 months, seven days a week, eleven hours a day. Multi-dynamic system for interaction in social scenarios.	Extended Kalman Filter (EKF) localization approach; Topological maps used for path planning; Smoothing process to avoid obstacles	Social behavior depended on scenarios, triggered by special events. Capability of speech in four languages, robot motion and button illumination. LED matrix capable of representing facial expressions, icons and animations.
Jinny (2004) [18]	Tested in office buildings and exhibits of Hyundai industries, interacting with people and performing tasks. Motion algorithm selection depended on environment condition. Integration of a knowledge base for human friendly interactions.	Monte Carlo Localization (MCL) algorithm localization; Topological map for global path planning, grid map for mapmatching, and active maps for local path planning	Receptive elements: voice recognizer, touchscreen and 12 LED buttons. Expressive elements: Voice synthesizer, LED matrix for expressions of emotions, LCD display, gestures.
PR2 (2010) [19]	Autonomous indoor robot for real office environments, capable of performing a wide variety of tasks such as fetching coffee, retrieving items from a refrigerator or cleaning a table. Performed the "office marathon", which consisted of navigating 42km in a populated office without incidents, while completing tasks.	With an a priori map, uses AMCL algorithm. Otherwise, it estimates position using odometry merged with IMU data; A* algorithm for path planning; DWA algorithm for collision avoidance.	Two arms and grippers to grab and manipulate objects. Pan tiltable head, omni directional base, stretching spine. Speech recognition and text-to-speech output.
Cobot (2011) [4] [20] [21]	Developed at Carnegie Mellon University (CMU), the Cobots performed autonomous tasks in a populated office environment, traveling over 200 Km for more than three years without hardware failures and minimal maintenance. Symbiotic autonomy with humans, overcoming perceptual and actuation limitations by asking for help with tasks from humans.	Robust real-time localization using Wi-fi data and depth camera information; Vector map representing obstacles by a set of line segments. Topological map for path planning; Obstacle avoidance is done by computing path lengths available in different angular differences	Web interface for semi-autonomous telepresence containing visual and motion control, task selection and information display. Touch-screen and speech-enabled tablet, microphones and speakers.

Chapter 3

Background Material

3.1 Mapping

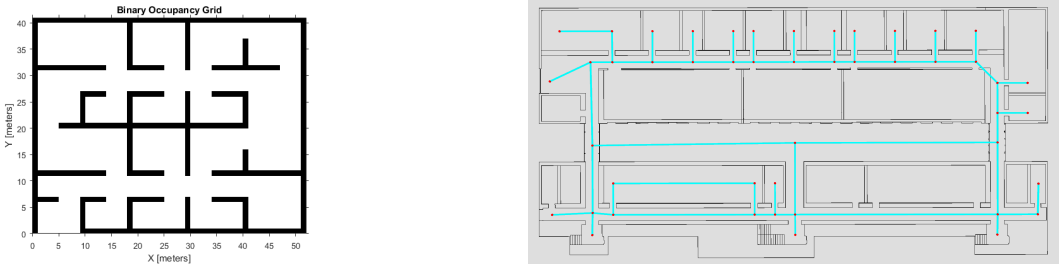
An autonomous robot must be able to either build a map representing its surrounding environment using different types of sensor data, or to use a given map, called a priori map. These maps are then used by the localization algorithms, so that the robot can localize itself on the map and by the navigation algorithms to generate safe paths between locations on the map. Two major types of maps are considered in [22], and were the ones used for the purpose of this work.

3.1.1 Metric Maps

Metric maps considered in this work are discrete, two-dimensional occupancy grids as proposed in [22]. Each grid cell <x,y> has associated a value with it that measures the probability of the cell being occupied. This occupation measure value is increased in each iteration if the laser scan sensor readings indicate that the cell is occupied, and decreased if the scan readings find an empty cell. The complexity of grid-based maps depends solely on the resolution: lowering cell sizes will increase resolution. An example of a metric map is represented in Fig. 3.1a).

3.1.2 Topological Maps

Topological Maps are built on top of grid-based maps [22], allowing for a high-level representation of the environment. Nodes are used to represent landmarks/checkpoints and they are connected with each other using arcs, that correspond to the path between adjacent nodes. Shortest paths in topological maps can be easily found using a graph search algorithm, such as Djikstra's or Floyd and Warshall's [27] shortest path algorithm. Figure 3.1b) shows an example of a topological map.



(a) (b)

Figure 3.1: Two types of maps used by autonomous robots: (a) Metric map: Black cells are occupied, while white cells are empty. (b) Topological map: Blue lines represent paths while the red spots are nodes.

3.2 Hector SLAM

A robot can only navigate autonomously if it is able to localize itself in a map, in a way that it can correctly perceive its surrounding environment. Although there are many different algorithms to map environments and many more to localize the robot, the Hector SLAM [23] algorithm was the one chosen to use in this work because it is able to perform simultaneously localization and mapping using only laser scan data. Figure 3.2 displays an overview of the inputs required for the Hector SLAM algorithm, and the outputs produced by it. The Hector SLAM's approach attempts to perform an alignment of the laser scan's points with the grid map that is being published or through a map obtained before , a priori map. The scan matching process takes into account the previous scans, and attempts to align them with the grid map resulting in the robot's current pose. If a new static obstacle appears and it didn't previously exist in the map, occupancy grids are correctly updated so that the robot can avoid these obstacles. In the event of a dynamic obstacle appearance, like a moving person, the Hector SLAM iterates continuously to make sure that this obstacle should not be saved as a static one, so it is not updated on the map permanently.

Figure 3.2: Hector SLAM algorithm, inputs required and outputs produced.

3.3 Hybrid Motion Planner

The Hybrid Motion Planner (HMP), depicted in [24] is a path planning method integrated in a framework named Collabnav, developed at the Institute of Systems and Robotics of Coimbra University (ISR-UC). This framework was initially designed to work in a brain-actuated robotic wheelchair so a safe navigation was required in constricted environments, using smooth trajectories to avoid collisions at all costs. For the purpose of this work, only the path planning module of Collabnav will be used. Figure 3.3 represents HMP modules, and the way they are related.
Seeing as HMP is a hybrid planner, it combines the advantages of both global and local approaches, containing two separate planners: a global planner using a modified A* algorithm, and a local planner to solve situations that can not be solved by the global planner.

3.3.1 Costmaps

Three different types of costmaps are used: (1) a 2D global costmap, obtained from the a priori metric map, that is used to avoid obstacles detected by the laser scanner in a certain radius. Obstacles inside this radius are inflated to mark all the areas where the robot's position is not valid. Therefore, the robot will tend to stay away from obstacles facilitating the execution of maneuvers;

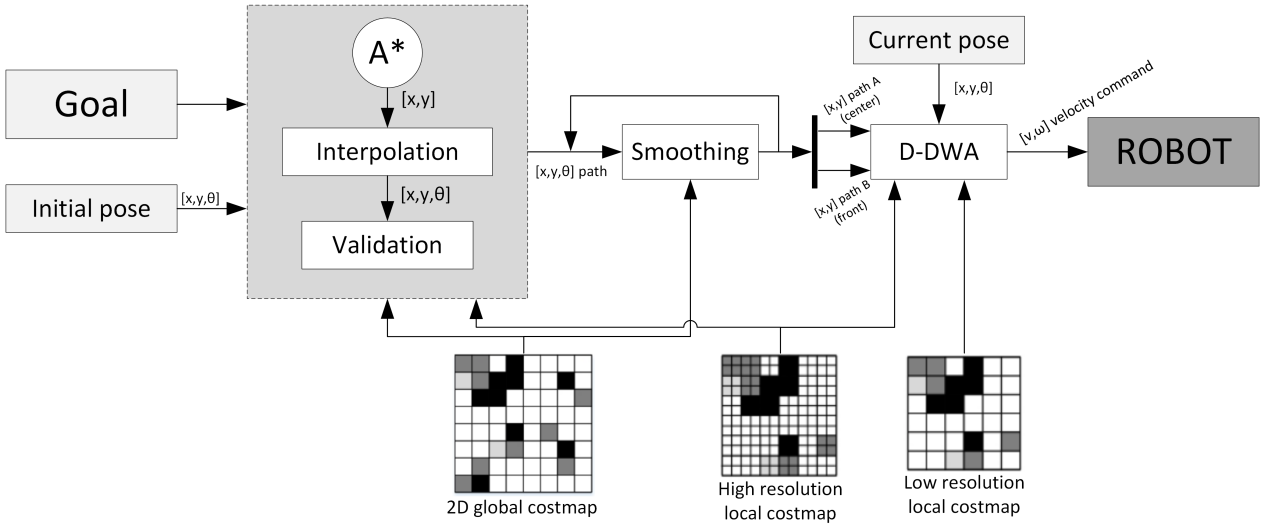


Figure 3.3: Hybrid Motion Planner (adapted from [24]).

(2) a low resolution local costmap, containing all the obstacles detected by the laser scanner used for D-DWA; (3) a high resolution local costmap, that also includes all the obstacles detected by the laserscanner, used in a specific D-DWA task to verify if a collision of the robot footprint occurs at any point of the planned trajectory, and therefore is used to validate the points generated by the 3D global path planner.

3.3.2 3D Global Path Planner

The first global path to a defined navigation goal is obtained using the A* algorithm [25] in the low resolution local costmap. Using the A* algorithm on an inflated 2-D costmap , a quick approximation of a feasible path is obtained. To solve collision problems due to the robot geometry, a planning validation process is applied: if a point of the robot's footprint collides with obstacle in the high-resolution 2-D costmap, the respective pose is corrected to a valid one.

3.3.2.1 Smoothing Algorithm

The fast 3-D path obtained will very often contain sharp transitions in the robot orientation. To fix this, a smoothing algorithm based on elastic bands methodology [26] is applied to avoid sudden variations in the orientation of the valid path.

3.3.2.2 Double-Dynamic Window Approach

Based on the 3D path plan generated, two 2D global path plans are determined: one in relation to the robot center of mass, and the other one in relation to a middle point located at the front of the robot. Considering that the robot does not have a circular geometry, this guarantees the required position and orientation to avoid obstacles.
In each iteration, the D-DWA module computes the velocity dynamic windows for both 2D plans and determines two trajectory sets. In the event that there is a point of the planned trajectory that collides with an obstacle in the high resolution costmap, the trajectory is rejected.

3.4 Floyd-Warshall Algorithm

The Floyd Warshall [27] is a graph search algorithm that compares all possible paths between each pair of vertices. Every combination of edges is tested recursively by incrementally improving an estimate on the shortest path between two vertices, until this estimate is optimal. When applied to topological maps, it can compute the shortest path between any two nodes connected by arcs, representing vertices and paths respectively.
While usually the Floyd Warshall algorithm only provides the lengths of the shortest paths between all pairs of nodes, it is possible to reconstruct the shortest path between any two endpoint nodes. There is no need to store the shortest path from each node to all other nodes, since this would be very costly memory-wise. Instead, a shortest-path tree is calculated to store each node tree, allowing to reconstruct a path from any two connected node. Pseudo-code for the Floyd Warshall's Algorithm, together with shortest path reconstruction, is shown in Algorithm 1 and Algorithm 2 , respectively.

Algorithm 1: floyd_algorithm(int P[N][N],D[N][N])

Input: Matrix path[N][N] containing the nodes indexes initialized to null
Matrix dist[N][N] containing minimum distances between nodes, initialized to ∞
Output: Matrix path containing the shortest path tree for each node

```
1  foreach path (u, v) do
2      dist(u, v) ← w(u, v) {w(u, v) is the weight of the path (u, v) or, in other words, the
       distance between node u and v}
3      path(u, v) ← v
4  for k ← 1 to N do
5      for i ← 1 to N do
6          for j ← 1 to N do
7              if dist(i, j) > dist(i, k) + dist(k, j) then
8                  dist(i, j) ← dist(i, k) + dist(k, j)
9                  path(i, j) ← path(i, k)
10 return path
```

After obtaining the shortest path tree for every node, it is possible to obtain the shortest path between any two end-point nodes, using Algorithm 2

Algorithm 2: shortest_path(sp[N][N], path[N][N], u, v)

Input: Pointer to matrix sp[N][N], containing the shortest path between two end-point nodes
matrix path[N][N] containing the shortest path trees
$u, v \leftarrow$ two end-point nodes
Output: matrix sp[N][N] containing the shortest path between the two input nodes

```
1 if path(u, v) = null then
2     return
3 sp = u
4 while u ≠ v do
5     u ← path(u, v)
6     sp.append(u)
7 return sp
```

3.5 Software Development Tools

3.5.1 Robot Operating System

The Robot Operating System [28] (ROS) is a robotics software middleware that contains many services that would be expected to exist in a real operating system, while also providing tools and libraries for writing, building and running code across multiple computers. There is also a very active online community behind it, providing frequent updates for open-source software used on robotic applications, allowing for a collaborative robotics software development. In [29] a technical overview of ROS is provided, where important concepts are defined such as the notion of node, package, topic, message, subscriber and publisher. The architecture is fully decoupled: a publisher node publishes messages of a certain type on a topic, while a subscriber node subscribes to messages from a topic and processes the information.

Since ROS is designed to allow distributed computing, this means that a ROS system can be running across multiple machines in the same network simultaneously. For that, we need one Master node to be running on one of the machines, all of the nodes must be configured to that Master, and there must be complete bi-directional connectivity between all machines. Figure 3.4 displays an overview of a ROS based robotics system.

3.5.2 Qt Framework

Qt Creator is a software development framework containing a set of source code libraries that provide common functionality to a variety of applications, leading to a reduction in the programming workload.

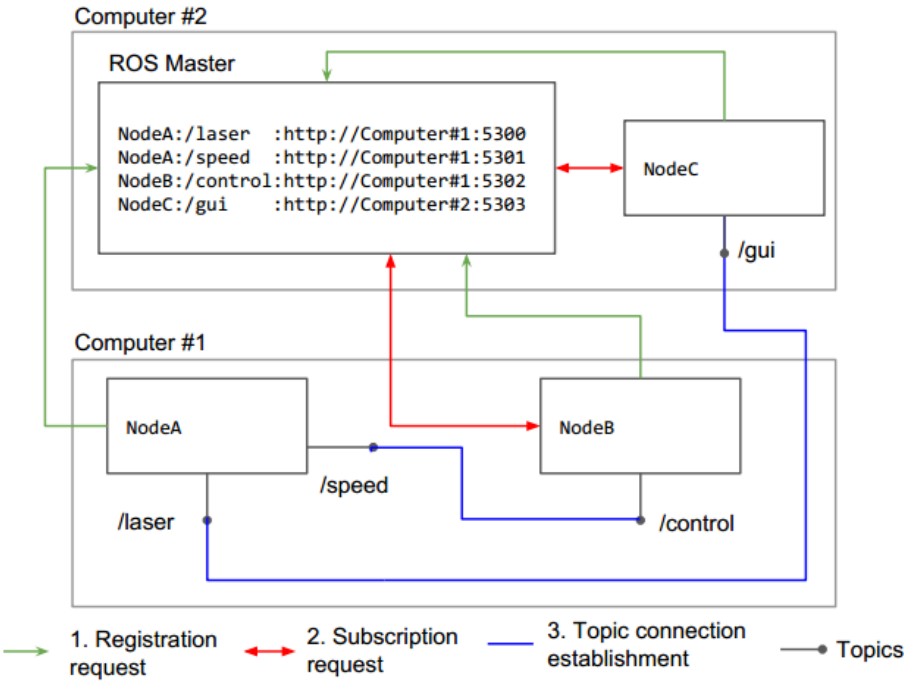


Figure 3.4: Overview of a ROS system. Nodes are registered in the master (green lines), nodes request information about the topic to subscribe (red lines), and direct connections are done between publishers and subscribers (blue lines)

3.5.2.1 Signals and Slots

One of the central features of Qt is the use of signals and slots mechanism for communication between objects. In applications that contain an user interface, like Interbot's Base Station or Tablet, when a button is clicked by the user it is expected to perform some operation that can lead to changes both in the GUI or in the underlying objects. As depicted in Fig. 3.5, in the occurrence of an event like this, a signal is emitted by the object. A slot is a function that is called in response to a signal, but can also act as a normal member function. Signals and slots are asynchronous: if a signal is emitted by a function, it does not block and continues its execution even if the signal is not captured by a slot. This way truly independent asynchronous components can be created with Qt.

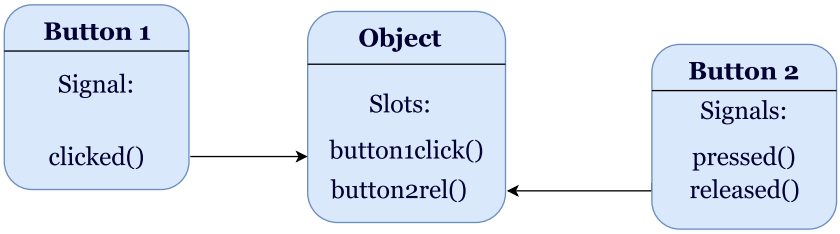


Figure 3.5: Signals and Slots connections between objects in Qt.

