POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Master's Degree Thesis
In collaboration with InnoTech Systems

Design and implementation of internal and external communication systems for a fleet of autonomous robots

Supervisors

Prof. Marcello CHIABERGE

Prof. Marina MONDIN

Candidate

Michele GIONFRIDDO

A.A. 2019-2020

Abstract

This thesis regards the design and development of the communication side of a fleet of autonomous service robots operating in the University Citadel of the California State University in Los Angeles. This work is part of the project run by InnoTech-System, a start-up based in San Diego, that aims to create an autonomous delivery robot service for students. After a short introduction on robotics and service robots, the project is briefly presented in all its parts. The preliminary step of the work was to collect information on the mobile autonomous delivery systems solutions existing on the market. From these examples, the communication requirement have been defined and the design phase begun. The design is composed of three different parts. Robot to Robot (R2R) communication, Robot to Infrastructure (R2I) communication and User Interface. The various protocols, together with the ISO/OSI communication stack and the Infrastructure of the system where then chosen. Once the design phase was completed, every code was then implemented and the system was simulated in its entirety, from the User Interface to the service devices. The work was then concluded with a verification step of the complete system, to underline possible weaknesses and bottom-necks under common stress conditions.

Acknowledgements

I would first like to thank Khashayar Olia, project tutor, CEO and co-founder of the InnoTech System startup, for the time and support he gave me in the last months.

I would also like to thank Marina Mondin and Fred Daneshgaran for their precious advice and for giving me the opportunity to live this experience. The time I dedicated to the InnoTech System project was a great opportunity to put into practice what I learned in these five years of university and to grow professionally in the fields I love most.
I couldn't meet you in person but I hope to meet you soon.

An infinite thanks to all my family. Grandparents, uncles, cousins. although I have spent most of the last years away from you and away from home, my heart has always been here in Sicily.

Thanks to Matteo, Sebastiano and Grazia. To the crazy family that has never made me miss anything, that has always pampered me and made me happy, that has allowed me to fully concentrate to reach this first great goal of my life.

Last but not least, thanks to friends and all the wonderful people I have spent time with over the past years. All of you have made me feel at home anywhere in the world.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

World is continuously changing and the human life with it. The First Industrial Revolution lead to mechanized production with water and steam power, the Second used electric power to create mass production, the Third electronics and IT to automate the world. Now a Fourth Revolution is building on the Third, and we stand on the brink of a Technological Innovation that will fundamentally alter the way we live, work, and relate to one another: Artificial Intelligence and Robotics era. Robotics is the intersection of science, engineering and technology that produces machines, called robots:
"autonomous machines capable of sensing their environment, carrying out computations to make decisions, and performing actions in the real world." [1]

As technology progresses, Robotics is continuously developing, pervading every part of the human life, from medical robotics to automated industrial production systems, from bots exploring Earth's harshest conditions places to robot that assist in almost every facet of healthcare.

Smart home systems, Roomba, exoskeletrons, are only few of the robotics systems that at least once a day we see, use or hear of. Defined from the IEEE standard as "Service Robotics", this category of robots includes all the robots that *"performs useful tasks for humans"* [2]

According to the Executive Summary World Robotics 2019 Service Robots [3] the total number of service robots for personal and domestic use increased by 59% to about 16.3 million units in 2018 and will continue to increase strongly in

the near future, expecting to reach 68 million units by 2022.

1.1 Autonomous Mobile Robots

Within the last years, many innovative manufacturing initiatives have been started all over the world, driving for re-establishing and regaining a significant industrial share in the economy. Among all, in the last decade especially, a new category of robots is rewriting the rules of the robotic world, overcoming one of the biggest weakness of traditional devices, lack of movement and autonomy: Autonomous Mobile Robots (AMRs). Mobile autonomous robotics is one of the most promising industry of the near future. Despite the numerous difficulties that mobility and autonomy inject in the design and implementation of the robots, technological improvements both in the hardware and in the associated software are two of the key reasons beyond the growing applications of these systems. The decreasing cost of high-computation control units and high-accuracy sensors, the efficiency of new communication protocols, allowed robots to complete tasks that until recently were outside the scope of robotic. In particular, with most of us confined to our homes to help contain COVID-19 pandemic, the topic of automated last-mile food delivery services – and the current overwhelming demand for them – has taken central stage. Not only has online food delivery industry had a great development in the last decade, growing steadily year by year, but in the period of forced quarantine that we experienced in the past months, it has also seen a further surge. The convenience of receiving food while staying comfortably at home together with the tranquility of the lack of direct contact with the delivery man, make this shipping method extremely popular and safe.

1.2 Goals of the thesis

The aim of this writing is to describe the work done with the InnoTech System startup during the design and implementation of an autonomous system for last-mile food delivery, using a fleet of automated robots. The project, that will be briefly described, has been divided into multiple sub-project and, among them,

the communication system design and implementation will be covered. In this paper, all the communication aspects will be studied: from the communication between the robot and the external world, up to the front-end interface that will allow the user to use the service, going through the communication between the internal robot components. The preliminary step of the work was to study the mobile autonomous delivery systems solutions existing on the market. From these examples, together with the InnoTech desires, the communication requirements have been defined and then the design phase begun. After the analysis of the most common solution adopted in similar communications, the different protocols and strategies to implement the system were defined and the coding started. This work presents then a simplified view of the codes of the system and, with the communication system complete, a verification step follows. It consists in the simulation and testing of the system under certain stress condition, to underline possible weakness or operative bottle-neck situation of the adopted solution. The discussion concludes with final remarks and future development ideas for the system.

1.3 Existing systems

Starship

The delivery device from Starship Technologies (1.1) [4], equipped with six-wheels and a sensor suite that includes cameras, GPS and inertial measurement units, can deliver items within a 6km radius. It has a large compartment able to hold deliveries of two grocery bags in size, with a maximum weight of 10kg. Once the robot has arrived at the customer, the compartment is opened using a unique code generated after placing the order from a smartphone app. In March 2017, Starship Technologies entered into a partnership with Dominos, to use the robots as "personal delivery devices".

Figure 1.1: Starship logo and robot

Kiwi

Kiwi (1.2)[5] is an autonomous ground robot that picks up and delivers food and personal-care items within a 1.5 square kilometers area around the basis. The robot uses Deep Learning and AI to interpret data gathered from its sensors and make intelligent decisions that ensure a safe, fast and cost-efficient delivery. It can cross streets, identify traffic lights and detect obstacles and objects avoiding collisions in a safe and reliable manner.

Figure 1.2: Kiwi logo and robot

Robomart

Robomart is a little mini-mart on wheels (1.3)[6]. With a driver-less technology and full autonomy for teleoperations, it is engineered with cutting-edge technology, including an RFID and computer vision-based checkout-free system, and purpose-built refrigeration and temperature control. The devices are nowadays used by big grocery store chains, to deliver groceries to its customers. When customers want to buy some groceries, they simply tap a button in a smartphone app to request the nearest Robomart. When it arrives, the user can unlock unlock the doors and pick the products he wants. Robomart tracks what customers have taken using patent-pending "grab and go" checkout-free technology and will charge them and send a receipt accordingly.

robomart

Figure 1.3: Robomart logo and robot

TeleRetail

The TeleRetail robot uses AI to automate last-mile logistics and minimize the ecological footprint of transportation. The aim of the company is to help shops and small businesses to compete with major e-commerce companies like Amazon. The robot is made by the Swiss startup TeleRetail (1.4)[7] founded in 2014 with the aim of solving the logistical problems of local businesses. With its 84 centimeter width and capability to carry up to 35 kilograms payloads, the robot is designed to travel on sidewalks, covering long distances up to about 80 kilometers.

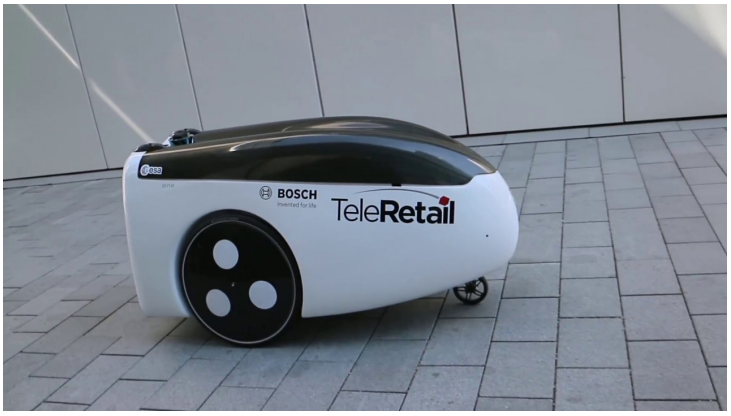

Figure 1.4: TeleRetail logo and robot

Amazon Scout

Amazon Scout (1.5)[8] is a six-wheeled autonomous delivery machine that uses self-driving technology to navigate through neighborhoods to deliver packages to Amazon Prime Customers. The vehicle, able to walk at a human pace, appears large enough to accommodate small and medium-sized parcels, but no word yet on what's under the hood.