3.5.2.2 Signal Mappers

User Interfaces used in applications can sometimes contain many of the same object such as buttons, which emit the same type of signal `clicked()`. One approach is to connect each button's `clicked()` signal to its own custom slot, however this approach becomes very inefficient and extensive with an increasing number of buttons. Using a `QSignalMapper` class, all buttons can be bundled to a single slot, and are identified with a number. When a button's `clicked()` signal is emitted, `QSignalMapper` emits a `mapped(int)` signal which is connected to a function that obtains the `int` value. This way, for any possible number of `N` buttons, only two signal/slots connections are made.

3.5.2.3 Timers

The `QTimer` class provides a high-level programming interface for timers. A `QTimer` can be started by calling `start(ms)`, and after the time specified is passed, it emits the `timeout()` signal. A timer cannot fire while the application is busy doing something else, therefore the accuracy of timers depends on the granularity of the application. Another class of timers provided by Qt is the `QElapsedTimer`. These timers differ from `QTimers` since they do not emit a `timeout()` signal. Instead, they are used to calculate the elapsed time from the last time the corresponding `start()` or `restart()` functions were called. They are very useful to determine how much time was spent in any operation, function or even to calculate the latency in a connection.

3.5.2.4 Qt Network

Qt Network provides a set of tools and classes for programming applications that use TCP/IP communication. TCP(Transmission Control Protocol) is a network protocol used for data transfer. It is reliable, stream-oriented transport protocol, therefore is well suited for the continuous transmission of data.
The `QTcpSocket` class provides an interface for TCP. A TCP connection must be established to a remote host and port before data transfer can begin. `QTcpSocket` works asynchronously and emits signals to report data arrival, status changes and errors. Two independent streams of data are used: one for reading, and one for writing.

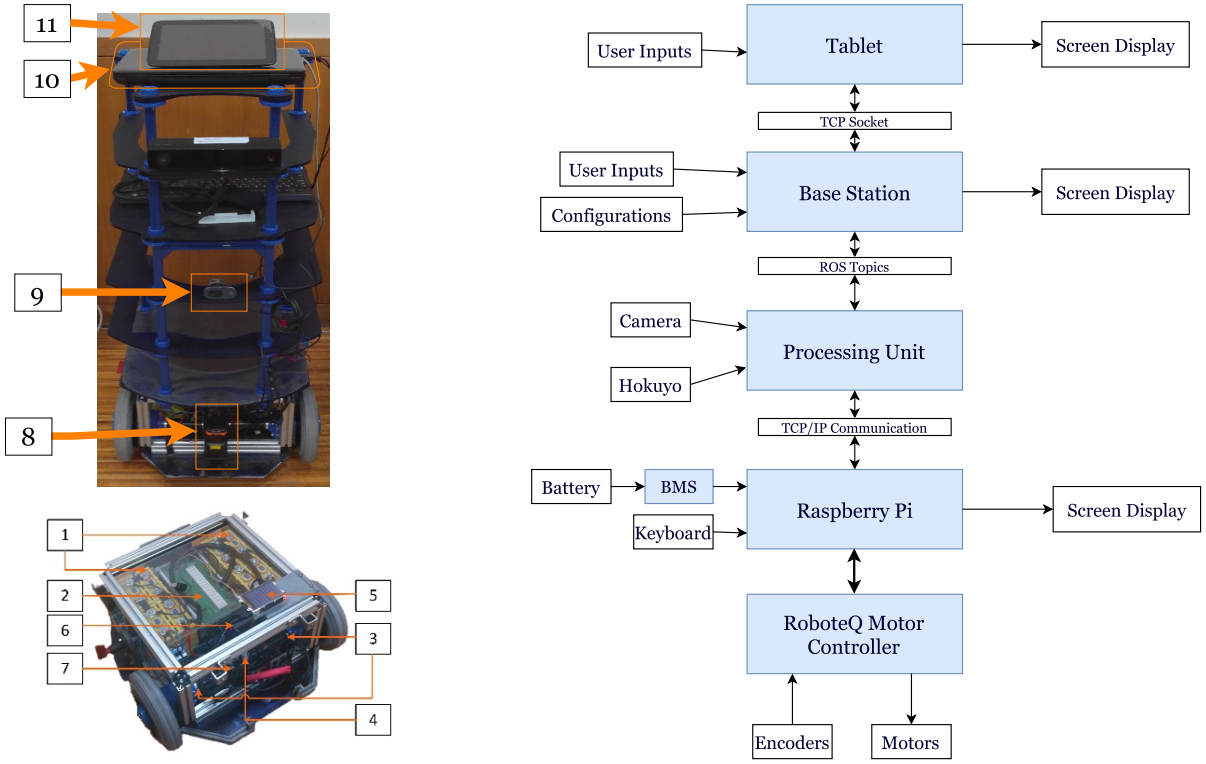
3.5.3 JSON File Format

JavaScript Object Notation (JSON) is a lightweight data-interchange format. It is commonly used for asynchronous client/server communication that uses human-readable text to transmit data objects over a connection. JSON is a text format that is completely language independent and easy for humans to read and write, making JSON an ideal data-interchange language.
The text representation of JSON encloses arrays in square brackets ([...]) and objects in curly brackets (...). Entries in arrays and objects are separated by commas. The separator between keys and values in an object is a colon (:).

Chapter 4

Interbot Platform

The Interbot Platform, described in [30], was developed in ISR-UC in the framework of the project "AMS-HMI12 - Assisted Mobility Supported by Shared-Control and Advanced Human-Machine Interfaces". Its main purpose is to allow the experimental testing of research methodologies on Human-Robot Interaction topics. This chapter gives an overview of the physical setup of the Interbot platform and describes both its hardware and software architectures, represented in Fig. 4.1a and Fig. 4.1b, respectively.



(a) Interbot Platform

(b) Interbot's Architecture

Figure 4.1: Interbot Platform is composed by: (1) 8 cells lithium battery; (2) BMS; (3) Two 24V DC motors; (4) RoboteQ Motor Controller; (5) Raspberry Pi 2; (6) 24V to 12V DC converter; (7) 12 to 5V DC converter; (8) Hokuyo UTM-30LX Laser Scanner; (9) Logitch Webcamera; (10) Processing Unit laptop; (11) Tablet.

4.1 Hardware Architecture

As it can be seen in Figure 4.1a and Figure 4.1b, Interbot's system is composed by a low-level part related to the power and mechanical components, and a high level part used for processing, user

interface and interaction. Each part of the hardware that composes Interbot is described below.

4.1.1 Tablet

A Nexus 10 tablet computer, that mainly acts as a Human-Robot Interface with an intuitive display, so that any user without training can give tasks to Interbot, move it freely or even help Interbot by indicating its initial position. It must be connected to the Base Station through a TCP socket to send requested goals/speed commands, or receive information such as: the current pose and orientation, the next destination or important status messages.

4.1.2 Base Station

A remote computer that allows for direct or shared control of the Interbot platform. It is fully decoupled from the lower levels of the Architecture since it only subscribes and publishes to ROS topics, therefore it could be used to control other, and even multiple, robots/platforms. The inputs can be configured, but usually they are the required information for visualization such as: the Camera stream, Hokuyo Laser scan and mapping data. It also controls the flow of information regarding the Tablet transmitted through a TCP socket.

4.1.3 Processing Unit

A laptop located in the Interbot platform, running software nodes to be used for mapping, localization and path planning, as well as generating speed commands to be sent to the Raspberry Pi. Receives as inputs the odometry information from the Raspberry Pi, laser range data from the Hokuyo Laser and an image stream from the Camera. The Processing Unit is located in the Interbot platform, so that incoming data from the Hokuyo laser scan used for navigation is obtained and processed with no delay. This also allows for Interbot to safely continue autonomously moving towards designated goals even if disconnected from the Base Station.

4.1.4 Vision Camera

The Logitech Camera is connected to the Processing Unit through USB connection and is used to obtain a real time video stream used for visualization purposes, so that a remote user using the Base Station can see what's in front of the robot and properly perceive the surrounding environment when performing teleoperation maneuvers.

4.1.5 Hokuyo Laser

The Hokuyo UTM-30LX Scanning Laser Rangefinder uses USB for communication and can obtain data in a field of view up to a distance of 30 meters and 270^{o} scanning range. It has a low power consumption, and can be used on battery-operated platforms like Interbot. Its long range scanning capabilities are very useful in long corridors, such as the ones in ISR, allowing it to generate scan messages of the surrounding environment that are sent to the Processing Unit to be used for mapping and localization.

4.1.6 Raspberry Pi

Raspberry Pi is a microcomputer that preferentially runs Linux and can be used in a vast array of applications. In this project, the Raspberry Pi was used as the interpreter between the low and high level parts of the Interbot platform. It connects to a RoboteQ Motor Controller, BMS and a keyboard. The Raspberry Pi receives encoder data from the RoboteQ Controller, processes the odometry data based on that information and sends it to the Processing Unit. It also receives speed commands from the Processing Unit and sends them to the RoboteQ Controller.

4.1.7 Battery Management System

The Battery Management System (BMS) manages the 8-cell lithium batteries that power Interbot and some of its components, protecting the batteries from operating outside safe values and sending the information to the Raspberry Pi.

4.1.8 RoboteQ Motor Controller

The RoboteQ Motor Controller receives speed commands from the Raspberry Pi and transforms them into voltage and current outputs towards driving one or two DC motors. It features a high performance 32-bit microcomputer and quadrature encoder inputs to read incoming pulses and transmit them to the Raspberry Pi using an USB communication so that they can be transformed into odometry information.

4.2 Software Architecture

In the previous version of Interbot's architecture, the highest level component was the Processing Unit so it had to perform all the computation regarding navigation, path planning, communication with the low-level parts, receive user inputs and display information to the user. The Processing Unit software modules are mainly programmed using ROS, and use launch files for setting configuration parameters, so whenever it is intended to change any parameter, the programs have to be shut down, so the settings can be configured, and relaunched.
Therefore, to reduce computational workload, increase software modularity and improve Human-Robot interaction two new components were added: Base Station and a Tablet. Figure 4.2 shows how the new components behave in the system architecture, as well as the possible control types, HRI roles and HRI interfaces as mentioned in the State of the Art.

The software developed for these two new components has a high level of modularity and portability: both of them contain multiple smaller modules that have specific functions, and can be re-utilized. For instance, the Virtual Joystick that will be depicted in Section 5.2.2.3 can be used with both touch or mouse inputs, depending on the interface. Designing software using a modular approach makes software debugging and bug fixing is easier to fix since errors can be traced to specific modules, thus limiting the scope of error searching. Most of the configuration parameters in these two new components are initialized with default values, but can be safely changed while the program is running, allowing for configurations on the fly.

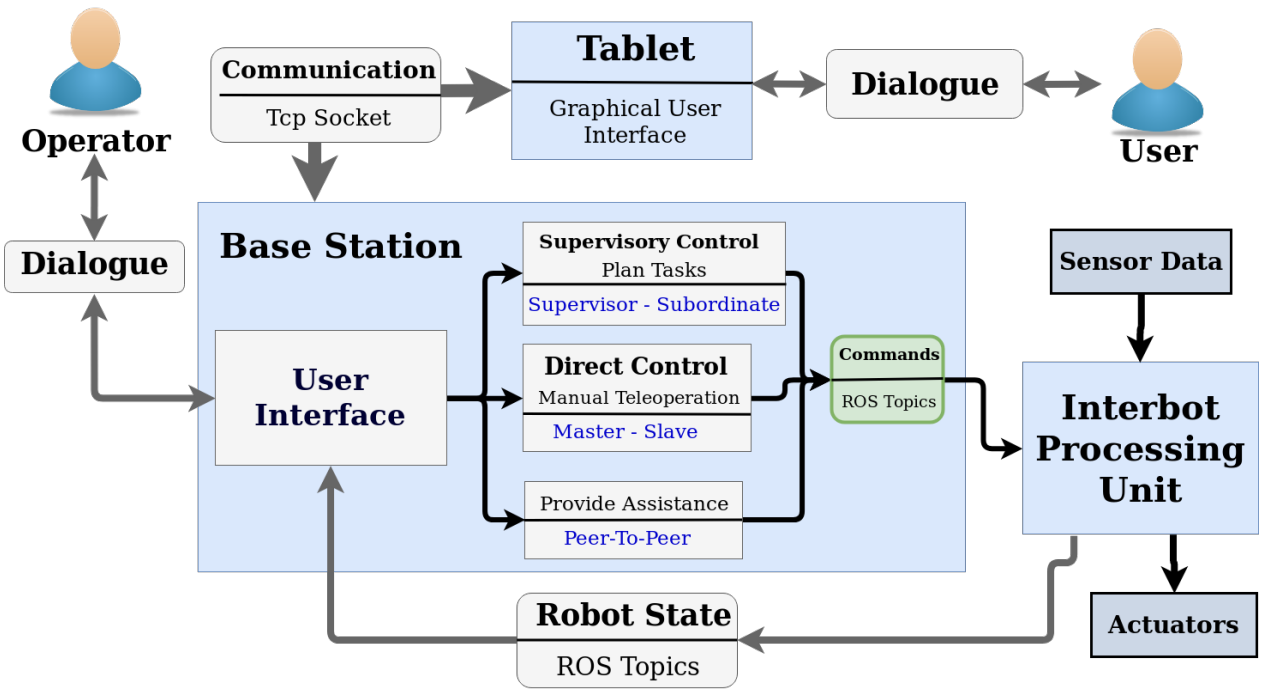


Figure 4.2: Overview of the Interbot software architecture along with types of human-robot interfaces, possible control types and HRI roles depending on the robot state.

4.2.1 Tablet

The Tablet works as a Human-Robot Interface. It requires to be connected to the Base Station through a TCP socket and runs the necessary software for visualization and different user input options, as depicted in Fig. 4.3. The developed software is compatible with any device using an Android operating system.

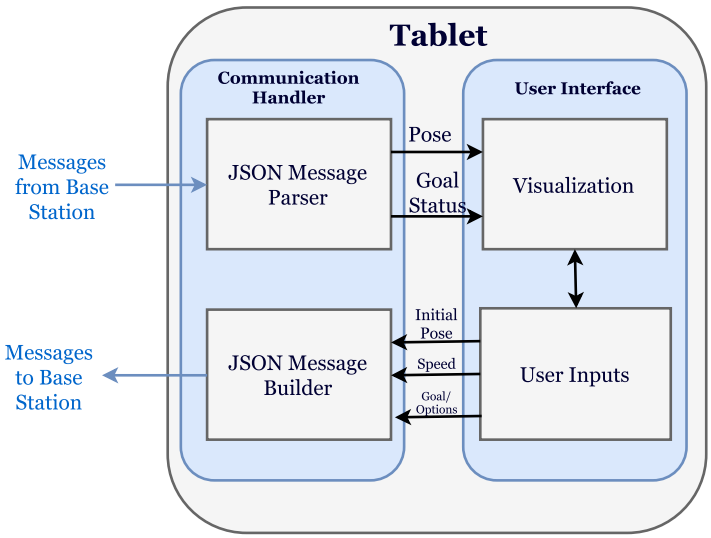


Figure 4.3: Overview of the Tablet software architecture, inputs and outputs

4.2.1.1 User Interface

The Tablet User Interface is simple and easy to understand, so that any user can be able to order tasks to Interbot and quickly perceive its state. In the main menu, the connection status to the Base Station is displayed as well as the average latency in the communication. The user can choose to take control of the robot using a virtual joystick with adjustable speeds or open the map interface. An interactive map of ISR's floor 0 is displayed along with different interest points, including the current position and orientation of Interbot.
If no initial pose has been set and the robot was not capable of localizing itself, Interbot will request the user to move it to one of the interest points and indicate the initial position and orientation. After localization is complete, the user can assign different tasks to Interbot such as move to one of the offices/laboratories or perform a guided tour. At any point during one of these operations, the user can request to stop/resume autonomous navigation, take manual control over Interbot's movement or even report a malfunction such as a collision, wrong position/orientation.

4.2.1.2 Communication Handler

Communication done between the Base Station and the Tablet happens through a TCP socket and the transmitted data is in JSON format. The incoming messages from the socket are parsed, and the information to send to the Base Station is built into JSON format before being sent through the socket.

4.2.2 Base Station

The Base Station works as a remote control station, connected to the Processing Unit and the Tablet, and runs the software required for visualization and control of Interbot's actions. Figure 4.4 shows the software architecture of the Base Station.

4.2.2.1 User Interface

For visualization purposes the Base Station uses RVIZ, which is a tool provided by ROS to visualize available topics published by nodes running in the Processing Unit such as laser scans, maps and 2D pose. It also runs an User Interface node that shows current speeds, orientation of the robot and status messages such as: the task being performed by the robot, error messages and distance required to travel to the next goal.
The User Interface presented in the Base Station must be intuitive so that with little effort of training, any user can use it in an efficient way. The user can directly take command of the robot's movement using either keyboard keys or a virtual joystick with adjustable velocity settings. If navigation is currently occurring, it can be paused and resumed. The user can also give goals to the robot using a target point of the map, or through a list of pre-defined goals. Since Interbot requires an initial pose to be defined, it can also be set through this node.