Figure 1.5: Amazon Scout logo and robot

FedEx SameDay Bot

FedEx SameDay Bot (1.6)[9] is the latest of a long line of last-mile delivery robots that transport small packages from shops centers directly to consumers. The bot is still in developing phase and the company will begin deployment as soon as performances are satisfactory. The bot features pedestrian-safe tech, multiple cameras, Light Detection and Ranging (LiDAR) and machine learning to help it detect and avoid obstacles, and navigate uneven surfaces.

Figure 1.6: FedEx SameDay logo and robot

1.4 The project

The project described in this thesis is run by InnoTech System [10], a startup based in San Diego, California, in collaboration with the California State University of Los Angeles (CALState LA). The aim of the project is to design and develop a fleet of autonomous guided vehicles (AGVs) to create a delivery service inside the University Citadel. The first step done for the realization of the project was to do a preliminary research on the necessities and the problems that the user (primarily the student) experiences to be fed inside the university citadel. Notwithstanding the majority of the university citadels has got its own restaurant service and

many food commercial exercises right near, the high food request, rush-hours and limited menu, make difficult one of the principal necessities of the student, nutrition. From the research brought on by the university of Los Angeles, a big part of the students, many times a week has a meal outdoor. But, because of the queuing time, especially during the rush hours, very often the student has to postpone or skip the meal. In addition, part of the interviewed people has also expressed their preoccupation and diffidence on the use of the university or public restaurant, mainly because of the rush of people in a closed place during the ongoing emergency of COVID-19. A delivery service can then offer a valid solution to solve all the aspect linked to the nutrition.

1.4.1 Overview

The driving idea of the project was to give the possibility to the user to receive food and drinks without moving too much from the current location (classroom or study room), saving time and energy. The project, started March 2020, is still in the prototyping phase. The following discussion and description has then to be intended as at prototype level. The company aims to create the first fully-working device by the end of 2020. The project has been divided into 3 macro areas, regarding:

- SLAM
- Path Planning
- Robot Communication

1.4.2 Device Architecture

Before going into the details of the project parts, it is useful to understand the architecture of the single device, with the hardware connections between the internal components. In figure 1.7 the details of the device.

The “brain” of the systems consists in the Jetson TX2. All the high level software is running inside of it, in a Linux Ubuntu distribution. The Jetson TX2 is in charge of collecting the data coming from the different peripherals and

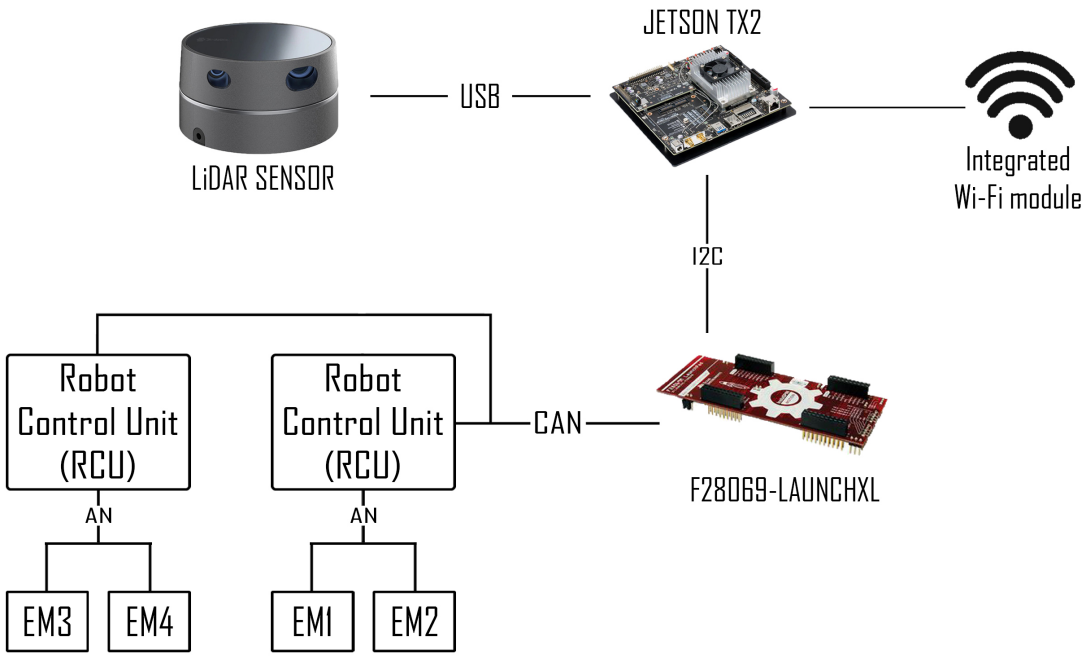

Figure 1.7: Device architecture

processes them for Path planning, localization, mapping, object recognition and obstacle avoidance. The board is connected over USB with the LiDAR hardware. During the navigation, the obstacle avoidance procedure uses two Narrow Field-Of-View LiDARs to detect the obstacle itself. These sensors send messages to the F28069-LAUNCH XL evaluation board that commands the electric motors to stop. The Jetson TX2 communicates with the main RCU (F28069-LAUNCH XL) via a I2C bus. The RCU runs the motors control algorithm, and controls messages are sent via CAN bus to two smaller custom printed PCBs RCUs provided with H-bridges to power the motors.

1.4.3 LiDAR-based SLAM

SLAM, or Simultaneous Localization and Mapping, solves the problem of localizing a vehicle in an unknown and unstructured environment while mapping it. With

the perception of sensors and cameras, the SLAM algorithm is able to create a map to exploit the environment, update the position of the robot and make successive localization faster and reliable. The most common SLAM systems rely on optical sensors, the top two being visual SLAM (VSLAM, based on a camera) or LiDAR-based (Light Detection and Ranging), using 2D or 3D LiDAR scanners. While the VSLAM system use images from the stereo-camera, an hardware setup capable of capturing two images of the same scene from two slightly different point of view, to create the map of the environment relying in the principle of Triangulation, a LiDAR-based SLAM system uses a laser sensor. In the project, the SLAM sensor used is a LiDAR-based one. LiDAR measures the distance to an object (for example, a wall or chair leg) by illuminating the object with multiple transceivers. Each transceiver quickly emits pulsed light, and measures the reflected pulses to determine position and distance. Because of how quickly light travels, very precise laser performance is needed to accurately track the exact distance from the robot to each target. This requirement for precision makes LiDAR both a fast and accurate approach. However, that 's only true for what it can see. One of the main downsides of 2D LiDAR, that is the most common solution adopted in robotics application, is that if one object is occluded by another at the height of the LiDAR, or an object is an inconsistent shape that does not have the same width throughout its body, this information is lost. After the map was generated the software would localize the robot and send such information to the RCU in order to complete the prescribed mission and reaching the final destination. The software, entirely developed in C++, runs on the Ubuntu environment installed on the NVIDIA development board, Jetson TX2, using the Hector SLAM and AMCL libraries.

1.4.4 Path Planning

Path planning, also known as motion planning, aims to find a sequence of valid configurations that moves one object from the source to the destination. In the navigation of a mobile robot, in particular, the path planning algorithm generates the path that, through speed and turning commands moves the robot from the starting to the final location, avoiding obstacles. To do such, the algorithm needs

the map of the environment and the current robot's location, both information correctly created from the SLAM. The path of the robot is generated using an A* algorithm, which is a modified version of breadth first search. Using a 2D map of the environment, a graph is generated, where each point is treated as a node. The nodes are divided into pathable and unpathable categories based on whether an obstacle is present in the physical space at the node's location. These pathable nodes are the graph space upon which the modified breadth first search algorithm is run. Starting from a designated "start" node, the BFS algorithm first discovers all of the node's neighboring nodes. Then, for each of these nodes a heuristic is applied to estimate the distance towards a designated end point. These nodes are then inserted into a priority queue, where the nodes estimated to be closest to the end point are prioritized. Then, with the first node in the queue selected, the neighbors of that node are inserted into the priority queue in the same manner. This process is repeated until the end point node is both discovered and at the front of the priority queue, or until the priority queue is empty. This means that either the shortest path to the end point has been discovered, or that there is no possible path to the end point. In figure 1.8 an example of the algorithm avoiding a wall.