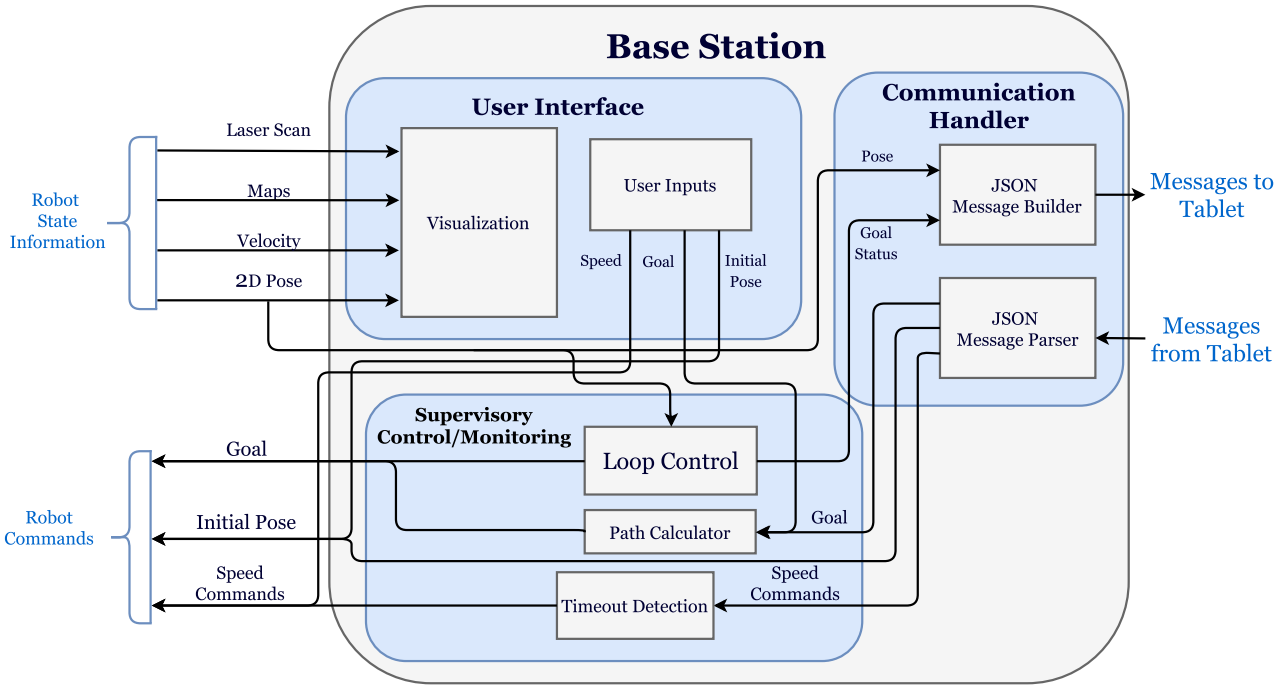


Figure 4.4: Overview of the Base Station software architecture, outputs produced and received inputs.

4.2.2.2 Supervisory Control and Monitoring

Every time Interbot is asked to perform some task and move to a destination goal, the Base Station calculates the shortest path from the current position through the topological map nodes, and continuously sends the list of intermediate nodes to the Processing Unit, until the destination is reached. The Base Station is constantly requesting the current pose/orientation from the robot, so that when a new goal is given it can correctly calculate the shortest path, or whenever the robot is close to one of the intermediate nodes it can send the next goal to the Processing Unit. If a connection to the Tablet is established, it also checks the latency in the communication and takes control over the Tablet speed commands whenever the latency is above the desired.

4.2.2.3 Communication Handler

Communication done between the Base Station and the Tablet occurs through a TCP socket, and the transmitted data is in JSON format. The incoming messages from the socket are decoded, and the messages to the Tablet are built into JSON format before being sent through the socket.

4.2.3 Processing Unit

The Processing Unit located in the Interbot platform runs the necessary software for mapping, localization, path-planning and velocity multiplexing. Its software architecture can be seen below in Figure 4.5

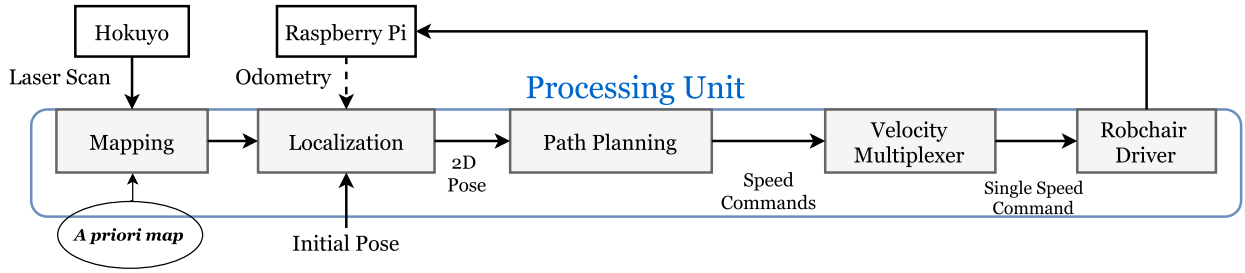


Figure 4.5: Processing Unit software architecture, displaying the main nodes and their outputs/inputs.

4.2.3.1 Mapping and Localization

Using the collected range data from a Hokuyo and an a priori map of the environment, the Interbot is capable of self-localization by using a modified version of the Hector SLAM algorithm. However, since it is incapable of obtaining the initial position by itself, it requires that an initial pose is given by the user. The Hector SLAM algorithm generates the robot's 2D pose, as well as an updated map of the environment, that is used to calculate the costmaps required by the Path Planning module. The odometry information computed in the Raspberry Pi is not used in the Hector SLAM algorithm, however it is available if some other type of localization method is to be used. Before this work started, the odometry computation was not reliable and could not be used on a localization algorithm, due to incorrect settings regarding the encoder parameters. There was an effort to correct the odometry calculation, and these corrections were tested using teleoperation test scenarios.

4.2.3.2 Path Planning

After obtaining the current pose, an updated map of the environment and the laser scan information, this module calculates the required speed commands necessary for Interbot to reach the intended goal. The used Path Planning node, herein designated `hybrid_motion_planner` [24], was developed in ISR-UC and is briefly described in Section 3.3. The outputted speeds allow for a safe navigation of the robot in the environment, actively avoiding dynamic or static obstacles. Configuration parameters can be set to change the maximum and minimum linear/angular velocities and the tolerance distance/orientation towards the target goal.

4.2.3.3 Velocity Multiplexer

The Velocity Multiplexer is a mechanism capable of obtaining all the speed commands sent by either the Path Planning module or the Base Station, and multiplexes them using a priority based scheme. This mechanism allows for a shared, manual or autonomous control of Interbot's movement. The final speed command message is then sent to the Raspberry Pi. The way this method was implemented is explained in the following chapter.

4.2.3.4 Robchair Driver

The Robchair Driver acts as the bridge between the Raspberry Pi and the Processing Unit, by rerouting speed commands published by the Speed Multiplexer to the Raspberry Pi.

Chapter 5

Interbot: Software Design

This chapter will present how the software for the two new components was designed and implemented, as well as the reasoning behind the proposed solutions. Different types of diagrams such as sequence, class, and state, are used to model and explain the structure and behavior of functions, modules and connections.

5.1 Requirements

In order to meet the software design requirements specified in the Introduction, it was necessary to make appropriate decisions regarding which programming languages, middleware and design frameworks, communication protocols and interface design toolkits.

1. **ROS**: This middlware is used for most of the software implementations running in the Processing Unit, therefore it appeared to be appropriate to continue using it in Interbot. Additionally there is a great amount of available documentation, frequently updated modules and the publisher/subscriber mechanism is suitable for a remote control station. The Master node is running on the Processing Unit along with the other required nodes detailed in Section 4.2.3. The Base Station contains remaining remote nodes that are connected to the Master.

2. **Qt Framework**: Partially meets all the demands, since it is a well-structured framework with extended libraries, cross-platform development and modules in the areas of networking, communication protocols, GUI-application toolkits, JSON parsing/decoding and even ROS integration. It was the programming framework used to write, debug and deploy all the implemented modules in the Base Station and Tablet.

3. **JSON Messages**: Since the communication between Interbot's Base Station and Tablet needs to have multiple data types (number, string, boolean, etc) it was decided to use a serialization process to translate multiple data types into a format that can be stored and reconstructed later.

4. **TCP Connection**: Provides a reliable transport of data, since it is suitable to be used on continuous streams of data, allowing the messages from the Base Station to the Tablet to always be delivered, even in the event of a long delay in the communication.

5.2 Tablet

The Tablet acts as an human-robot interface with a great level of accessibility, containing modules for visualization of Interbot pose and status, and allows for direct, supervisory and peer-to-peer control. All the code developed for the Tablet was written in C++ language. Qt Creator contains an Android tools library, which allows to compile and distribute the application to any Android device. Figure 5.1 shows the main signals and slots connections implemented in the Tablet.

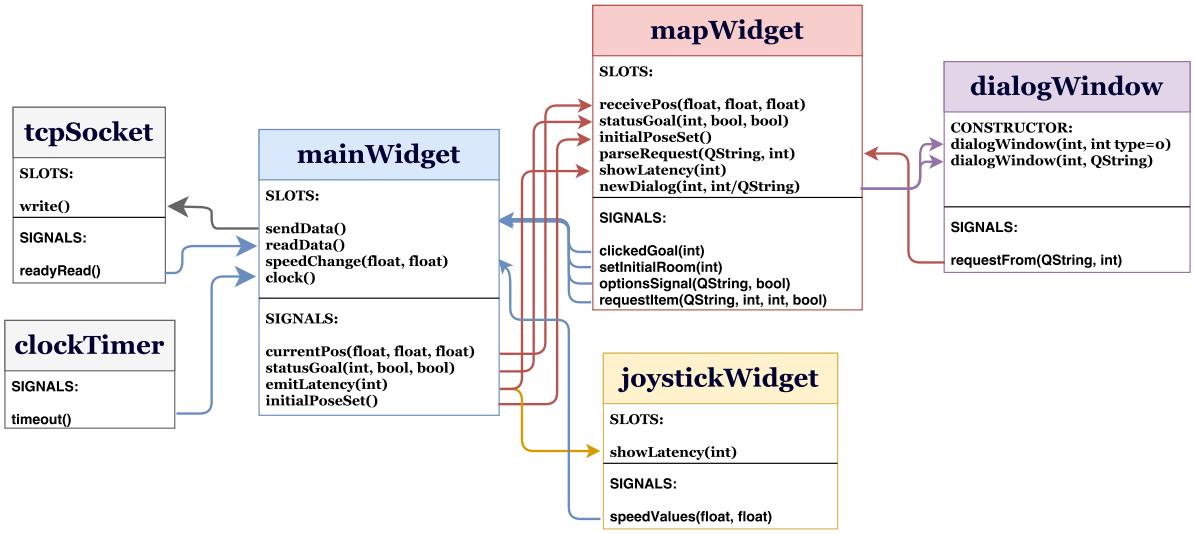


Figure 5.1: Signals and Slots connections of the Tablet. Arrow color indicates the corresponding receiving object of a signal.

5.2.1 Visualization

Tablet contains the `mapwidget` object which displays an interactive map of ISR floor 0, containing points of interest with information (see Fig.3 from Appendix I). If Interbot's current position is known, then it will be displayed in the map along with its orientation. The user has the possibility of ordering the Interbot to guide him to any of the displayed rooms, to perform a guided tour of the whole floor, or to perform a task such as transporting an object between rooms.

5.2.1.1 Map Features - Pose / Points of Interest

Every time new data arrives through the `tcpSocket` containing pose information, a signal `currentPos` is emitted by the `mainWidget` and is received by the `mapWidget` object. This pose information comes in ROS coordinates, so a transformation must be applied together with the correct scaling. Figure 5.2 and Algorithm 3 display details about the transformation process. There is also a small offset on the y-axis that must be accounted for, due to the way the *a priori* maps of ISR floor 0 were obtained, so that the previous topological map could be utilized. On this interactive map, all rooms that belong to the topological map are labeled, and their surface area is directly linked to a button.

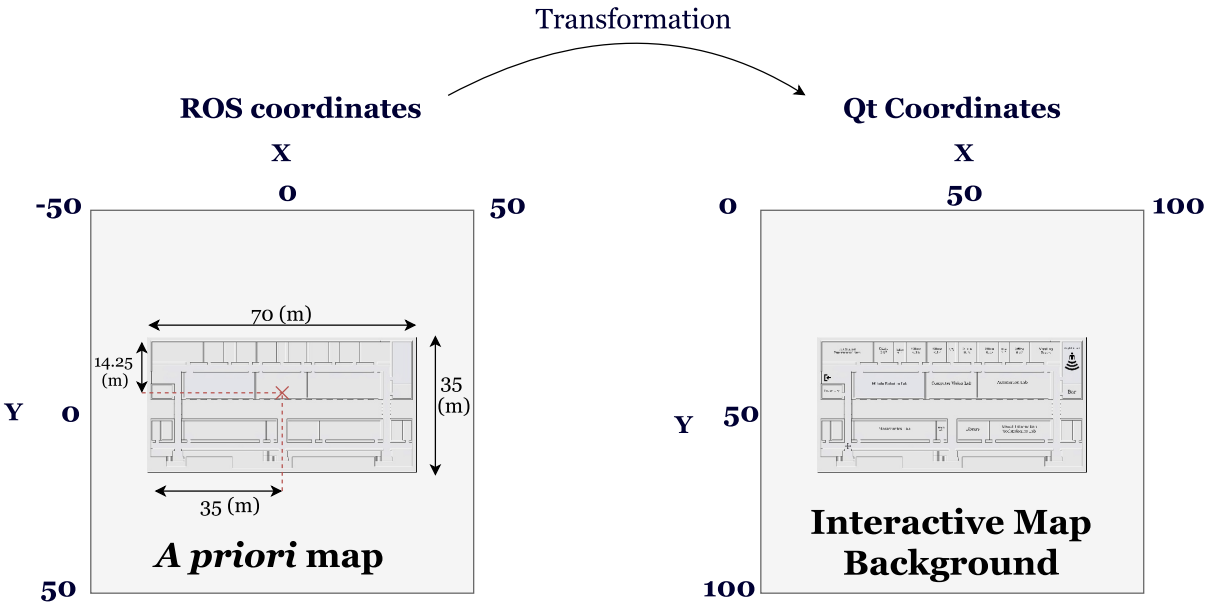


Figure 5.2: Map Transformation from ROS to Qt coordinates .

Horizontal and vertical scales are obtained by dividing the width and height of the `mapWidget` by the corresponding map size. Constant scaling of the objects is important, so that any Android device can be used without resolution compatibility issues.

Algorithm 3: receivePos(float x, float y, float a)

Input: $x, y, a \leftarrow$Current position in ROS coordinates

heigth, width $\leftarrow$ Pixel Size of the window

1 $h\_scale \leftarrow$ heigth/35

2 $w\_scale \leftarrow$ width/70

3 $pos\_Y \leftarrow (y + 14.25) * h\_scale$

4 $pos\_X \leftarrow (x + 35) * w\_scale$

5 $arc \leftarrow a$

6 $repaint()$

5.2.2 User Commands

When using the Tablet, there are a variety of actions that the user can perform. Figure 5.3 shows a graph depicting the possible flow of these actions. Most of them are simply done by pushing buttons that are displayed in the interface while others, like using the virtual joystick or inserting details about a request, require a little more elaborate set of inputs. There are three main courses of action an user can take:

- **Room Selection**: After selecting a room the user can order Interbot to move there, and during this process the user can pause, resume or cancel the navigation using on-screen buttons. The user can also request something from that room and choose the target for delivery of the request. The Interbot will navigate to inside the selected room, interact and display the request to whoever is in the room, and if there is a response to the request, the Interbot

will move to the destination to complete the task. Since most of the rooms in ISR floor 0 are typically locked or required card access, only a few rooms have this option, for now.

- **Request Guided Tour**: Upon requesting a guided tour, Interbot will move to the main entrance of every room, (and inside the rooms that allow for requests), pause navigation, display information about the room and wait for user input to resume or end the tour.
- **Assist with Position**: The user can assist the Interbot, by using the virtual joystick, to move towards one of the labeled rooms, and set the initial position.

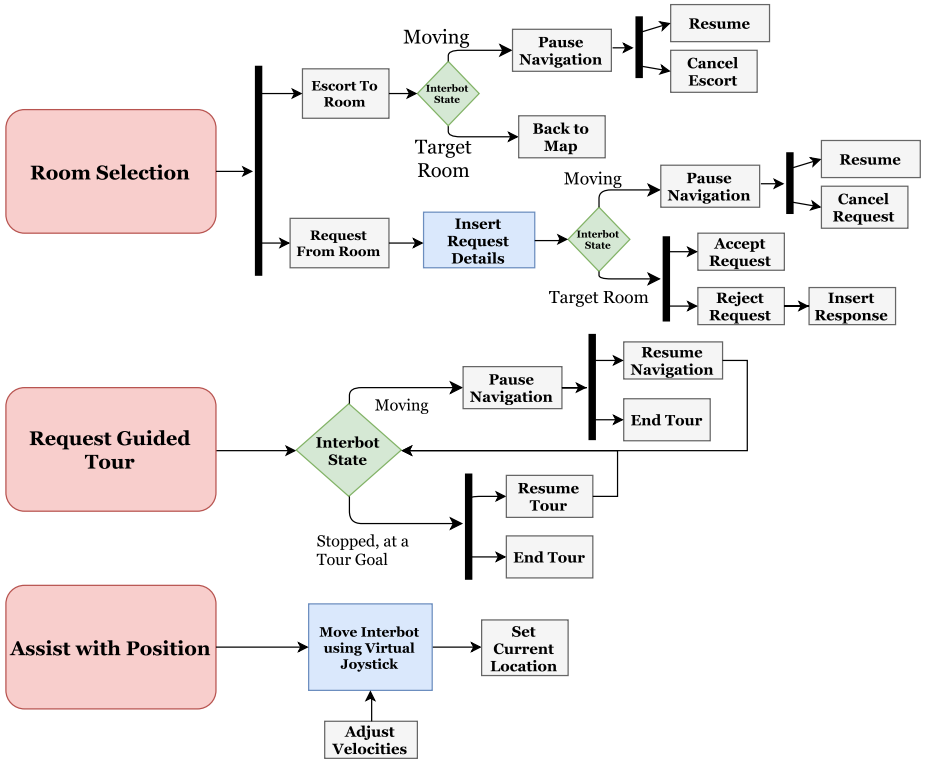


Figure 5.3: Graph showing the possible actions that a user can perform using the Tablet.

5.2.2.1 Assistance with Position

Interbot's mapping and localization modules require an approximate initial position to be set, since the current SLAM method is not capable of solving the initial global localization problem, before they can work properly and locate the robot in the surrounding environment. The most common approach is to set this initial pose using Rviz, which is running in the Base Station. In the rare occurrence of a localization loss, or if there is not someone currently operating the Base Station, the Interbot can not move autonomously until someone helps it to locate itself. In order to prevent these deadlocks until help arrives, as well as improve human-robot interaction and collaboration, a way to request assistance in setting the initial pose was implemented, as follows:

1. If the initial position is not set, or if localization has been lost, in the `mapWidget` interface, a request for help is shown and the user can click a "Help me!" button to give assistance.
2. The user is asked to move Interbot, using the virtual joystick, towards one of the highlighted rooms.

3. After Interbot has been moved close to one of the rooms, the user should set it as the initial location by clicking on the respective room, where a window like Figure 5.4 is displayed. This information is sent to the Base Station, initial pose will be set and Interbot will no longer request assistance regarding its position.

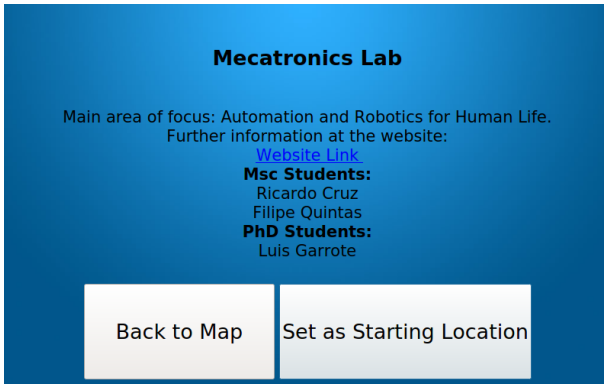
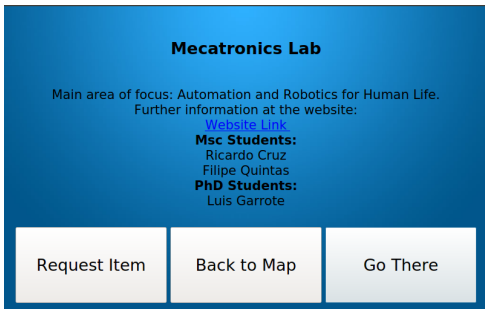


Figure 5.4: Assisting Interbot in setting the initial position

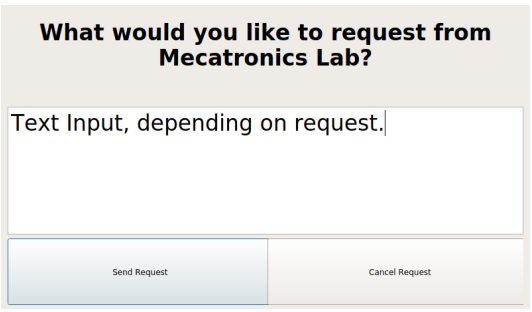
Human cognition and ability to operate the robot are key aspects to assist the robot correcting the initial pose. This is because Interbot trusts that the user will correctly move him close to one of the rooms and select the correct one as the initial location. Otherwise, Interbot will not be able to localize itself properly, and no further tasks will be able to be done until the pose is corrected, by either a supervisor in the Base Station, or by a more cooperative user using the Tablet.

5.2.2.2 Ordering tasks to Interbot

While using the interactive map, displayed by the `mapWidget` object, points of interest like laboratories, offices or utility rooms are shown. If a user touches the area containing these points of interest, a dialog window displaying information about that particular room is shown, seen in Fig. 5.5a. From this window, the user can ask Interbot to show the way to this room or, in a few cases, to deliver a message or request something from this room. In the event of a request/message, a new window is displayed where the user can type what he wishes to request and which room the request should be delivered to, seen in Fig. 5.5b



(a)



(b)

Figure 5.5: Dialogue windows presented in the Tablet UI for user input: (a) Window containing information about the room, and the available action options are presented to the user; (b) Window containing a text input, where the request can be specified.

In both types of tasks ordered by the user to Interbot, there is a corresponding room that is the target destination. Figure 5.6 shows the sequence diagram portraying the interaction between the components, when a manual goal task is given. A signal `clickedGoal` containing the goal number, corresponding to the room, is emitted and received by the `mainWidget`, that parses it into a JSON message and writes in the `tcpSocket`. For this to happen, a `QSignalMapper` mechanism was implemented, mapping each of the buttons representing points of interest to the corresponding node number in the topological map (see Fig.1 from Appendix I). This way when the Base Station receives this message, it will immediately know which room to calculate the shortest path to.

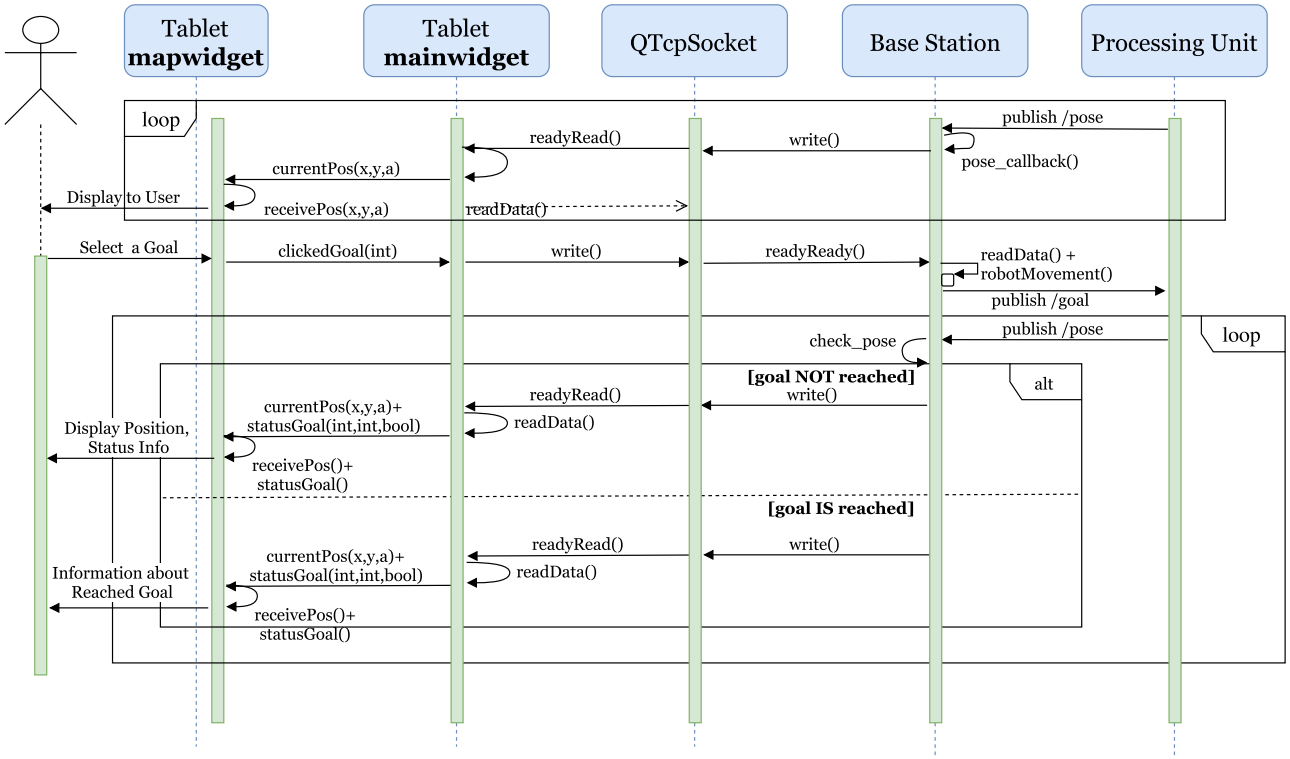


Figure 5.6: Sequence diagram representing the flow of information between objects and their processing sequence, whenever a new goal is given using the Tablet.

5.2.2.3 Virtual Joystick

To teleoperate Interbot using the Tablet, some sort of joystick implementation is required. Taking advantage of the Tablet's touch screen ease of use, and taking into account that Qt contains a dedicated class called `QTouchEvent` that emits information about touched points of objects in the UI, a virtual touch joystick was developed.

Figure 5.7 shows the window that is displayed in the Tablet. Two axis are used: one for linear speed and another for angular speed. The maximum values of each axis, and therefore of each speed type, can be changed using the respective slider. This way, every point inside the white circle corresponds to a specific linear and angular speed command, as explained in equations 5.1 and 5.2.

$$linear\_vel = \frac{(center_Y - pressed_Y)}{radius} * linear\_max \tag{5.1}$$

$$angular\_vel = \frac{(pressed_X - center_X)}{radius} * angular\_max \tag{5.2}$$

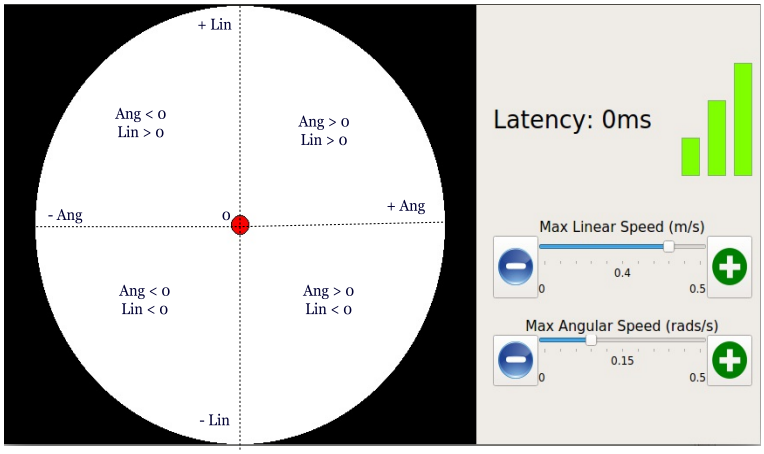


Figure 5.7: Virtual Joystick that can be used to control Interbot using the Tablet. Latency is displayed on the side, and maximum speeds limits can be changed using buttons.

When the user touches the white circle to move Interbot, the corresponding speed values are emitted through a signal, and caught by the `mainWidget` to be parsed into JSON messages and sent to the Base Station. Touching in the black area will send the speed command relative to the outer edge of the white circle, corresponding to the maximum linear and angular speeds for that angle. Finally, when the touch screen is released, the speeds are set to the zero.
It is important to notice that the latency in the connection to the Base Station is displayed, and in the event of a lossy connection the user is warned that the speed commands will not be processed by the Base Station until proper connection is re-established, as explained in Section 5.3.4 .

5.3 Base Station

As described in the previous chapter, the Base Station contains a variety of software modules that allow for remote monitoring and control of Interbot. It contains an intuitive User Interface, so that anyone can quickly understand and use it to its full extent (see Fig.2 from Appendix I). All the code developed for the Base Station was written in C++ language, using Qt Creator. The signals and slots mechanism, described before, was consistently used to perform the communication between UI objects and different functions. After being launched, the Base Station must perform several tasks: it creates the `QTcpServer` and listens for connections, connects the various signals from UI elements to different slots, starts the necessary timers for concurrency handling, creates its own ROS node with the required subscribers/publishers and finally loads all the nodes information to a circular buffer.

When using the Base Station UI, the user can perform the following control tasks regarding Interbot:

- **Teleoperation**- Interbot can be directly moved using the Base Station in different ways: Using the arrow keypad, keyboard shortcuts and with the virtual joystick. There are also buttons to pause/resume autonomous navigation, and an emergency stop.

- **Tablet Control**- If the Tablet is connected, the Base Station user can determine if the requests from the Tablet should be accepted or not, and which type of requests are allowed. If Interbot is currently performing a task requested through the Tablet, the user can directly superimpose this task and stop it or even cancel it.
- **Configurations**- Several parameters can be configured using the UI: the remaining distance tolerances to be considered when Interbot is approaching goals, maximum linear and angular speeds for teleoperation, latency ceiling on the connection with the Tablet.
- **Tasks** - In the same way as the Tablet UI, the Base Station UI can display an interactive map where the user can select rooms and give tasks, such as: move to a room, delivering a message between rooms, requesting an item from a room or perform a guided tour.

5.3.1 Connection with Tablet

A TCP connection must be established between the server and client before any data transfer can begin. At any time, one side can close the connection and data transfer will stop immediately. The `QTcpSocket` works asynchronously, emits signals to report status changes and contains two independent streams of data: one for reading and one for writing. Data can be written into the socket using `QTcpSocket::write()` and to be read using `QTcpSocket::read()`. Figure 5.8 displays the sequence diagram representing the connection process between the Base Station and Tablet.

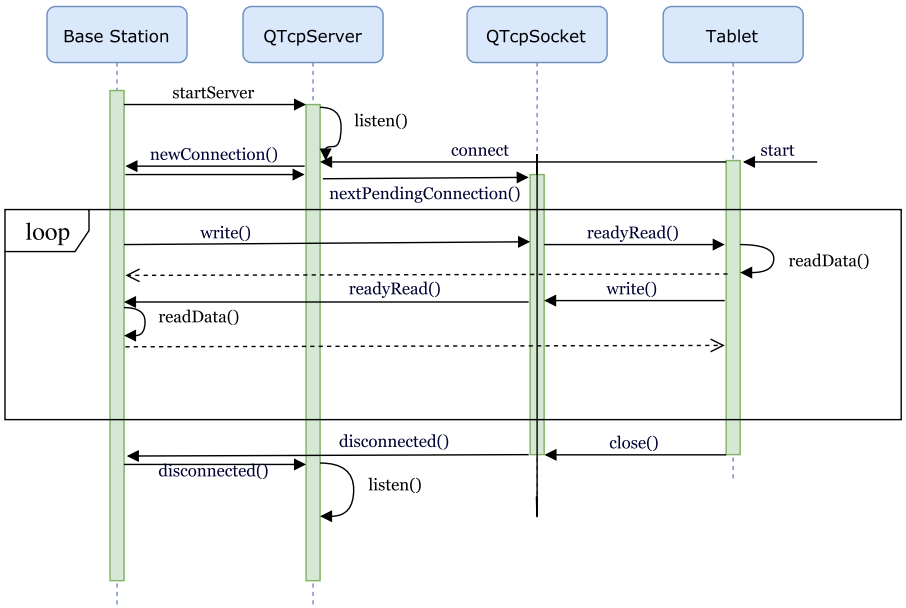


Figure 5.8: Sequence diagram displaying the connection process between the Tablet and the Base Station.

Interbot's Base Station works as the TCP Server using the `QTcpServer` class. On startup, it calls `QTcpServer::listen()` to start the server and connects the `QTcpServer::newConnection()` signal, which will be triggered whenever a new client connects, to the `newConnection()` function, which calls `QTcpServer::nextPendingConnection()` to accept the connection and return it as a connected `QTcpSocket` object for communication called `tcpSocket`.

After connection has been established, and `tcpSocket` is up and running, we connect the signal `readyRead()`, which is emitted every time a new cluster of data has arrived through the socket, to the function `readData()`. Whenever a new block of data arrives through the socket, it is immediately read and put into an array of bytes for further parsing, seeing as every message is in JSON format. If the Tablet is closed, the signal `closed()` is emitted and the QTcpSocket is destroyed, but not before emitting a `disconnected()` signal to the Base Station, so that it can restart the QTcpServer and listen for new incoming connections.