1.4.5 Robot Communication

The solution adopted was to use a Server as the Infrastructure, available on a fixed IP over the Internet. The devices, through Wi-Fi, establish a connection to the system with a TCP communication. The single device, uniquely identified by a number, connects to the server with a socket connection and exchange information with the system. Together with the Robot to Infrastructure communication, a Robot to Robot communication is adopted. With an aware-coordinated-distributed R2R communication approach, devices inside the system exchange logistic messages, help requests and other useful data directly each other. The server is implemented as a multi-threaded TCP server written in C++ that:

- Accepts and manages the robot connections;
- Keeps track of the status of the devices connected on the system;

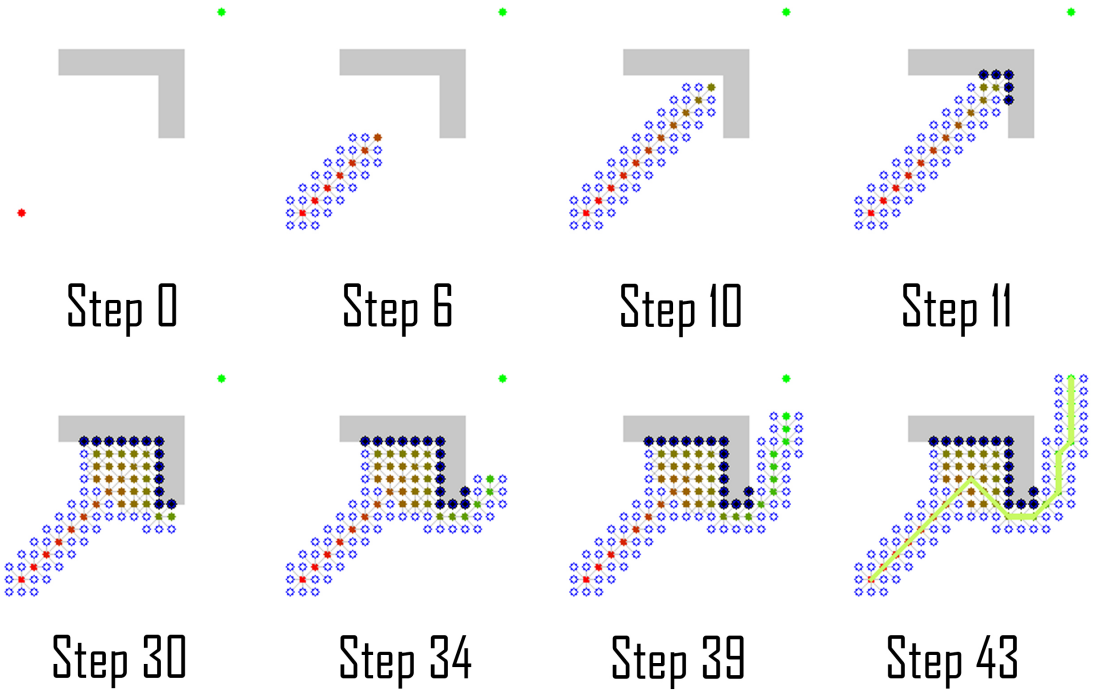

Figure 1.8: Path Planning A* algorithm

- Handles the user's command messages to start the delivery operation as soon as possible.

Using an HTML and Javascript code, when the user accesses the WebPage, the server solves the page request and, through a Web socket connection, provides the information about the system and devices. Once the User selects a task to execute, the server receives, checks and forwards the command to the robots.

Overall system architecture on Figure 1.9

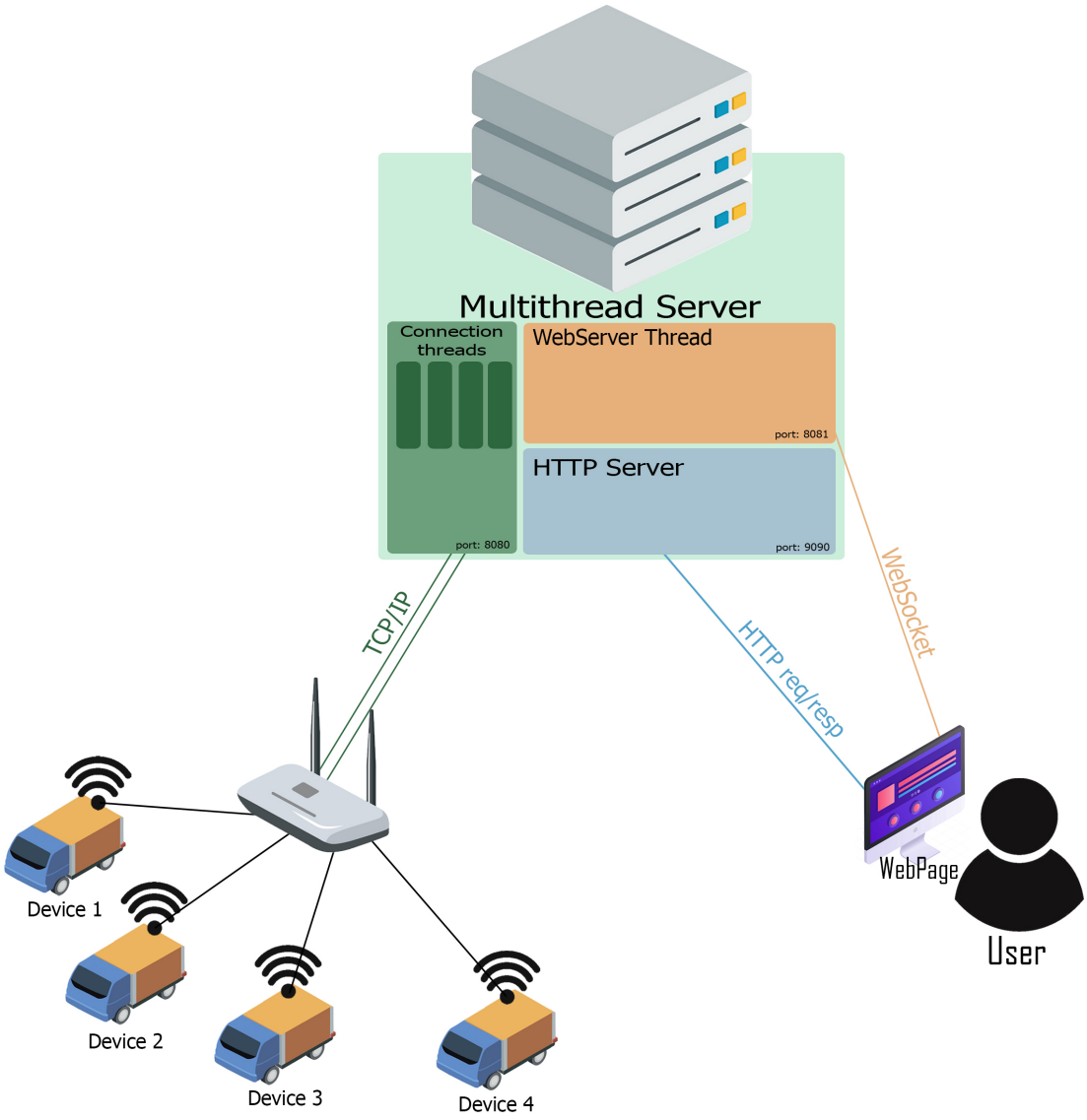

Figure 1.9: Overall system architecture

Inside the single robot, the internal communication was implemented using the Robotic Operating System (ROS). Using multiple topics and standard ROS messages, SLAM, Path Planning, RCU and sensors exchange position, localization and mapping information with the help of a middle node, called "i2c_node", created to overcome the i2c physical connection separation (Figure 1.10).

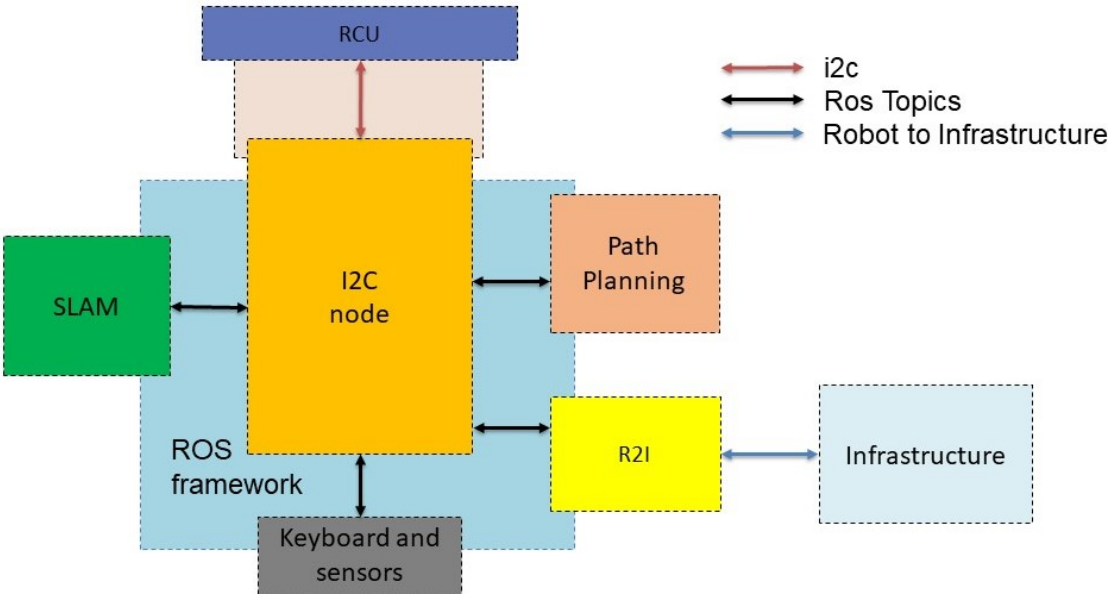

Figure 1.10: Overall system architecture

1.5 Structure of the work

The design and implementation of a communication system from scratch consists in the choice of the best protocols, libraries, structures and coding strategies to achieve an efficient and reliable information exchange inside the system. For this reason, the next chapters will present a brief overview on all the possible available solutions to implement a communication system. The thesis work is structured as follows:

- **Chapter 2** presents an overview of the ISO/OSI model, to understand the different communication stack possible.
- **Chapter 3** offers different coding strategies and concepts, needed to create the codes of the communication system.
- **Chapter 4** introduces the reader to different tools and programs used in the development and simulation phase of the system.
- **Chapter 5** is a discussion on the robot communication theory

- **Chapter 6** presents the design part of the project, underlining the requirements of the system.
- **Chapter 7** describes how the consideration made in the previous chapters have been efficiently implemented into a working code able to perform the information exchange inside the system.
- **Chapter 8** explains the simulation and testing scenario used to test the system
- **Chapter 9** closes the work with final conclusions and ideas for future development of the system.

Chapter 2

ISO/OSI Model

The starting point to create a communication is the definition of the ISO stack model where the data exchange takes place. The Open Systems Interconnection model (OSI model) is a conceptual model that standardizes the communication functions of a telecommunication system independently from the underlying technology infrastructure. Its goal is the interoperability of diverse communication systems with standard communication protocols. The model partitions a communication system into abstraction layers. A layer serves the layer above it and is served by the layer below it. For example, a layer that provides error-free communications across a network provides the path needed by applications above it, while it calls the next lower layer to send and receive packets that constitute the contents of that path.

The model, product of the Open Systems Interconnection project at the International Organization for Standardization (ISO), divides data communication into seven abstraction layers:

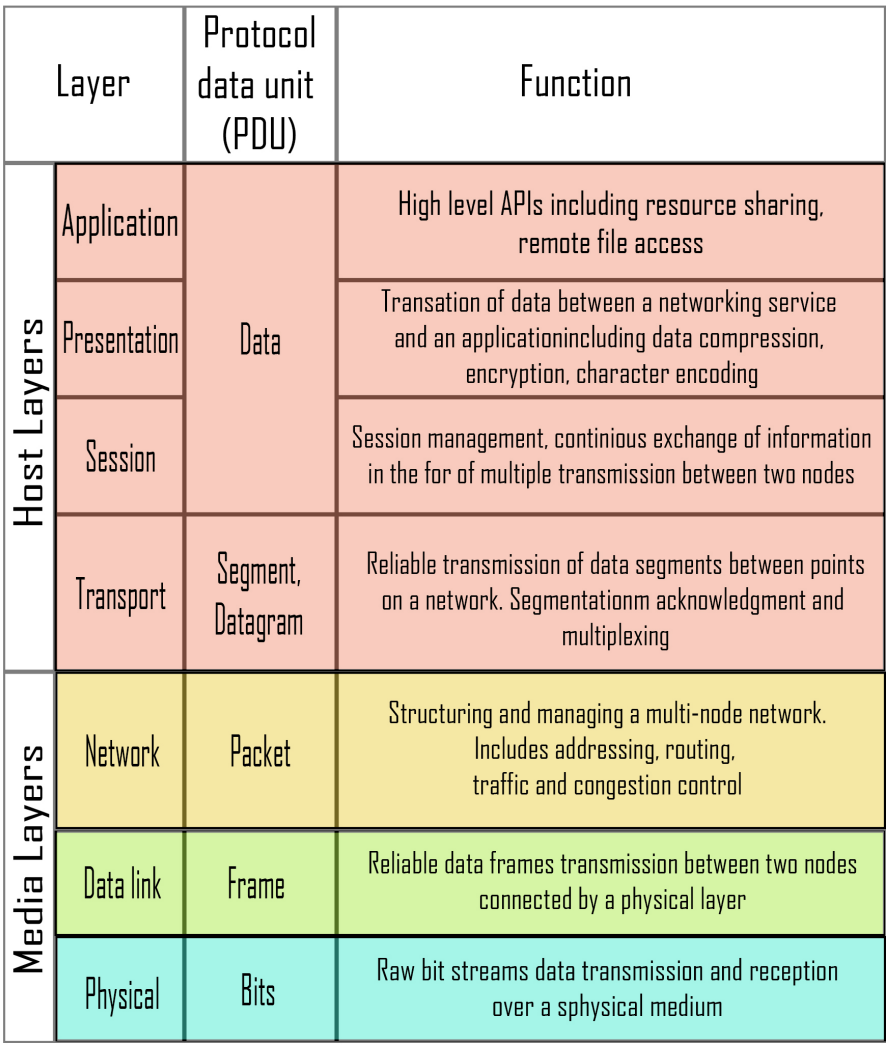

Figure 2.1: ISO/OSI Stack

- **Physical Layer**
  The physical layer, the lowest layer of the OSI model, is responsible for the transmission and reception of unstructured raw data between a device and a physical transmission medium. It converts the digital bits into electrical,

radio, or optical signals. Layer specifications define a range of aspects, including: voltage levels, physical data rates, data transmission performance, timing of voltage changes, layout of pins, communication modes, network topologies, modulation of the signals, cable specifications, signal timing and frequency for wireless devices.

- **Data Link Layer**
  The data link layer provides node-to-node data transfer—a link between two directly connected nodes. It detects and possibly corrects errors that may occur in the physical layer. It defines the protocol to establish and terminate a connection between two physically connected devices. It also defines the protocol for flow control between them. IEEE 802 [11] divides the data link layer into two sub-layers: Medium access control (MAC) layer – that controls how devices in a network gain access to a medium and permission to transmit data. Logical link control (LLC) layer – that identifies and encapsulate network layer protocols, with error checking and frame synchronization. The MAC and LLC layers of IEEE 802 networks such as 802.3 Ethernet, 802.11 Wi-Fi, and 802.15.4 ZigBee operate at the data link layer.

- **Network Layer**
  The network layer provides the functional and procedural means of transferring variable length data sequences (called packets) from one node to another connected in "different networks". If the message is too large to be transmitted from one node to another on the data link layer between those nodes, the network may implement message delivery by splitting the message into several fragments at one node, sending the fragments independently, and reassembling the fragments at another node.

- **Transport Layer**
  The transport layer provides the functional and procedural means of transferring variable-length data sequences from a source to a destination host, while maintaining the quality of service functions. The transport layer controls the reliability of a given link through flow control, segmentation/desegmentation, error control. In addition, keeps track of the segments and re-transmit

those that fail delivery in connection-oriented communications. Although not developed under the OSI Reference Model and not strictly conforming to the OSI definition of the transport layer, the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) of the Internet Protocol Suite are commonly categorized as layer-4 protocols within OSI, meaning that these protocols are the most reliable.

- **Session Layer**
  The session layer controls the dialogues (connections) between computers. It establishes, manages and terminates the connections between the local and remote application. It provides for full-duplex, half-duplex, or simplex operation, and establishes procedures suspending, restarting, and terminating a session.

- **Presentation Layer**
  The presentation layer establishes context between application-layer entities, in which the application-layer entities may use different syntax and semantics if the presentation service provides a mapping between them. This layer provides independence from data representation by translating between application and network formats.

- **Application Layer**
  The application layer is the closest OSI layer to the end user because both the OSI application layer and the user interact directly with the software application. This layer interacts with software applications that implement a communicating component. Such application programs fall outside the scope of the OSI model. Application-layer functions typically include identifying communication partners, determining resource availability, and synchronizing communication.

Now that the parts of the OSI stack were described, practical layer implementation and their details will be presented. The discussion aims to give the reader an insight, with pros and cons, on the possible communication solution normally adopted in this kind of communications and systems. However, only a set of the

layer will be discussed, because . To avoid a long discussion, not all protocols will be presented, but only the ones that are suitable for the kind of system of the project.

2.1 Physical and Link layer

Among all the physical/link layer technologies possible, for the purpose of the project, the kind of communication protocols to take into account are the wireless one. In fact the system relies on an autonomous movement of the devices, that will be free to move in open space without any physical connection.
In early robotic communication, infrared technology was by far the most widely used due to its low cost. The main problem with this technology was that the infrared signal cannot pass through physical obstacles such as walls and furthermore infrared systems are of poor quality and speed (rain effect). For this reason, radio frequency technologies are preferred in the design of mobile robotic communications. Robots can communicate point-to-point with other links in the network or through the broadcasting transmission mechanism. FHSS (Frequency Hop Spread Spectrum) and DSSS (Direct Sequence Spread Spectrum) modulation technologies are widely applied to the Industrial Scientific Medical (ISM) band (2.4 GHz), which is unlicensed in many countries. The proliferation of Internet-like networks has motivated research to address wireless LAN (IEEE 802.11). Nowadays, there are many radio communication standards. Wi-Fi, Bluetooth, Zigbee, are just three of the most common. Each of them has its different features, cost and capabilities. Not all of them are well matched to the delivery task of this project.
Each of these protocols will be briefly presented, with its ISO/OSI stack (Figure 2.2) and characteristics (Figure 2.1).