5.3.2 Messaging / Data

When transmitting data through a QTcpSocket, it is required for it to be stored in a byte string. Using the `QJson` class, the desired output information is built into a JSON message before being written into the `QTcpSocket`. When data is available in the `QTcpSocket`, the incoming JSON message is decoded and transformed into the respective data formats for further processing. Listings 5.1 and 5.2 display the different possible messages and the data types they contain.

5.1: Tablet JSON messages

```
{
"speed": {
    "linear": float,
    "angular" : float
    },
"goal": int ,
"pause": bool ,
"stop": bool,
"tour": bool ,
"home": true,
"requestFrom" : {
    "request": string,
    "room": int,
    "deliverTo": int,
    "status": bool
    },
}
```

5.2: Base Station JSON messages

```
{
"initial_pose" : bool,
"pose": {
    "x_pos": float,
    "y_pos" : float,
    "angle" : float
    },
"goal": {
    "goal": int,
    "reached": bool,
    "tourReached": bool
    },
}
```

The Tablet JSON messages has seven available message types, but only one of them is sent at each time. This is because if the user is controlling Interbot using the virtual joystick only `"speed"` type messages should be sent, and in other possible scenarios like defining a goal, or requesting to pause navigation, a fast request to the Base Station is desired so a boolean flag is used. Finally, if a user has selected the request task, the JSON message `"requestFrom"` has to contain the request details, target room and status of the request.

On the other hand, messages from the Base Station to the Tablet are normally more detailed: if Interbot's position is set, the Base Station constantly sends the pose information through the

socket and in the event that the robot is moving towards a goal then it must also inform the Tablet of which goal it is and whenever the destination is reached. When the initial pose is not set, only a simple boolean message `"initial_pose"` is sent.

5.3.3 Data Initialization

The topological map contains information regarding the position of each node and the correct orientation that have to continuously be used whenever a new task is ordered (see Fig 1 of Appendix I). Since the number of nodes is fixed, a data structure similar to a ring buffer is implemented as shown in Fig. 5.9.

As depicted in Algorithm 4, initially the data is written into an empty `ringBuffer` of size N (representing the number of nodes in the topological map), using the function `writeIntoBuffer`. This function simply loads the input information of a node and stores it in the current index of `ringBuffer`, incrementing this index before finishing. After this, the distance between every node is stored in a matrix `D` to be used by Algorithm 1 to calculate the shortest path between every possible node.

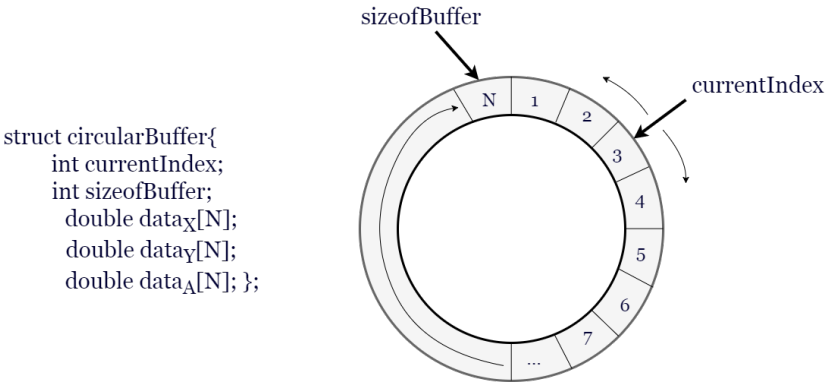


Figure 5.9: Ring Buffer structure used to store data. When storing in the buffer, on every iteration the data is saved and the current index indicator is incremented.

5.3.4 Watchdog Timer

Due to the fact that Interbot's communication is done over Wi-Fi, it can get congested or the signal at long ranges can be lossy, which leads to high latency values since floor 0 of ISR is over 50 meters long. The biggest possible threat scenario is when an user is controlling Interbot's movement using the Tablet virtual joystick: latency to the Base Station can be high or connection may be lost momentarily when the Tablet has to reconnect to a different Wi-Fi hotspot. In this scenario, the Base Station is still processing the last received speed commands from the Tablet and wrongly sends them to the Processing Unit until connection to the Tablet is re-established. In this scenario, to prevent any human or material damages, a watchdog timer is implemented in the Base Station.
Watchdog timers are used to protect the system from software failures in the occurrence of a time-out. In our case, it can be directly correlated to latency: when it is higher than the one defined in the watchdog timer, this means that something is wrong in the connection with the Tablet. In this case, if there were any speed commands coming from it, then they must be discarded.

Algorithm 4: initialData()

Data: A matrix $node[3][N]$ containing the position (X,Y) and orientation (radians) of all the N nodes in the topological map.
A matrix $D[N][N]$ containing distances between all nodes

```
1  RingBuffer ← empty circular buffer, with size N and starting index 0.
2  for i ← 0 to N do
3      writeIntoBuffer(RingBuffer, node[0][i], node[1][i], node[2][i])
4  for i ← 0 to N do
5      for j ← 0 to N do
6          if (j == i) or (j == i − 1) or (j == i + 1) then
7              D[i][j] ← distance between adjacent nodes.
8          else
9              if (i == 0 and j == N − 1) or (i == N − 1 and j == 0) then
10                 D[i][j] ← distance between adjacent nodes.
11             else
12                 D[i][j] ← value unchanged.
13 P ← floyd_algorithm(D).
```

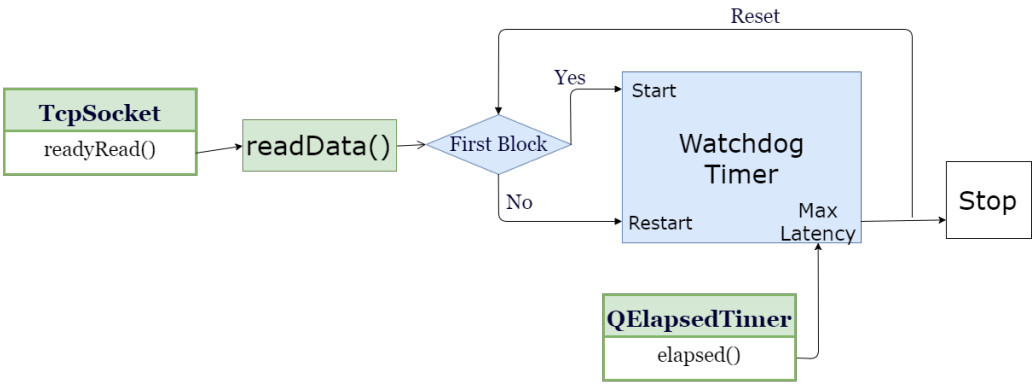


Figure 5.10: Watchdog Timer overview, containing used signals and slots for timeout control.

As seen in Figure 5.10, the timer starts when the first block of data regarding speed commands arrives through the socket. Every time a new chunk of data is received, the timer is restarted and the time elapsed is the latency in the connection between the Tablet and the Base Station. In case of a timeout due to excessive latency, the Base Station stops publishing speed commands from the Tablet to the Processing Unit.

5.3.5 Path Planning Algorithms

The previous version of Interbot included a path planning algorithm that allowed for the execution of long distance routes using intermediate goals along the way, depending on the initial pose and target location. Each of these intermediate goals would correspond to a position where the robot could perform rotations, and then continue to its destination. However, there was a drawback in the

algorithm: the robot would always find the closest node to choose as the initial point, and move to it before performing the path to the intended destination. This could be undesirable in situations where the closest node wouldn't exactly be the first node on the way to the destination, as seen in Fig. 5.11. Starting at marked initial position, if an order was given for Interbot to move to node 4, Interbot would first move to the closest node, node 5, before moving to node 4.

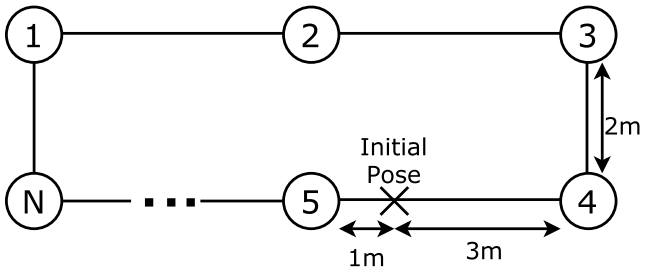


Figure 5.11: Simplified example of an application of the path-planning algorithm.

To solve this problem the following solution was adapted:

1. Obtain the distances to all nodes, finding the closest one.
2. Calculate distances to the possible paths leading out from the closest node and return the one closer to the initial position. This way, we know which is the other node that can be used as the starting node, if required.
3. Calculate the distance that it would take to reach the intended goal using the two possible starting nodes, and the shorter path of the two is the one selected.

Applying this method to the example of Figure 5.11, Interbot would directly move to goal 4, since moving to the closest node would not result in the shortest path.

This solution was implemented in the `robotMovement()` algorithm, described below in Algorithm 5. Whenever a new goal is selected, either through the Tablet or Base Station this function is called, the most efficient path is calculated and stored in the matrix SP, and the first intermediate goal is published. The control function `clock()`, which will be described in 5.3.6, will then constantly be calculating the remaining distances to the next goal, and whenever the following intermediate goal has to be set, it will increment the goal index and publish the goal stored in `SP[goal_index]`, until the final goal is reached.

In the event of a guided tour request, Interbot must visit all the nodes in the map, so an implementation of a circular buffer system was done, as follows:

1. Load all the information from the `RingBuffer`, starting at the current index which corresponds to the closest node, to the `GuidedBuffer`.
2. Publish the first goal using the first element of the `GuidedBuffer`.
3. Whenever a guided tour intermediate goal is reached, the function `clock()` increments the goal index and publishes the goal from `GuidedBuffer[goal_index]`.

Algorithm 5: robotMovement()

$N \leftarrow$ number of nodes in the topological map
$RingBuffer_{\{x,y,a\}}[0:N] \leftarrow$ matrix containing the position (x,y) and orientation of all the N nodes
$final\_node \leftarrow$ destination node
$pose\_x, pose\_y \leftarrow$ current position in ROS coordinates
$dist1 \leftarrow$ distance to final goal, moving to the closest node first
$dist2 \leftarrow$ distance to final goal, moving through the other possible starting node
if $dist1 < dist2$ **then**
    $initial\_node\_ \leftarrow$ initial node is the closest one
else
    $initial\_node\_ \leftarrow$ initial node is in the opposite direction
if !$guidedTour$ **then**
    $SP = shortest\_path(SP, P, initial\_node\_, final\_node\_)$
    $goal\_number = sizeof(SP)$
    $goal.pose \leftarrow SP[0]$
    publish($goal$)
else
    $insertIntoGuidedBuffer(RingBuffer, GuidedBuffer)$
    $goal\_number = N$
    $next\_node \leftarrow$ current index of RingBuffer
    $goal.pose \leftarrow guidedBuffer_{\{x,y,a\}}[0]$
    publish($goal$)
$toMove =$ true

5.3.6 Information Monitoring and Control

The Base Station must be constantly receiving, processing and monitoring information from the Processing Unit regarding Interbot's state and, whenever connected to, send the required information to the Tablet. For this, there is a control loop function called `clock()` that is launched whenever the defined QTimer `timeout()` signal is emitted.
Figure 5.12. displays the state diagram of the `clock()`, describing the behavior of the function. Inside this function, three main handling segments can be found regarding Interbot's speed, pose and communication with the Tablet, as follows:

- **Speed** - Obtains Interbot's current linear and angular speed and updates the speedometers in the UI. If there are any speed commands from the Base Station UI, either from the keyboard controller or through the mouse joystick, publishes the speed commands in the `station_vel` topic. In the same way, if there are incoming speed commands from the Tablet, and the user is allowed to control Interbot, then it publishes the corresponding speed commands in the `tablet_vel` topic.

- **Pose** - Obtains Interbot's current position and orientation and emits them through a signal, so that they can be captured by the UI object and displayed properly. If Interbot currently has a task to move to a goal, then it calculates the distance to the goal and processes it differently, depending on the scenario, as follows:

  - **Guided Tour** - While performing a guided tour, every node must be visited and Interbot must be at a short distance and in the correct orientation before stopping to inform the user about this goal. When these conditions are satisfied, it pauses navigation and waits for the user to resume or stop the guided tour. Upon resuming, it publishes the new goal and moves to it.
  - **Single Goals** - When a single goal is selected, then the shortest path towards a goal is calculated, using Algorithm 5. It can contain many intermediate goals to make navigation easier. However, moving close to and getting the correct orientation in every intermediate goal is not effective in this scenario. Therefore, an approaching distance is defined below a certain tolerance. Whenever Interbot is under this tolerance it checks if it is the final goal and two possible courses of action occur: (a) It is not the final goal, and the next intermediate goal is published; or (b) It is the final goal, so Interbot will move closer to it and reach the correct orientation before coming to a halt.

- **Communication** - If a connection to the Tablet is established through the `tcpSocket` and Interbot's initial position is not set, it sends that information through the socket until an initial position is set, either through the Tablet or Base Station. After Interbot is localized, it constantly sends the current position and orientation through the socket to allow visualization in the Tablet UI, informs it if Interbot is moving towards a goal, and whenever a goal is reached through boolean flags, as explained in Section 5.3.2.

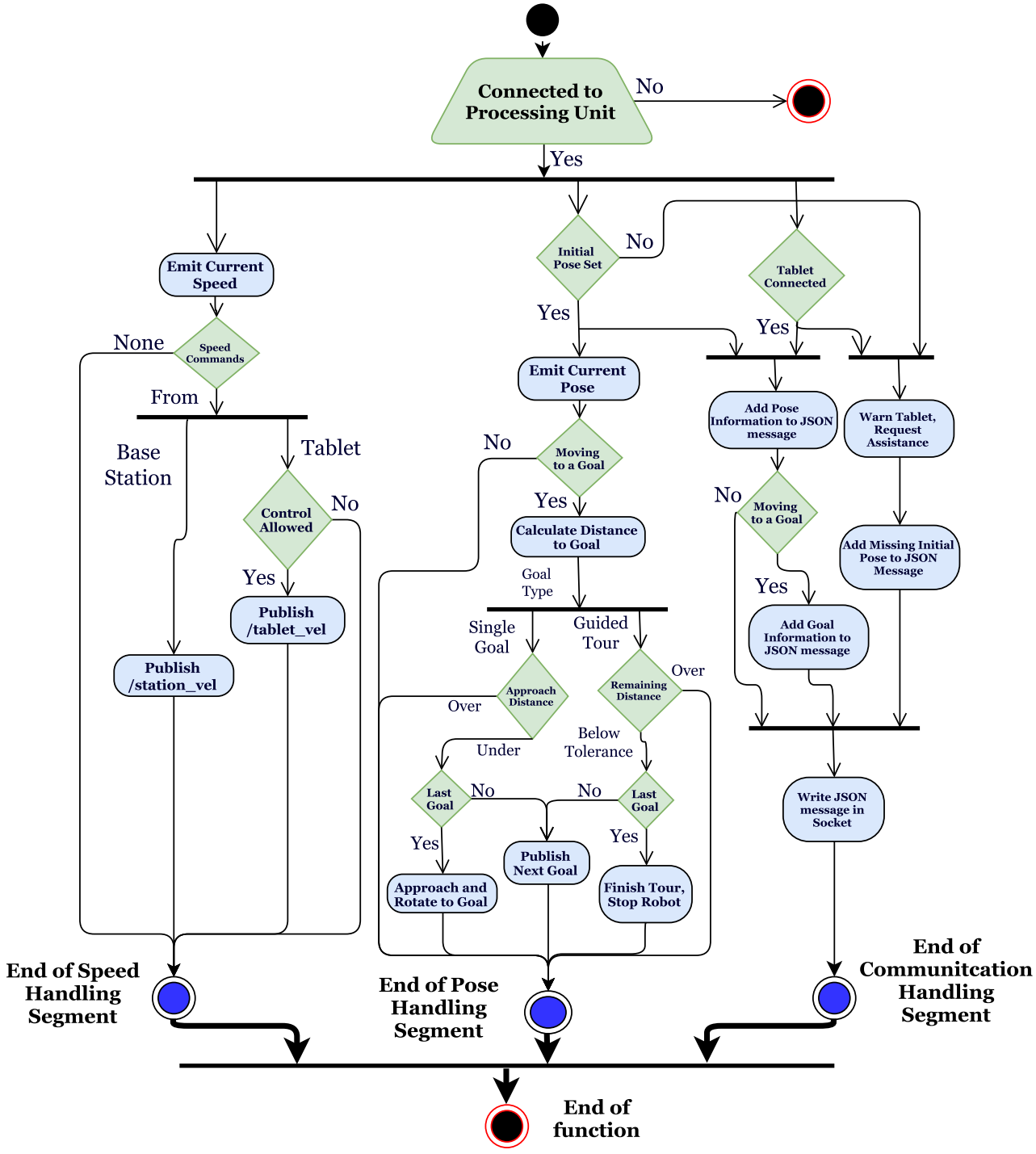


Figure 5.12: State Diagram of the `clock()` function, describing the behavior of the three main handling segments. Actions are represented by blue rounded squares, and decisions by green diamonds. After all three handling segments are finished, the function ends.

5.4 Processing Unit

As depicted in Section 4.2.3, the Processing Unit runs several ROS nodes for mapping, localization and path planning that were already implemented in the previous version of Interbot. In this work, the Robchair Driver received a kernel update with new configurations, and there was an addition of a Speed Multiplexer module.

5.4.1 Physical Layer

Figure 5.13 represents the Physical Layer module. It is present in the Processing Unit and performs the bridge between low-level hardware and respective software drivers. The `hokuyo_node` is a software driver provided by the ROS community, to be used with an Hokuyo laser range finder, publishing a /scan topic message of the obtained laser scanner data . The `robchair_driver` node was developed for the ISR intelligent wheelchair Robchair, being compatible with Interbot, since its low level platform architecture is very similar. It receives the /cmd_vel topic containing speed messages, sending it to the low-level part of Interbot. It also receives odometry data from the encoders, however this data was not used. The `usb_cam` node is provided by the ROS community [1] and allows to publish an `/image` topic containing the video stream from an USB camera. The `twist_multiplexer` will be presented in Section 5.4.1.1.

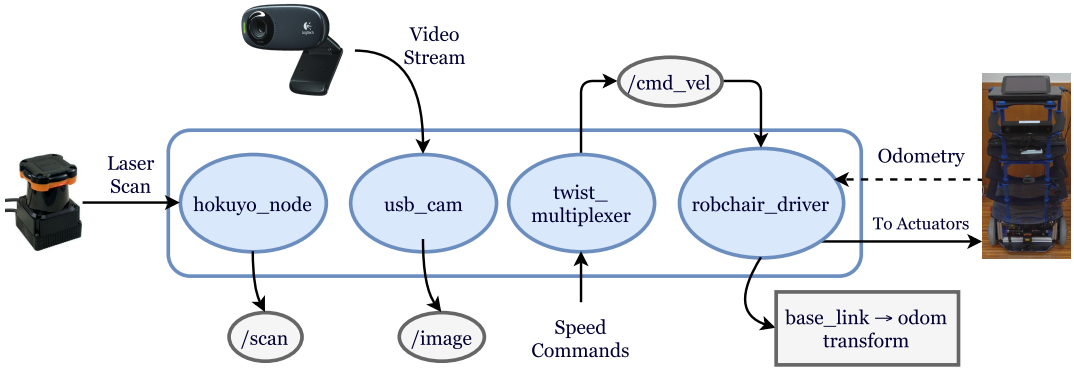


Figure 5.13: Physical Layer nodes, along with the subscribed and published topics.

5.4.1.1 Speed Multiplexer

`twist_multiplexer` is a package provided by the ROS community [2], which allows to multiplex several velocity commands at once (topics) and also to prioritize or disable them (locks). Since there are multiple implemented ways to control Interbot: through the Tablet on board, the remote base station or the path planner commands, it is important to multiplex all these input sources into a single final command message that is sent to the low-level part of Interbot. Figure 5.14 shows the topics and locks that are subscribed by the `twist_multiplexer` node.

The `twist_multiplexer` node subscribes to all the input topics, multiplexes them using a priority-based scheme. There are also lock topics that can be configured to control the input topics.

[1] http://wiki.ros.org/usb_cam
[2] http://wiki.ros.org/twist_mux

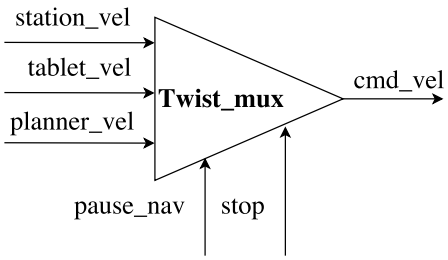


Figure 5.14: Twist_multiplexer node, along with the subscribed topics, locks and the published topics.

Several parameters can be configured, as follows:

- **topic** : Input topic/lock name that the node will subscribe to.
- **time-out** : If no messages arrive after the time-out period, this topic is disregarded. Very important since some of the speed messages are sent from the Base Station, and in the occurrence of a network delay, old messages would be correctly ignored.
- **priority** : The higher this value, the more priority it has over other topics. For locks configuration, if the lock is activated then it will ignore all the input topics that have a lower priority value.

The base station has maximum priority (`station_vel`), followed by the tablet (`tablet_vel`), and finally the autonomous navigation provided by the planner node (`planner_vel`). Regarding locks, one is used to pause the navigation whenever requested, while the other acts as an emergency stop that ignores all input topics and sets the robot to a halt. The following listings 5.3 and 5.4 show the configurations for the input topics and locks, respectively, that were used for most of the Test Scenarios in Chapter 6.

5.3: Input topics to multiplex

```
topics:
-
  topic: station_vel
  timeout: 0.5
  priority: 100
-
  topic: tablet_vel
  timeout: 0.5
  priority: 50
-
  topic: planner_vel
  timeout: 0.05
  priority: 10
```

5.4: Locks to control the input topics

```
locks:
-
  name: Pause Navigation
  topic: pause_nav
  timeout: 0.0
(not used for locks)
  priority: 25
-
  name: Emergency Stop
  topic: stop
  timeout: 0.0
(not used for locks)
  priority: 255
```

Chapter 6

Experimental Results

This chapter is dedicated to present the main experimental results obtained from different Test Scenarios. To experiment and evaluate he different types of control that can be applied in Interbot system, as mentioned in Section 4.2 and Fig.4.2, three test scenarios, inside of ISR floor 0, were designed in order to evaluate the system performance and capability of user interaction. Finally, the topic of existing latency in the communications, how it affected the results and the implemented measures against it are discussed.

6.1 Test Scenario 1

In the first test scenario, Interbot navigates a small circuit, as depicted in Fig. 6.1 receiving speed commands from directly the Processing Unit. In this scenario, Interbot moves along the corridor, performs two counter clockwise rotations, moves back to the starting point and finished with two clockwise rotations.
The purpose of this test scenario is to verify if the speed commands transmitted to the lower part of Interbot are converted into the correct voltage and current outputs to control the wheel motors. Even though the computed odometry resulting from this operation is not necessary for the Hector SLAM method, and the scan matching process can safeguard localization even if the values of odometry are not correct, it is in the best interest for an efficient teleoperation that the robot moves at a speed approximated to what it was meant to.

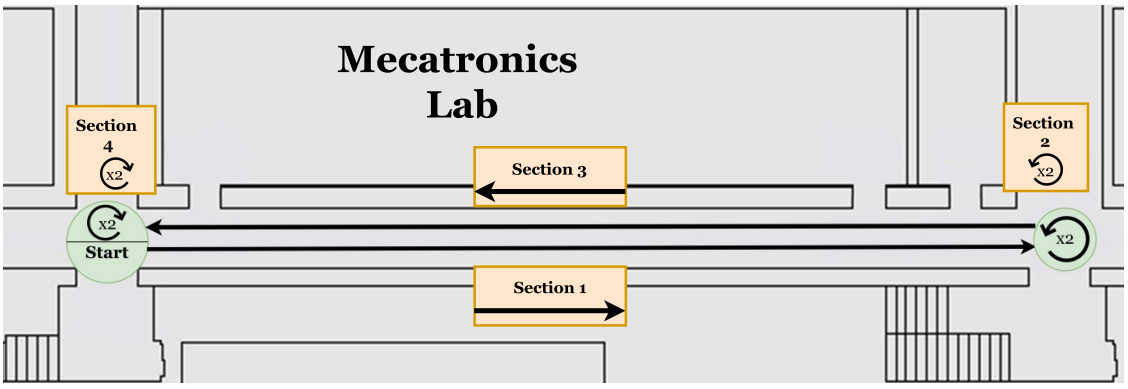


Figure 6.1: Experimental circuit used for Test Scenario 1. Sections 1 and 3 correspond to the linear corridors, while Sections 2 and 4 to places where the robot performs pure rotations.

6.1.1 Calibrating Odometry

To calibrate odometry only the Processing Unit was used. The Base Station and Tablet are disconnected. The speed commands are given on the same computer, and for that reason there is no latency involved. Therefore, none of the speed commands will exceed the timeout period defined in the Velocity Multiplexer and are all sent in time to the low-level part of Interbot.

6.1.1.1 Uncalibrated Tests

Table 6.2 shows the results of the experimental tests performed by Interbot in trajectory mentioned in Fig. 6.1. The expected duration values are obtained from Equations (6.1) and (6.2). Values from Sections 1 and 3 are averaged, as well as the values for the two rotational Sections 2 and 4.

$$ExpectedDuration(Corridor) = \frac{Distance}{SpeedCommands} \tag{6.1}$$

$$ExpectedDuration(PureRotations) = \frac{2\pi}{SpeedCommands} \tag{6.2}$$

Table 6.1: Metrics obtained for Scenario 1.

Test #	**Sections (Average)**	**Speed Commands**	**Distance (m)**	**Expected Duration** (s)	**Measured Duration** (s)	**Error Rate (Absolute)**(%)
1	1 and 3	0.6 m/s	26	43.333	40	7.690%
2	2 and 4	0.3 m/s	26	86.667	79.262	8.544%
1	1 and 3	0.6 rads/s	3.06	10.472	9.56	8.709%
2	2 and 4	0.6 rads/s	3.06	10.472	9.606	8.270%

The previous values show that both the linear and angular velocities were slightly faster than expected. In an attempt to correct these values, the configurations of the kinematic parameters were tuned, the kernel was updated and a new set of tests was performed.

6.1.1.2 Calibration Results

The results after calibration was performed are shown in Table 6.2. There is a noticeable decrease in both the linear velocity and angular error rate. There are factors that lead to a small amount of existing error rate, such as sensor data uncertainty, the physical condition of the wheels, motors, encoders, or even outside elements like floor friction and small tile bumps.

Table 6.2: Metrics obtained for Scenario 1.

Test #	**Sections (Average)**	**Speed Commands**	**Distance (m)**	**Expected Duration** (s)	**Measured Duration** (s)	**Error Rate (Absolute)**(%)
1	1 and 3	0.42 m/s	26	61.9	63.533	2.638%
2	2 and 4	0.30 m/s	26	86.667	89.788	3.601%
1	1 and 3	0.6 rads/s	3.06	10.472	10.76	2.750%
2	2 and 4	0.6 rads/s	3.06	10.472	10.73	2.464%

6.2 Test Scenario 2: Guided Tour

For the second test scenario, Interbot was planned to perform a guided tour around the floor 0 of ISR, while autonomously navigating towards every intermediate goal associated with rooms.
While the guided tour was in progress, visited rooms would be highlighted as green on the interface, and the next target in red, along with the current pose as shown in Figure 6.2a. Whenever one of the goals was reached, Interbot would pause navigation, show information about the room in the Tablet using a dialogue window, as depicted in Figure 6.2b , and wait for the user to either resume or end the tour.
Table 6.3 shows the configuration parameters set to be used by the path planner module of navigation, as well as the goal tolerances that the Base Station controlled before pausing navigation at each intermediate goal. Looking at the results in Table 6.4 and Fig.6.3 , it can be seen that Interbot can reliably perform guided tours in floor 0 of ISR, without having localization or collision issues, while visiting every intermediate goal corresponding to a room. Only two rooms, ISR Shared Experimental Area and Mecatronics Lab, were visited inside, while every other intermediate goal was set for the main entrance of every room/laboratory.

Table 6.3: Configuration parameters used for autonomous navigation

Maximum Speed		**Minimum Speed**		**Goal Tolerances**	
Linear (m/s)	Rotational (rad/s)	Linear (m/s)	Rotational (rad/s)	Linear (m)	Rotational (rad)
0.25	1	0	-1	0.5	0.1

Table 6.4: Metrics obtained for Scenario 2.

Test #	**Duration** (mm:ss)	**Success**	**Distance Traveled(m)**	**Collisions** (#)	**Localization Losses** (#)	**Mean linear speed** (m/s)
1	19:50	YES	238.55	0	0	0.1215
2	20:40	YES	231.525	0	0	0.1292

(a) (b)

Figure 6.2: Different moments of a guided tour: (a) Guided Tour in progress, previous visited rooms are highlighted as green, and the next one as red; (b) Upon reaching a goal, a window with information is displayed, and options to resume or finish the tour.

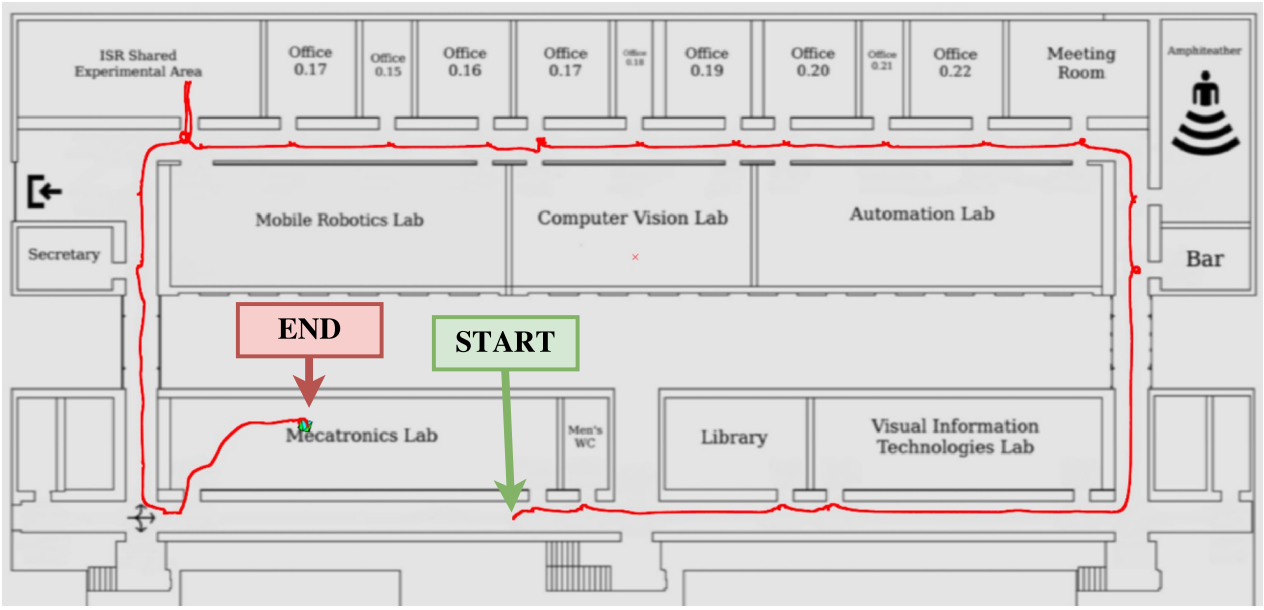


Figure 6.3: Trajectory performed during test number 2 of the guided tour Test Scenario.

6.3 Test Scenario 3

In the previous test scenarios, it was shown that Interbot's system is capable of performing autonomous navigation during long courses while giving a guided tour with user interaction. There is also the possibility to teleoperate Interbot using the Base Station. Therefore, it seems like a good idea to compare both approaches towards the completion of the same specific task, so that conclusions can be extracted from the advantages and drawbacks of each approach.
For this test scenario, Interbot has the mission of moving from a starting point, towards inside the Mecatronics Lab for the completion of a task given through the Tablet, leave the room and move towards the Bar, which corresponds to the final task destination as well as the end of the mission. During this mission, there are several intermediate points where the robot must pass and have an approximate correct orientation. Times are obtained at these points and are used to calculate the performance of each approach in the different sections. Figure 6.4 shows the experimental circuit used for this test, where the main and intermediate points are represented, as well as the expected orientation at each point.

Table 6.5: Metrics obtained for Scenario 3, regarding localization and obstacle detection/ avoidance.

Test Type #	**Duration** (mm:ss)	**Collisions** (#)	**Localization Losses** (#)	**Minimum Clearance** (cm)			**Mean linear speed** (m/s)
				min	**max**	**mean**	
Teleoperation	08:19	0	0	15.2	177.7	70.327	0.2739
Autonomous	09:46	0	0	25.5	167.9	82.95	0.1691

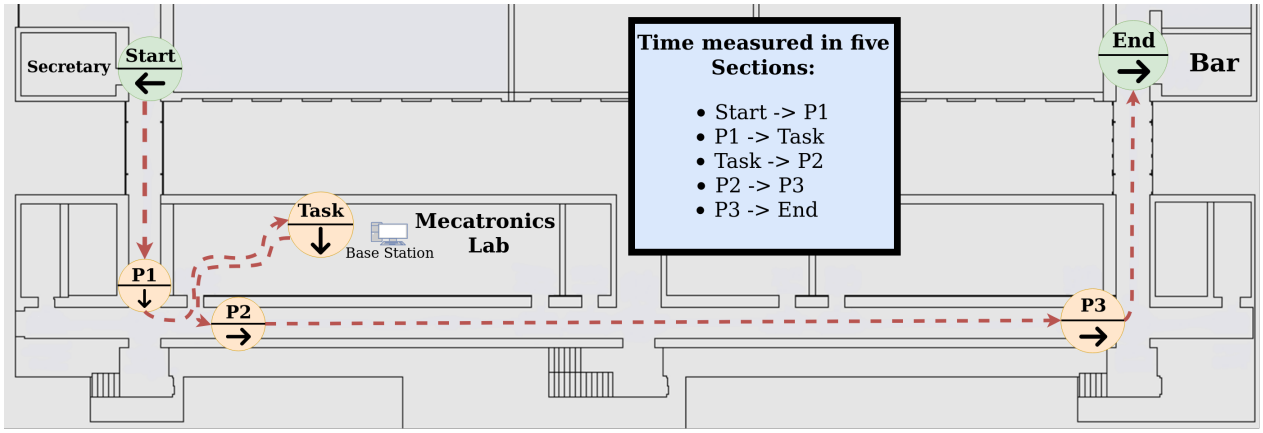


Figure 6.4: Experimental circuit for Test Scenario 2. Five important sections are separated by checkpoints.