Figure 2.2: Wi-Fi, Bluetooth and Zigbee ISO/OSI Stack

	ZigBee	Bluetooth	Wi-Fi
Frequency band(s)	868, 915 MHz, 2.4 GHz	2.4 GHz	a:5GHz, b,g:2.4GHz
Bandwidth	600 kHz, 1.2, 2.0 MHz	51 MHz	a: 16.25 MHz, b: 22 MHz g: 16.25-22 MHz
N. transmission power	(-25) – 0 (dBm)	0 – 10 dBm	15 – 20 dBm
Raw data rate (max)	250 Kbps	3 Mbps	a,g < 54 Mbps; b < 11 Mbps
Supported number of nodes	65,535	8	2007
Power consumption	Very low	Low	High
Nominal range (m)	10-30	1-100	100
IEEE standard	IEEE 802.15.4	IEEE 802.15.1	IEEE 802.11a/b/g
Typical applications	Monitoring and control	Cable replacement	Computer networking

Table 2.1: Protocols characteristics

2.1.1 Bluetooth

Bluetooth is a technology used for exchanging data wirelessly over short distances. It uses short-wavelength UHF radio waves of frequency ranging from 2.4 to 2.485 GHz in the ISM band. Bluetooth has many applications, such as in telephones, tablets, media players, robotics systems, etc. The range of Bluetooth technology is between 50 – 150 meters and the data is being shared at a maximum data rate of 1 Mbps. The main cons to this technology is that it does not permit internet connection to the device.

2.1.2 ZigBee

Zigbee is an IEEE 802.15.4-based specification for a suite of high-level communication protocols used to create personal area networks with small, low-power digital radios, such as for home automation, medical device data collection, and other low-power low-bandwidth needs, designed for small scale projects which

need wireless connection. Hence, Zigbee is a low-power, low data rate, and close proximity (i.e., personal area) wireless ad-hoc network. The technology defined by the Zigbee specification is intended to be simpler and less expensive than other wireless personal area networks (WPANs), such as Bluetooth or more general wireless networking such as Wi-Fi. Applications include wireless light switches, home energy monitors, traffic management systems, and other consumer and industrial equipment that requires short-range low-rate wireless data transfer. Its low power consumption limits transmission distances to 10–100 meters line-of-sight, depending on power output and environmental characteristics.[2] Zigbee devices can transmit data over long distances by passing data through a mesh network of intermediate devices to reach more distant ones. Zigbee is typically used in low data rate applications that require long battery life and secure networking (Zigbee networks are secured by 128 bit symmetric encryption keys.) Zigbee has a defined rate of 250 kbit/s, best suited for intermittent data transmissions from a sensor or input device.

2.1.3 Wi-Fi

One of the most common solution is to use the standard IEEE 802.11 (WiFi - Wireless Fidelity). Used for wireless local area network (WLAN), utilizes the IEEE 802.11 standard through 2.4 GHz UHF and 5 GHz ISM frequencies. The standard has its pros & cons. The infrastructure or device cost for Wi-Fi is low & deployment is easy but the power consumption is high and the Wi-Fi range is quite moderate.So, the Wi-Fi may not be the best choice for all types of applications. One of the strongest pros of the Wi-Fi is, instead, that it provides internet acces to devices that are within the range of about 20 - 40 meters from the source. It has a data rate up to 600 Mbps maximum, depending on channel frequency used and the number of antennas.

2.2 Network and Transport layer

Regarding the Network layer, there is no much discussion to present because the primary and omnipresent protocol is the Internet Protocol (IP), implemented in

two versions, IPv4 and IPv6. However, on top of this layer, multiple Transport layer can be adopted. Among them, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are the most commonly used.

2.2.1 Internet Protocol

The Internet Protocol is the principal communication protocol for relaying datagrams across the network boundaries. The IP routing function establishes the internet and has the task of delivering packets from the source to the destination host with just the information of the IP address in the packet headers. IP defines addressing methods used to label the data-gram with source/destination information and packet structures that encapsulate the data themselves. The protocol was, historically, the connection-less data-gram service in the original Transmission Control Program introduced by Cerf and Kahn in 1974, which was complemented by a connection-oriented service that became the basis for the Transmission Control Protocol (TCP). The first major version of IP, Internet Protocol Version 4 (IPv4), is the dominant protocol of the Internet. Its successor is Internet Protocol Version 6 (IPv6), which has been in increasing deployment on the public Internet since 2006.

2.2.2 TCP, UDP, RTP

On the different communication system witch are present nowadays on the market, different types of information are exchanged among robot, user and the system itself. Generally, there are three types of data:

- **Administrative data** (such as access control, user validation, and configuration data) and user control commands (such as desired translation velocity and rotation angle): small packet size, non-periodic transmission and requiring reliable delivery.
- **Image data** (the most important and costly information feedback): large packet size, periodic transmission, real-time delivery is required, requires significant bandwidth, and the most recent information is preferred should packets become lost.

- **Continuous control data** (movement commands) and feedback information on the scene and the robot (such as position of the robot and sensing data): small packet size, periodic transmission, real-time delivery is required, and the most recent information is preferred should packets become lost.

From the above categorization, it is clear that all types of information require real-time delivery except for once-for-all administrative data and user control commands. Consequently, to obtain the optimal performance, different transport protocols should be used for the transmission of each information category. There are currently three main transport protocols available for implementing remote control applications over the Internet: the User Datagram Protocol (UDP) [12], the Transmission Control Protocol (TCP) [13], and the Real-time Transport Protocol (RTP) [14].

UDP is based on the idea of sending a datagram from a device to another as fast as possible without due consideration of the network state. This protocol does not maintain a connection between the sender and the receiver, and it does not guarantee that the transmitted data packets will reach the destination as well as the chronological order of the data at the receiving end. TCP is a more sophisticated protocol which was originally designed for the reliable transmission of static data such as e-mails and files over low-bandwidth, high-error-rate networks. In each transmission session, TCP establishes a virtual connection between the sender and receiver, performs the acknowledgment of received data packets and implements the re-transmission mechanism when necessary. TCP can also adapt to the variation of network condition by applying strict congestion control policy with slow start, fast recovery, fast re-transmit and window-based flow control mechanisms. RTP is the standard for delivering real-time multimedia data. The protocol provides facility for jitters compensation and detection of out-of-sequence arrival in data, and it is usually used in conjunction with the RTP Control Protocol (RTCP).

From the previous analysis, it is well recognized that RTP is suitable for video streaming; UDP has advantages in sensing data transmitting while TCP is the best in delivering administrative data and user control commands.

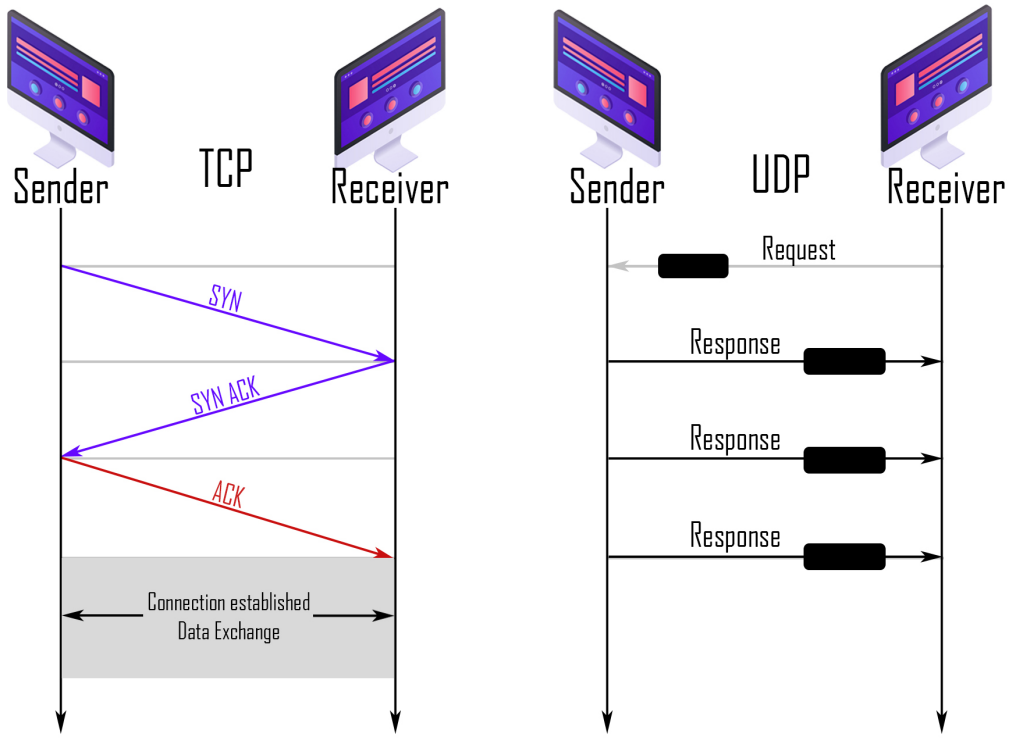

Figure 2.3: TCP vs UDP communication flow

2.3 Application layer

2.3.1 HTTP

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. This is the foundation for data communication for the World Wide Web (i.e. internet) since 1990. HTTP is a generic and stateless protocol which can be used for other purposes as well using extensions of its request methods, error codes, and headers. Basically, HTTP is a TCP/IP based communication protocol (Figure 2.4), that is used to deliver data (HTML files, image files, query results, etc.) on the World Wide Web. The default port is TCP 80, but other ports can be used as well. It provides a standardized way for computers to communicate with each other. HTTP

specification specifies how clients' request data will be constructed and sent to the server, and how the servers respond to these requests.
The HTTP protocol is a request/response protocol based on the client/server based architecture where web browsers, robots and search engines, etc. act like HTTP clients, and the Web server acts as a server.
Client: The HTTP client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a TCP/IP connection.
Server: The HTTP server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity meta information, and possible entity-body content.
There are three basic features that make HTTP a simple but powerful protocol:

- **HTTP is connection-less**: The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client waits for the response. The server processes the request and sends a response back after which client disconnect the connection. So client and server knows about each other during current request and response only. Further requests are made on new connection like client and server are new to each other.

- **HTTP is media independent**: It means, any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content. It is required for the client as well as the server to specify the content type using appropriate MIME-type. (the most important and costly information feedback): large packet size, periodic transmission, real-time delivery is required, requires significant bandwidth, and the most recent information is preferred should packets become lost.

- **HTTP is stateless**: As mentioned above, HTTP is connection-less and it is a direct result of HTTP being a stateless protocol. The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither

the client nor the browser can retain information between different requests across the web pages.

HTTP/1.0 uses a new connection for each request/response exchange, where as HTTP/1.1 connection may be used for one or more request/response exchanges.

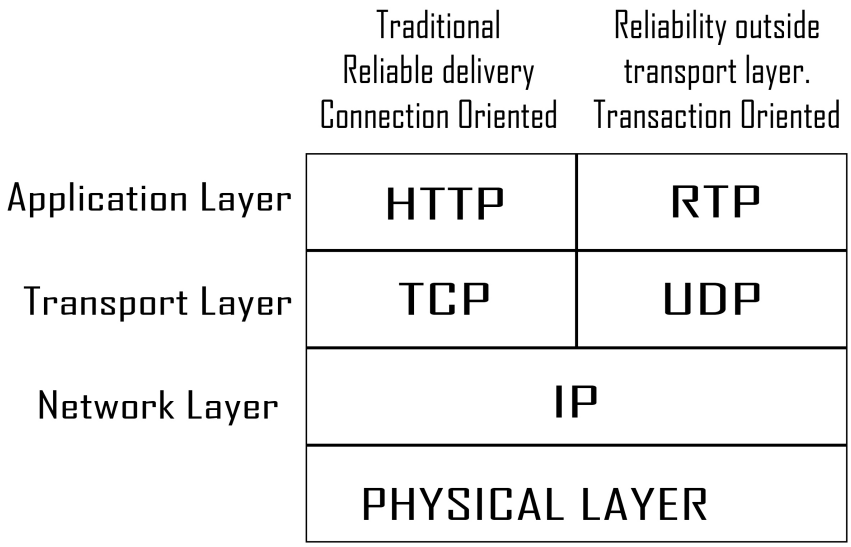

Figure 2.4: HTML stack detail

Chapter 3

Coding languages and concepts

When it comes to communication systems, it is obvious the need to use programming languages and methods to implement the system itself. The software component, which is the basis of every system, is made up of codes, data structures and approaches, which allow the communication itself to function correctly. In this chapter, a brief description of these concept will be presented. In particular, multi-threading and socket programming, the concept of device catalog, the HTML/Javascript languages, together with the Vue framework, will be discussed. In fact, given the nature of the project, made up of multiple devices continuously communicating with the system, is useful to provide the reader details on multi-threading and socket programming, as they are the most commonly used programming techniques in this type of systems. Multi-threading, with its type of concurrent instruction execution, and sockets, with an easy communication link, allow the managing of multiple data and computational flows in parallel. Similarly, HTML and Javascript are two of the most popular languages for the creation of a web page and they permit a wide range of page implementation.

3.1 Multi-threading

Multi-threading is a specialized form of multitasking that allows your computer to run two or more programs concurrently. In general, there are two types of multitasking: process-based and thread-based. Process-based multitasking handles the concurrent execution of programs. Thread-based multitasking deals with the concurrent execution of pieces of the same program.

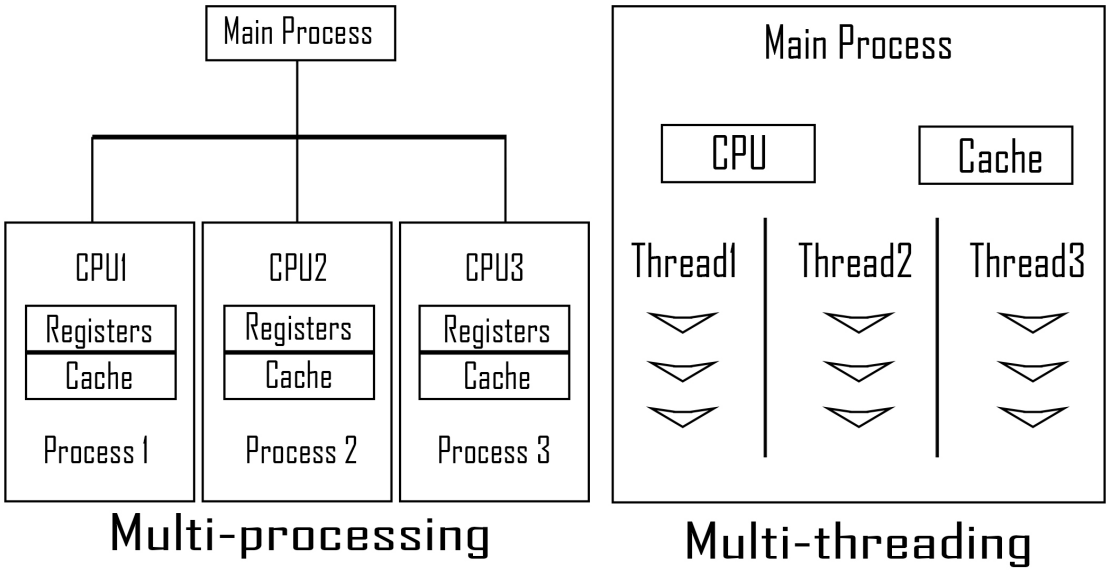

Figure 3.1: Multi-Threading vs Multi-Processing

A multi-threaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution. C++ does not contain any built-in support for multi-threaded applications. Instead, it relies entirely upon the operating system to provide this feature. The header used to implement multi-threading is the "pthread.d",

```
1        #include <pthread.h>
```

that contains the following functions to create, shut down, join and detach a new thread.

```
1        pthread_create (thread, attr, start_routine, arg)
2        pthread_exit (status)
3        pthread_join (threadid, status)
4        pthread_detach (threadid)
```

The use of the above functions in the code needs then for a particular compiler option to be correctly translated in executable code

```
1        $gcc code.cpp -lpthread
```

3.2 Socket programming

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

The application programming interface (API) that programs use to communicate with the protocol stack, using network sockets, is called a socket API. Development of application programs that utilize this API is called socket programming or network programming. Internet socket APIs are usually based on the Berkeley sockets standard. In the Berkeley sockets standard, sockets are a form of file descriptor, due to the Unix philosophy that "everything is a file", and the analogies between sockets and files. Both have functions to read, write, open, and close. In Figure 3.2 the socket communication flow, with both the client and server side and all the function of the connection. In the standard Internet protocols TCP and UDP, a socket address is the combination of an IP address and a port number, much like one end of a telephone connection is the combination of a phone number and a particular extension. Sockets need not have a source address, for example, for only sending data, but if a program binds a socket to a source address, the socket can be used to receive data sent to that address. Based on this address, Internet sockets deliver incoming data packets to the appropriate

application process. Socket often refers specifically to an internet socket or TCP socket. An internet socket is characterized by:

- Local socket address: consisting of the local IP address and (for TCP and UDP) a port number
- Protocol: A transport protocol, e.g., TCP, UDP, raw IP. This means that (local or remote) endpoints with TCP port 53 and UDP port 53 are distinct sockets.

Several types of Internet socket are available:

- **Data-gram sockets** Connection-less sockets, which use User Datagram Protocol (UDP). Each packet sent or received on a datagram socket is individually addressed and routed. Order and reliability are not guaranteed with datagram sockets, so multiple packets sent from one machine or process to another may arrive in any order or might not arrive at all. Special configuration may be required to send broadcasts on a datagram socket. In order to receive broadcast packets, a datagram socket should not be bound to a specific address, though in some implementations, broadcast packets may also be received when a datagram socket is bound to a specific address.
- **Stream sockets** Connection-oriented sockets, which use Transmission Control Protocol (TCP), Stream Control Transmission Protocol (SCTP) or Datagram Congestion Control Protocol (DCCP). A stream socket provides a sequenced and unique flow of error-free data without record boundaries, with well-defined mechanisms for creating and destroying connections and reporting errors. A stream socket transmits data reliably, in order, and with out-of-band capabilities. On the Internet, stream sockets are typically implemented using TCP so that applications can run across any networks using TCP/IP protocol.
- **Raw sockets** Allow direct sending and receiving of IP packets without any protocol-specific transport layer formatting.