Table 6.6: Metrics obtained for Scenario 3, regarding the time it took to move between each marked checkpoint.

Test Type #	**Time between checkpoints** (s)				
	Start -> P1	**P1 -> Task**	**Task -> P2**	**P2 -> P3**	**P3 -> End**
Teleoperation	27.892	135.692	88.74	149.868	97
Autonomous	80.48	71.423	62.505	291.648	80

6.3.1 Teleoperation and Autonomous Navigation

The results of Test Scenario 3 are presented at Table 6.5 and Table 6.6. It can be seen that teleoperating the robot using the Base Station is slightly faster overall. This is mainly because of the travel distances along the corridors, were the autonomous navigation does not go above 0.2 m/s, while the teleoperation interface allows to move at upwards of 0.6 m/s.

However, there are drawbacks when using teleoperation as mentioned in Section 2.1.1.1, due to the existence of many factors that directly affect performance, such as: limited field of view, depth perception, frame rate of the video stream, motion sickness and time delays due to the latency in the connection. Most of these factors are very noticeable in areas where the robot has to constantly adjust speeds due to the existence of many obstacles, when moving through a doorway or having to turn the robot towards the correct orientation without overshooting.

During autonomous navigation, none of the issues that affect teleoperation exist, and the robot quickly adjusts its speed to move through doorways, efficiently executes corners and turns towards the correct orientation at a gradual decreasing rate. Given the modularity of the system, since the navigation is done on the Processing Unit, latency is not a factor. In the event of a momentary loss of connection with the Base Station, since the goal was previously defined, the robot continues moving towards the destination.

6.3.2 Shared Control

After observing the previous architectures results, it seems like a good idea to combine the best of both approaches, consisting in using teleoperation during the long corridor stretches, while allowing for autonomous navigation to take over the robot's movement during corners, doorways and inside the laboratory. Table 6.7 shows the results with the same metrics as before, so that the system performance can be compared. Figure 6.4 shows the trajectory performed by Interbot, along with the different sections where teleoperation and autonomous navigation were used, as well as a few points of remark for discussion.

Table 6.7: Metrics obtained for Scenario 3, using Shared Control.

Test Type	**Duration** (mm:ss)	**Collisions** (#)	**Localization Losses** (#)	**Minimum Clearance** (cm)			**Mean linear speed** (m/s)
				min	**max**	**mean**	
	06:11	0	0	22.1	165.1	77.54	0.2656

	Time between waypoints (s)				
	Start -> P1	**P1 -> Task**	**Task -> P2**	**P2 -> P3**	**P3 -> End**
Shared Control	43.406	72.691	63.056	134.101	61

After analyzing the results, it is perceivable that navigation using shared control is more efficient than the previous methods. Looking at Figure 6.5, there are five marked points on the map that correspond to events that trigger decision making mechanism, and are explained as follows:

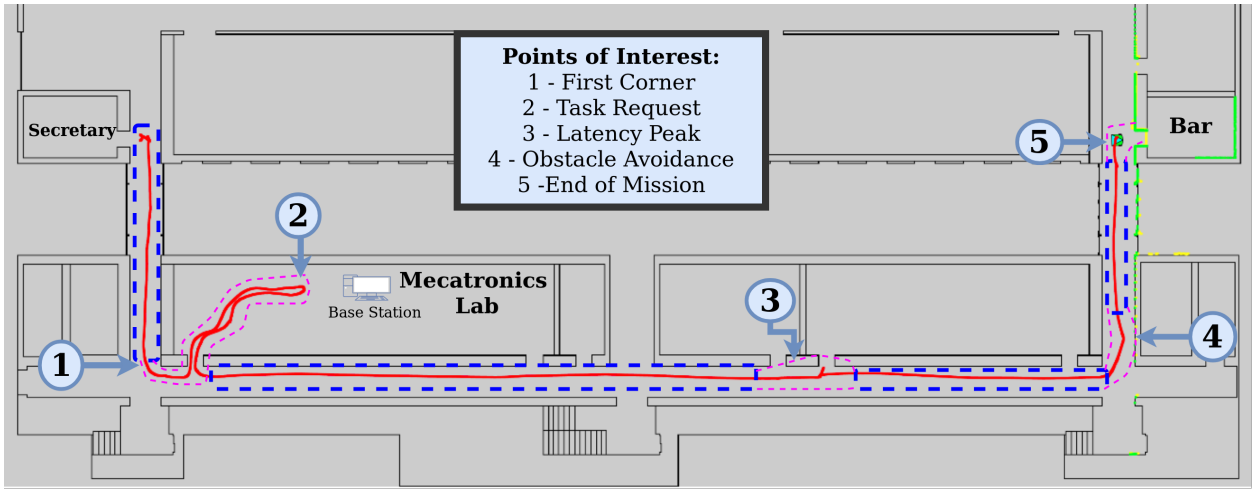


Figure 6.5: Experimental circuit used for Shared Control. Blue dashed sections correspond to teleoperation, and pink dashed sections to autonomous navigation. Points of event occurrence are marked and numbered.

1. **First Corner** - Until this point, Interbot was directly being teleoperated through the Base Station. Upon reaching a corner, the user let autonomous navigation take over.

2. **Task Request** - After reaching the intermediate goal inside the room, as requested, Interbot pauses and using Tablet shows the information regarding the requested task, how the user

should cooperate and the final destination. For the purpose of this test, the task was almost immediately met by the user controlling the Base Station.

3. **Latency Peak** - A momentary Wi-Fi loss occured, while the Processing Unit was reconnecting to a different hotspot, resulting in a disconnection with the Base Station. Since teleoperation was happening, the Speed Multiplexer detects the timeout and ignores the last speed command from the Base Station, allowing for the autonomous navigation take over the movement, moving Interbot to the next intermediate goal and towards the final destination.

4. **Obstacle Avoidance** - During autonomous navigation around the final bend, an unexpected moving human appeared in the way of the initially planned trajectory, so the Path Planner module has to update it to avoid the new obstacle.

5. **End of Mission** - Interbot approaches and rotates towards the destination using autonomous navigation, completing the initially requested task.

6.4 Latency Analysis

While the previously mentioned test scenarios were being tested, as well as during the development and testing of the implemented direct control mechanisms using the Tablet virtual joystick or the Base Station, it was noticed that there were zones of the ISR floor 0 where latency in the connection would grow higher than desired, and information would take longer than expected to travel from one component to another. This happened because there are multiple Wi-fi Access Points(AP) in ISR floor 0, placed in some of the laboratories, and either the Tablet or the Processing Unit would automatically reconnect to another AP when located close enough to it, leading to a longer delay in the communication. The existence of zones with a higher quantity of materials that block Wi-Fi signals such as metal, concrete or bricks could also lead to a degradation in the receiving signal quality.

The trajectory of Scenario 1 was repeated: only this time the Base Station and the Tablet would directly control Interbot, so that conclusions could be drawn regarding the existence of delays and the efficiency of the implemented mechanisms against it: the timeout periods of the Speed Multiplexer. The speed commands are sent every 100 ms, during the control loops of the Base Station and Tablet. Figure 6.6b shows the normalized distribution of the Base Station `clock()` function period during Test 1, proving that it is executed within acceptable times and non-blocking.

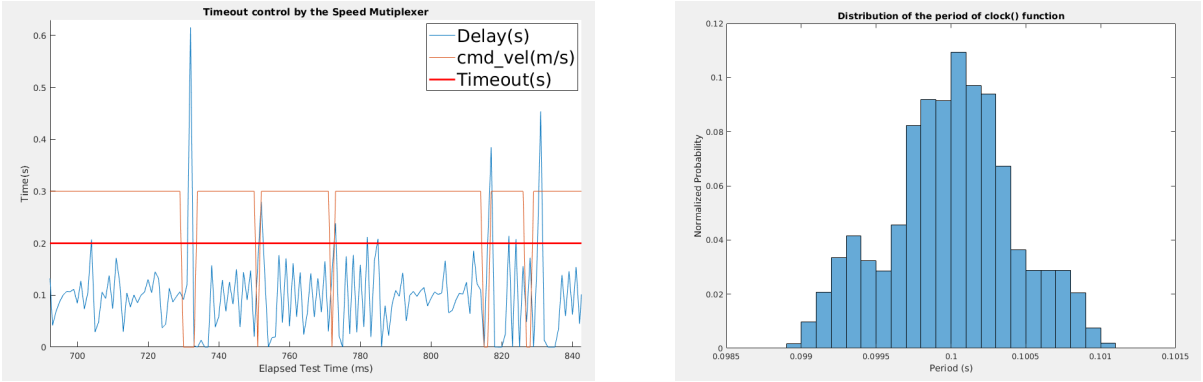
Looking at the presented results in Table 6.8, it is noticeable that Section 2 is the most critical, where the connection gets weakened and the period of the received messages start to deviate from the norm. In the tested trajectory, Section 1 and 3 also pass through or near this critical zone, which leads to some peak values that disturb the mean and standard deviation values. In most situations the mean value on the received commands is very similar or exact to the period on the sender side.

Figure 6.6a shows a situation where the timeout defined in the Speed Multiplexer was exceeded, during Section 1 of Test 1, while using the Base Station. The timeout limit was 0.2 seconds, so if an incoming message from the Base Station with speed commands took longer than it, the output

velocity `cmd_vel` was set to 0 momentarily, safeguarding the robot safety even if the connection delay is above the desired.

Table 6.8: Values obtained for the mean and standard deviations of the received commands period.

Test #	Controller	Period of received speed commands (ms)							
		Section 1		Section 2		Section 3		Section 4	
		mean	std	mean	std	mean	std	mean	std
1	**Base Station**	100	56.3	101.2	119.1	100	122.5	100	58.2
2	**Base Station**	100	62.1	100.8	80.5	146.7	330.9	100	39.4
3	**Base Station**	100	73	101.3	133.5	99.9	74.8	100	50.9
1	**Tablet**	100.9	41.2	116.2	117.9	100	34.1	100	39.6
2	**Tablet**	103.3	100.1	110.8	135.9	101.7	74.5	100.4	63.2
3	**Tablet**	106.2	75.9	116.3	112.7	101.8	44.1	100	38.8



(a) (b)

Figure 6.6: Performance data during Test 1: (a) Situation where the speed command timeout was exceeded, and the `cmd_vel` is set to 0; (b) Normalized distribution of the Base Station `clock()` period, mentioned in 5.3.6.

The previous results seem to prove that there were indeed problematic areas regarding Wi-Fi connection. All the latency data from the previous Test Scenarios was analyzed, and points where this timeout exceeded 1 second were saved. Figure 6.7 displays a map containing all these points, and it is noticeable that near the Library there is an area where connection is momentarily dropped constantly.

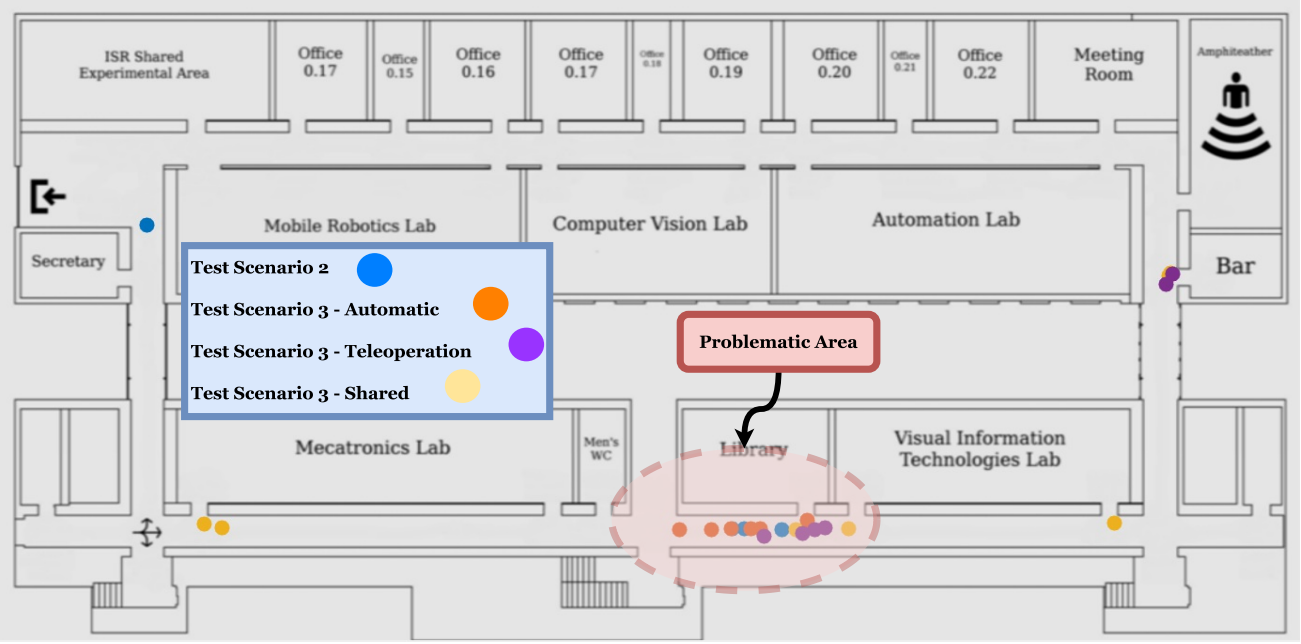


Figure 6.7: Map containing points where the connection between Base Station and Processing Unit exceeded 1 second delay during the Test Scenarios.

6.4.1 Discussion

Even though there were problems regarding latency in some areas on both connections present in Interbot system, it is still capable of reliably performing tasks and interact with humans, as the previous Test Scenarios showed. This is possible due to the decoupled software architecture implemented, as depicted in Section 4.2, since in the event of a connection delay with one of the higher level components while performing a task, the Path Planner and SLAM still continue to function and move Interbot towards the next intermediate or final goal. Once the connection is back to normal, thanks to the reliability of both TCP connections, the Base Station will receive and process the delayed information, plan the next task or goal, send it to the Tablet and resume normal work-flow.

There are factors that need to be taken into account, such as the limitation on the long term deployment of Interbot due to the battery capacity of itself and the Processing Unit laptop. Using a TCP connection for controlling a high traffic ROS system might not be the best option, as presented in [31], since it leads to higher latency values, delays and jitter in Wi-Fi networks. During these tests Interbot did not have any navigation problem or localization losses, but in the event of a highly dynamic environment this could not be the case anymore. Therefore, the SLAM method used should be improved by integrating some other type of sensor data, like the improved calculated odometry obtained from the tuning in Test Scenario 1.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this dissertation, the bulk of the work was mainly focused on improving Interbot's Human-Robot Interaction component. A study of different types of office robots was conducted, leading to the decision of implementing a remote control station capable of supervisory and direct control, and to add a Tablet as an accessible user interface capable of presenting the robot state, to order tasks or ask for human assistance.

Most of the integrated software in these two new components had to be developed from scratch, from the algorithms/functions to the UI design, and in the end they proved to be appropriate for the initial goals. Several modules of this software could even be re-used in other robot projects, which was one of the software design goals. Qt Framework also proved to be instrumental in the conception, testing and debugging of every implemented functionality. The Tablet is used as a Human-Robot Interface, is capable of providing an easy and fast way for users to be escorted through ISR floor 0, or to order Interbot to perform tasks such as message or objects delivery between rooms. The Base Station GUI, monitoring and control functions provide an effective way to plan tasks sequentially, to display the robot information in a perceivable way and to dynamically adjust configurations on the fly.

With the future goal of deploying Interbot in a populated office environment, the contributions of this work laid the groundwork for the social aspect of Interbot, allowing for further improvements to be implemented, while achieving the initial main goal.

7.2 Future work

To continue improving the current work on the Interbot platform several areas have to be focused on, such as:

- **SLAM fusion with other methods**: Relying only on laser scan information can lead to localization losses on long corridors without features. Therefore, a fusion with another localization method would be beneficial. Odometry data can now be integrated, since there was an improvement in the calculated odometry, as shown in Section 6.1.1.2. Other methods can be considered, such as using the Microsoft Kinect for 3D mapping, Wi-Fi localization or IMU data.