Computer processes that provide application services are referred to as servers, and create sockets on startup that are in the listening state. These sockets are

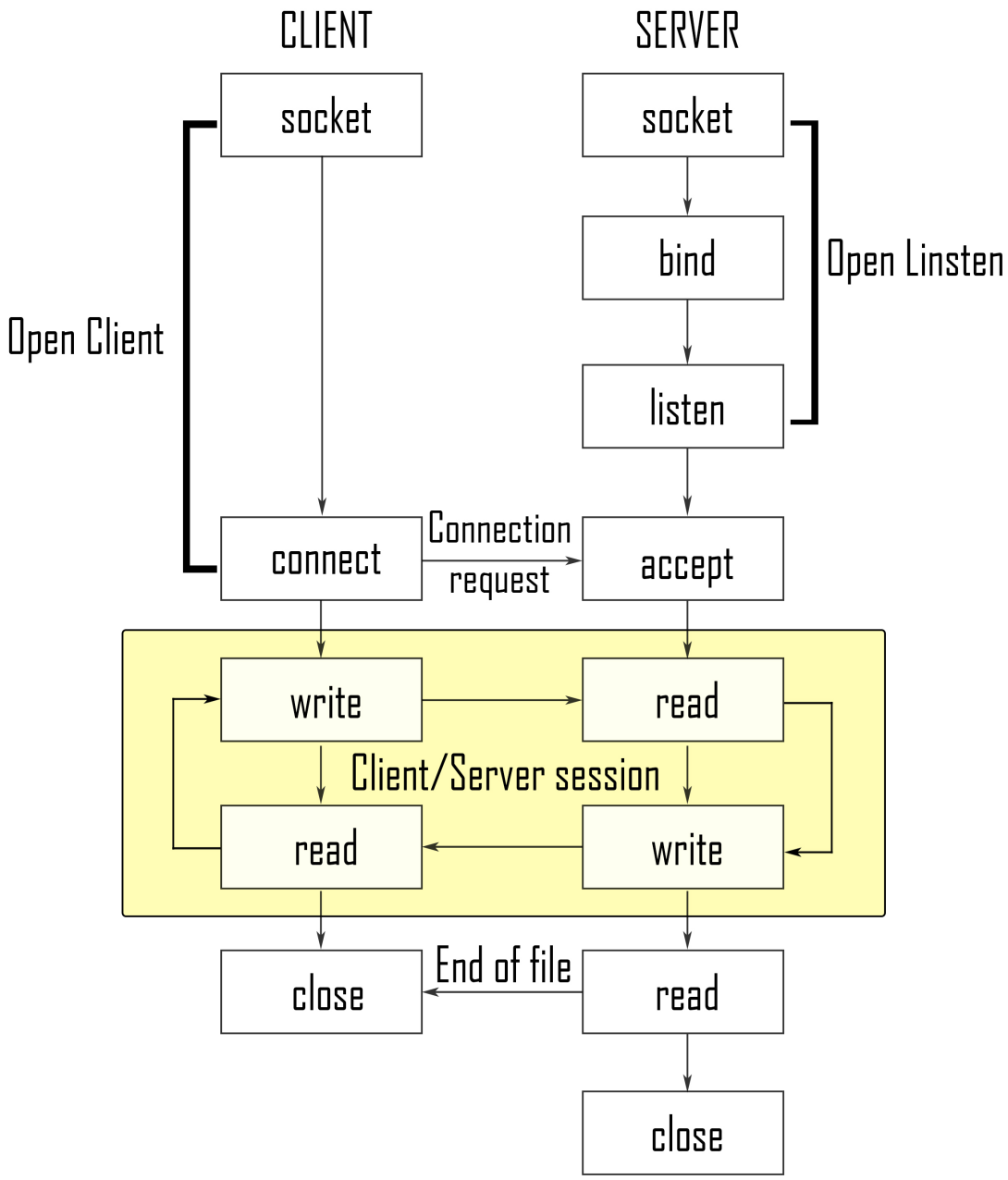

Figure 3.2: Socket communication flow

waiting for initiatives from client programs. A TCP server may serve several clients concurrently, by creating a child process for each client and establishing

a TCP connection between the child process and the client. Unique dedicated sockets are created for each connection. These are in established state when a socket-to-socket virtual connection or virtual circuit (VC), also known as a TCP session, is established with the remote socket, providing a duplex byte stream. A server may create several concurrently established TCP sockets with the same local port number and local IP address, each mapped to its own server-child process, serving its own client process. They are treated as different sockets by the operating system since the remote socket address (the client IP address or port number) is different; i.e. since they have different socket pair tuples.

WebSocket

WebSocket is a computer communications protocol, providing full-duplex communication channels over a single TCP connection. The WebSocket protocol enables interaction between a web browser (or other client application) and a web server with lower overhead than half-duplex alternatives such as HTTP polling, facilitating real-time data transfer from and to the server. This is made possible by providing a standardized way for the server to send content to the client without being first requested by the client, and allowing messages to be passed back and forth while keeping the connection open. In this way, a two-way ongoing conversation can take place between the client and the server.

3.3 Device catalog

Mostly common on the IoT world, the Device catalog, known as resource catalog, is a device registry system. It registers and provides a registry of available devices and the resources they expose. Because of the multiple devices present on the system, this structure is useful to store all the information about devices, such robot status, communication detail (socket number), position and more. Once an entity, such a user or another device, connects to the system, it can retrieve all these information simply requesting them to the system itself. The Device catalog can be implemented in many different ways, both file-based, with a Json file or another type of file storing all the information, or execution-based, with a

variable inside the system code that is allocated and deallocated everytime the server is turned on and off.

3.4 HTML

Hypertext Markup Language (HTML) is the standard markup language devised to allow website creation. HTML can embed programs written in a scripting language such as JavaScript, which affects the behavior and content of web pages. Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document. HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by tags, written using angle brackets. The tags directly introduce content into the page. Other tags surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags, but use them to interpret the content of the page.

3.5 Javascript

JavaScript, often abbreviated as JS, is a programming language that conforms to the ECMAScript specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions. Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. JavaScript enables interactive web pages and is an essential part of web applications. The vast majority of websites use it for client-side page behavior, and all major web browsers have a dedicated JavaScript engine to execute it. As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative programming styles.

It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM). However, the language itself does not include any input/output (I/O), such as networking, storage, or graphics facilities, as the host environment (usually a web browser) provides those APIs. JavaScript engines were originally used only in web browsers, but they are now embedded in some servers, usually via Node.js. They are also embedded in a variety of applications created with frameworks such as Electron and Cordova.

Vue

Vue [3.3] is an approachable, versatile JavaScript framework that helps the programmer to create a more maintainable and testable code base. It is used with CSS and JavaScript because of the great support it gives in the creation of efficient and functional web pages. The core library is focused on the view layer only, and is easy to pick up and integrate with other libraries or existing projects. On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with modern tooling and supporting libraries. In addition, Vue offers a command line interface too, to give the programmer another easy way to manage files and units inside projects.

Figure 3.3: Vue, HTML and JavaScript logo

Chapter 4

Development tools

This chapter presents all the useful development tools used during the project. Because of their strong simulation tool and develop possibilities, the Robot Operating System, together with the Gazebo and turtleBot3 environment, are widely used platforms for the prototyping of robotic projects. With their up to date and continuously developing software, they are one of the best option to develop and test a system.

4.1 ROS

The Robot Operating System (ROS) is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. [FROM ROS.ORG] Although ROS is not an operating system, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management.

ROS processes are represented as nodes in a graph structure, connected by edges called topics. ROS nodes can pass messages to one another through topics, make service calls to other nodes, provide a service for other nodes, or set or retrieve shared data from a communal database called the parameter server. A process called the ROS Master makes all of this possible by registering nodes to itself,

Figure 4.1: ROS Kinetic logo

setting up node-to-node communication for topics, and controlling parameter server updates. Messages and service calls do not pass through the master, rather the master sets up peer-to-peer communication between all node processes after they register themselves with the master. This decentralized architecture lends itself well to robots, which often consist of a subset of networked computer hardware, and may communicate with off-board computers for heavy computation or commands.

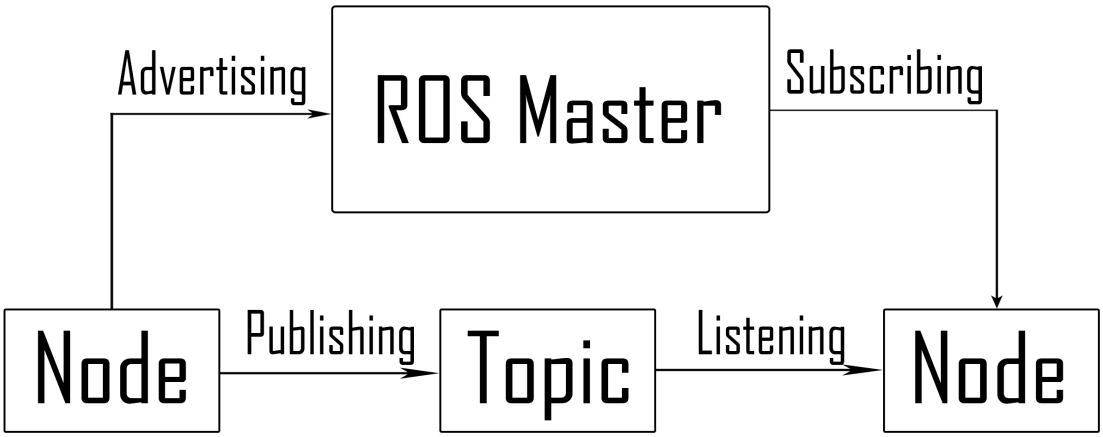

Figure 4.2: ROS Topics example

A node represents a single process running the ROS graph. Every node has a name, which it registers with the ROS master before it can take any other actions. Topics are named buses over which nodes send and receive messages. Topic names must be unique within their namespace as well. To send messages to a topic, a node must publish to said topic, while to receive messages it must subscribe.