- **Latency, Transport Protocols**: ROS by default uses TCP to communicate between nodes. However, with an increasing flow of data, more robots connected to the same master, or with

a low-bandwith connection, this could lead to undesired delays. Different transport protocols, such as User Datagram Protocol (UDP), or even a Multi-Master ROS system, as mentioned in [31], could improve the performance.

- **Social aspect**: More features can be added to the Interbot human interfaces, such as speech outputs and voice recognition. More tasks where assistance is required can be integrated, such as opening elevator doors and selecting the floor number.
- **Cooperation with other robots**: The Base Station can further be improved to be able to control multiple robots at once, since the only inputs it requires are ROS topics.

Chapter 8

Bibliography

[1] T. Fong, C. Thorpe, and C. Baur, "Collaboration, dialogue and human-robot interaction," in *10th International Symposium of Robotics Research*, 2001.

[2] A. Conceicao, "Interbot mobile robot: Navigation modules," Master's thesis, Department of Electrical and Computer Engineering, University of Coimbra, 2016.

[3] S. L. Pfleeger, *Software Engineering: Theory and Practice.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1998.

[4] S. Rosenthal, J. Biswas, and M. Veloso, "An effective personal mobile robot agent through a symbiotic human-robot interaction," in *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pp. 915–922, 2010.

[5] T. Sheridan, "Eight ultimate challenges of human-robot communication," in *Proceedings 6th IEEE International Workshop on Robot and Human Communication. RO-MAN'97*, pp. 9–14, 1997.

[6] F. Driewer, M. Sauer, and K. Schilling, "Discussion of challenges for user interfaces in human-robot teams," in *Proceedings of the Third European Conference on Mobile Robots*, 2007.

[7] T. Fong and C. Thorpe, "Vehicle teleoperation interfaces," *Autonomous Robots*, vol. 11, pp. 9–18, 2001.

[8] B. Lewis, B. Tastan, and G. Sukthankar, "Improving multi-robot teleoperation by inferring operator distraction," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, (Richland, SC), pp. 1505–1506, International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[9] J. Y. C. Chen, E. C. Haas, and M. J. Barnes, "Human performance issues and user interface design for teleoperated robots," *IEEE Transactions on Systems Man and Cybernetics Part C (Applications and Reviews)*, vol. 37, pp. 1231–1245, 2007.

[10] E. Prassler, G. Lawitzky, A. Stopp, G. Grunwald, M. Hägele, R. Dillmann, and I. Iossifidis, *Advances in Human-Robot Interaction.* Springer Tracts in Advanced Robotics, Springer Berlin Heidelberg, 2004.

[11] J. Burke, R. Murphy, E. Rogers, V. Lumelsky, and J. Scholtz, "Final report for the darpa/nsf interdisciplinary study on human–robot interaction," *IEEE Transactions on Systems Man and Cybernetics Part C (Applications and Reviews)*, vol. 34, pp. 103–112, 2004.

[12] M. B. Dias, B. Kannan, B. Browning, E. Jones, B. Argall, M. F. Dias, M. B. Zinck, M. Veloso, and A. T. Stentz, "Sliding autonomy for peer-to-peer human-robot teams," Tech. Rep. CMU-RI-TR-08-16, Pittsburgh, PA, April 2008.

[13] R. Jarvis and A. Zelinsky, *Robotics Research: The Tenth International Symposium*. Springer Tracts in Advanced Robotics, Springer Berlin Heidelberg, 2014.

[14] J. Blanchette and M. Summerfield, *C++ GUI Programming with Qt 4*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2006.

[15] D. Kragic, "Acting, interacting, collaborative robots," in *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '17, (New York, NY, USA), pp. 293–293, ACM, 2017.

[16] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "Experiences with an interactive museum tour-guide robot," *Artificial Intelligence*, vol. 114, pp. 3–55, 1999.

[17] R. Siegwart, K. O. Arras, S. Bouabdallah, D. Burnier, G. Froidevaux, X. Greppin, B. Jensen, A. Lorotte, L. Mayor, M. Meisser, R. Philippsen, R. Piguet, G. Ramel, G. Terrien, and N. Tomatis, "Robox at expo.02: A large-scale installation of personal robots," *Robotics and Autonomous Systems*, vol. 42, pp. 203–222, 2003.

[18] G. Kim, W. Chung, K.-R. Kim, M. Kim, S. Han, and R. Shinn, "The autonomous tour-guide robot jinny," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 4, pp. 3450–3455, 2004.

[19] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *IEEE International Conference on Robotics and Automation*, pp. 300–307, 2010.

[20] B. Coltin, J. Biswas, D. Pomerleau, and M. Veloso, "Effective semi-autonomous telepresence," in *Proceedings of the RoboCup Symposium*, 2011.

[21] M. Veloso, J. Biswas, B. Coltin, S. Rosenthal, T. Kollar, C. Mericli, M. Samadi, S. Brandao, and R. Ventura, "Cobots: Collaborative robots servicing multi-floor buildings," in *Proceedings of IROS'12, the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5446–5447, 2012.

[22] S. Thrun and A. Bucken, "Integrating grid-based and topological maps for mobile robot navigation," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 944–951, 1996.

[23] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pp. 155–160, 2011.

[24] A. Lopes, J. Rodrigues, J. Perdigao, G. Pires, and U. Nunes, "A new hybrid motion planner: Applied in a brain-actuated robotic wheelchair," *IEEE Robotics and Automation Magazine*, 2016.

[25] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning a star," *Artificial Intelligence*, vol. 155, pp. 93–146, 2004.

[26] S. Quinlan and O. Khatib, "Elastic bands: connecting path planning and control," in *Proceedings IEEE International Conference on Robotics and Automation*, pp. 802–807, 1993.

[27] N. Garfield, *Floyd-Warshall Algorithm.* Anim Publishing, 2011.

[28] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *International Conference on Robotics and Automation*, 2009.

[29] D. Goncalves, "Robchair 2.0: Simultaneous localization and mapping and hardware/software frameworks," Master's thesis, Department of Electrical and Computer Engineering, University of Coimbra, 2013.

[30] A. Conceição, D. Pereira, and U. Nunes, "Assisted mobility supported by shared-control and advanced human-machine interfaces," tech. rep., Department of Electrical and Computer Engineering, University of Coimbra, 2015.

[31] D. Tardioli, R. Parasuraman, and P. Ogren, "Pound: A ros node for reducing delay and jitter in wireless multi-robot networks." `https://arxiv.org/abs/1707.07540`, 2017.

Appendices

Appendix I

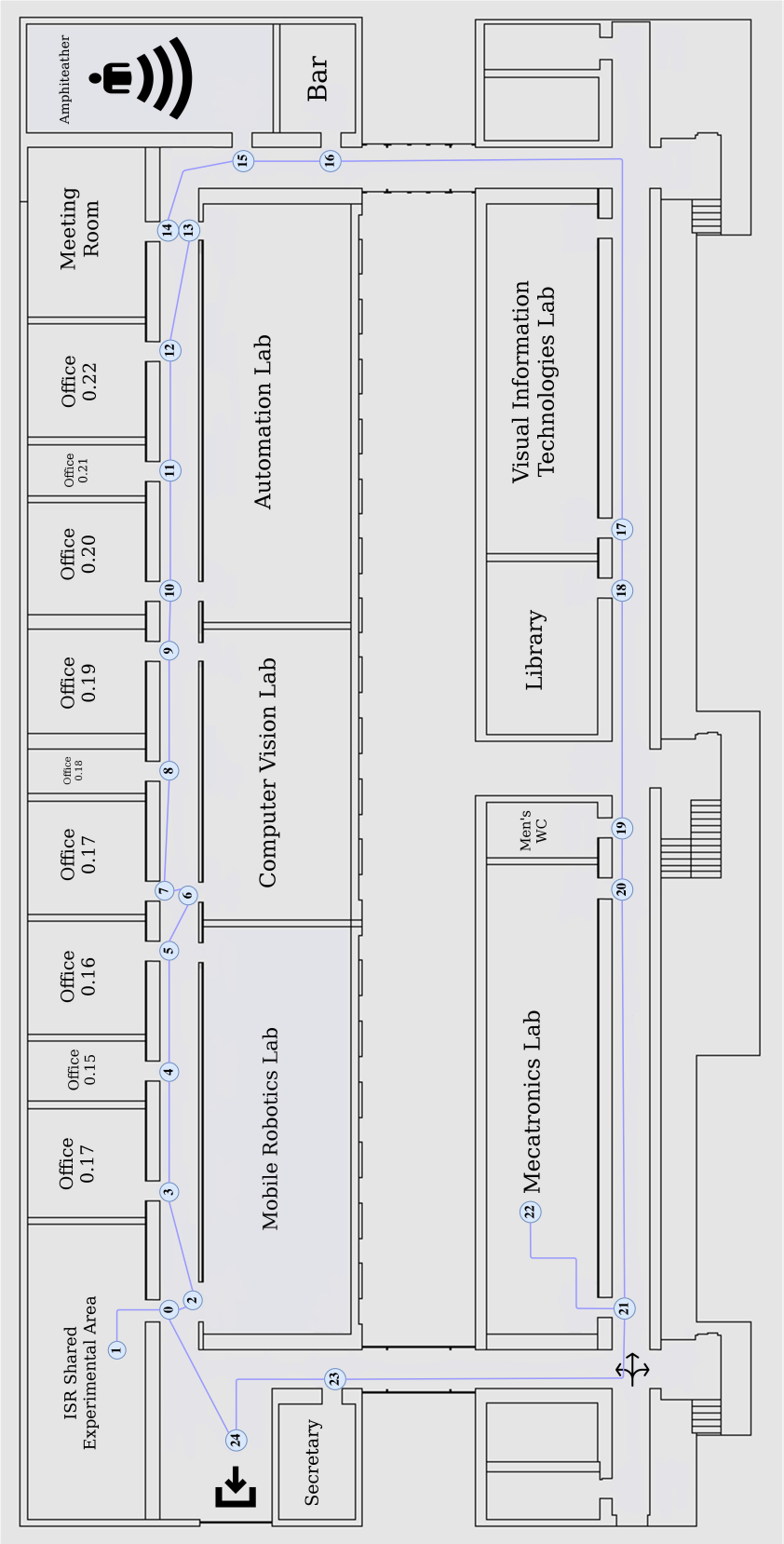


Figure 1: Node diagram for the nodes composing the graph used by the robotMovement algorithm, representing the position of each node and the arcs that connect each node.

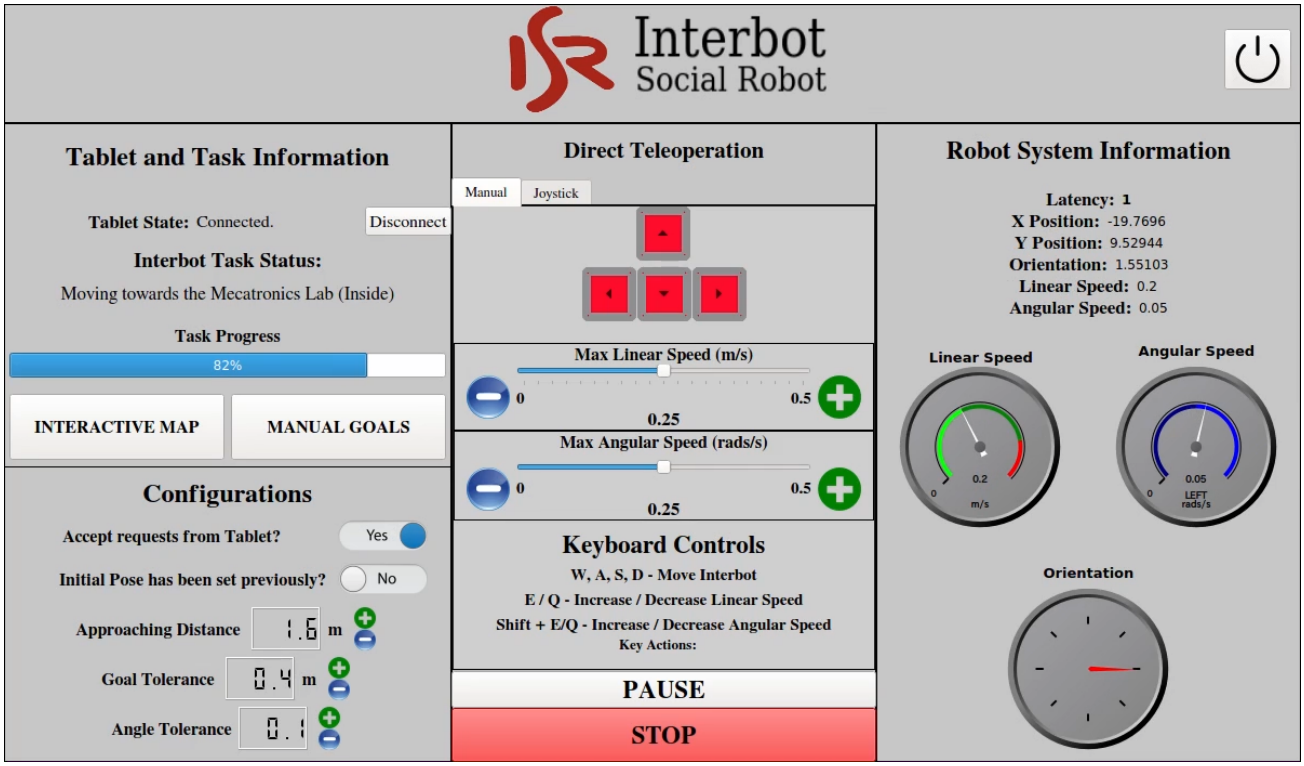


Figure 2: Base Station User Interface, while a mission was being performed, displaying information about the robot state, task and tablet status, configurations and teleoperation windows.

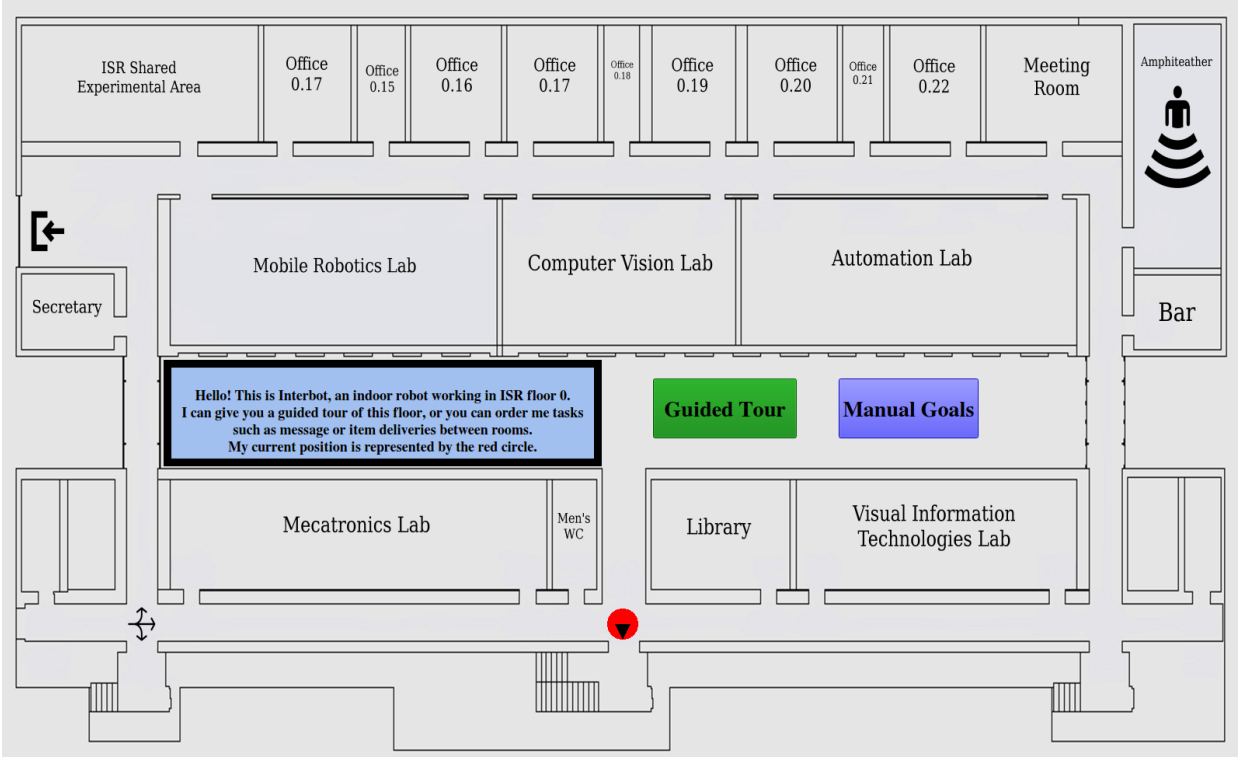


Figure 3: Tablet User Interface displaying the interactive map, dialogue messages and Interbot current position, represented by the red circle. Each named room corresponds to a point of interest.