The distribution used for this project is the Kinetic, for Ubuntu 18.04.

Lib12c

As shown in Figure 1.7, the Jetson is linked to the RCU and motors with a serial i2c connection. For this reason is necessary to use a As we can see from Figure [X] (chapter of the robot structure) of the internal architecture of the robot, the Jetson is connected to the RCU and motors via an I2C serial connection. For this reason it is therefore necessary to use a special library, called "libi2c" which allows the use of simple functions for the exchange of information through I2C. The libi2c library is an easy and customable Linux user-space i2c library that supports both C/C++ and Python. The library provides read/write/ioctl functions to operate i2c devices:

```
1    i2c_open()
2    i2c_close()
3    i2c_read()
4    i2c_write()
5    i2c_ioctl_read()
6    i2c_ioctl_write()
```

4.2 Gazebo

The simulation of a robot is an essential tool for robotics. Through a simulator you can test algorithms, design robots, perform tests without the need to physically have a robot in your hands. Gazebo offers the possibility to simulate complex indoor and outdoor environments with scrupulous efficiency. The tools available to this simulator are: a robust physics engine, high quality graphics and comfortable programmatic and graphic interfaces. Developed by the Open-Source Robotic Foundation, it is an open source tool.

4.3 Turtlebot3

Turtlebot is an educational, programmable, ROS-based mobile robot created to reduce the size of the platform and price without the sacrifice of functionalities and quality. The project, created at Willow Garage in November 2010, is able to move autonomously, map a 3D environment, avoid obstacles and much more, giving the opportunity to developers and small companies to develop application without caring about the robot physical implementation. Among all the functionalities and features, that are always up to date and compatible with all ROS versions, TurtleBot supports development environment that can be programmed and developed with a virtual robot in the simulation. For this reason TurtleBot and Gazebo are a really powerful prototyping tool and the perfect platform to test the previous discussed communication network.

4.4 Single HTTP server

The "Simple HTTP server" is a python-based web-server implementation able to manage the HTTP requests of a certain web-page exposed to the web. It is ready to use and doesn't need any setup. When launched, the primary function of the web-server is to store, process and deliver web pages to clients. The communication between client and server takes place using the Hypertext Transfer Protocol (HTTP). Pages delivered are most frequently HTML documents, which may include images, style sheets and scripts in addition to the text content.

Chapter 5

Robot communication theory

Before going deeper in the presentation of the work in this chapter a theory description of the robot communication paradigms is shown. The purpose of this part is to give the reader a clear insight on what are the theoretical aspects of the topic, with its difficulties and recent development.

5.1 Robot to Infrastructure

In a Multi-Robot system (MRS) with Robot to Infrastructure (R2I), or Machine to Infrastructure (M2I), communication we define any technology that permits information exchange between the device(s) and the Infrastructure. When we talk about R2I infrastructure we refer to any architecture that provides organizing structure and support for the system it serves. For example Cloud Servers, Local Network Servers or any other physical or logical entity that provides processing, storage and organization power to the system.

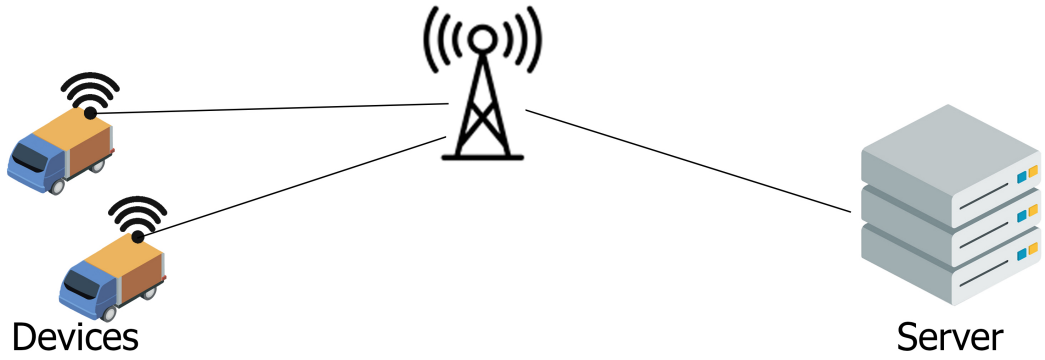

Figure 5.1: Robot to Infrastructure

5.2 Robot to Robot

With Robot to Robot (R2R), or Machine to Machine (M2M) communication, we refer to any technology that allows the exchange of information between devices. R2R networks are very similar to LAN of WAN networks, but are exclusively used to allow machines and sensors to communicate. These devices feed information they collect back to other devices in the network. This process allows a human (or an intelligent control unit) to assess what is going on across the whole network and issue appropriate instructions to member devices.

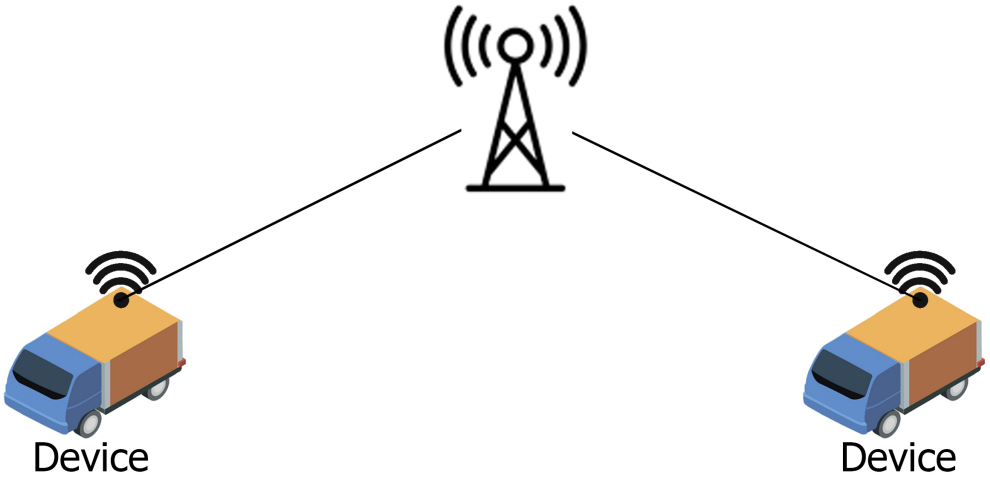

Figure 5.2: Robot to Robot

Under specific conditions, the use of this kind of communication is important to accomplish complex tasks, because the effectiveness of a solution to a single robot task could be, in some cases, improved using coordination among several robotic agents. M2M is among the fastest-growing types of connected device technologies in the market right now , largely because M2M technologies can connect millions of devices within a single network. Technological improvements both in the hardware and in the associated software are two of the key reasons beyond the growing of M2M and MRS. The increased availability and use of complex sensors results in robots equipped with reliable and effective hardware that improve their basic capabilities (laser range finders, cameras, infrared sensors, robotic arms, gripping devices etc.). In addition, the software techniques developed for the robotic applications take advantage of the hardware upgrade and improve complex and reliable solutions for the basic tasks that a robot should be able to perform: localization, path planning object transportation and recognition, tracking, etc. An overview of the specific test-beds which are commonly used in the MRS literature for validating and evaluating the proposed M2M coordination techniques, divided on the abstract task the robot should be able to execute is the following:

- **Foraging and Coverage**: components of the MRS pick up objects scattered in the environment.
- **Multi-Target Observation**: team of robots that detect and track a set of moving objects with the aim to maximize the time during which each of the moving target is being observed by at least one robotic agent in the MRS.
- **Exploration and Flocking**: these test-bed contains all the tasks that require MRS members to coordinate their movements in the environment, like flocking, formation maintenance or map building for example.
- **Box pushing and Object Transportation**: this task requires the robotic agents to cooperatively "move objects" in order to reach a desired configuration. Stockage, truck loading and unloading are analogue problems related to this test-bed. This is the category in which falls the project's robot fleet of delivery devices.

After the above description,a system classification will be now performed to discuss every kind of M2M coordination approach.

- **Unaware or aware systems**: Based on the knowledge of the MRS's members of other members of the team. In Unaware systems each member of the MRS executes its own task without any knowledge about the other members of the team. Because each member of the team does not have knowledge of other robots, the communication among the robotic agents can not be direct. In Aware systems, instead, robots of the team have knowledge of the presence of other robots in the environment and act together in order to accomplish the same goal.

- **Coordinated or not coordinated systems**: A system is coordinated when its MRS members perform actions taking in consideration the actions performed by other robots, in order to accomplish the task. Coordination may be weak or strong. Weak coordination consists in the absence of an explicit predefined coordination protocol (a set of explicit predefined rules wich are followed by all the robots of the system). In Strong coordination systems robot interact according to a predefined coordination protocol.

- **Centralized or distributed systems**: : In a coordinated centralized system, a particular robot (called leader) is in charge of organizing the work of the entire team, while the other members act according to it's directions. In Distributed systems each team member is executing a coordination protocol, while taking decisions in a completely autonomous fashion.

In Figure 5.3, the MRS taxonomy

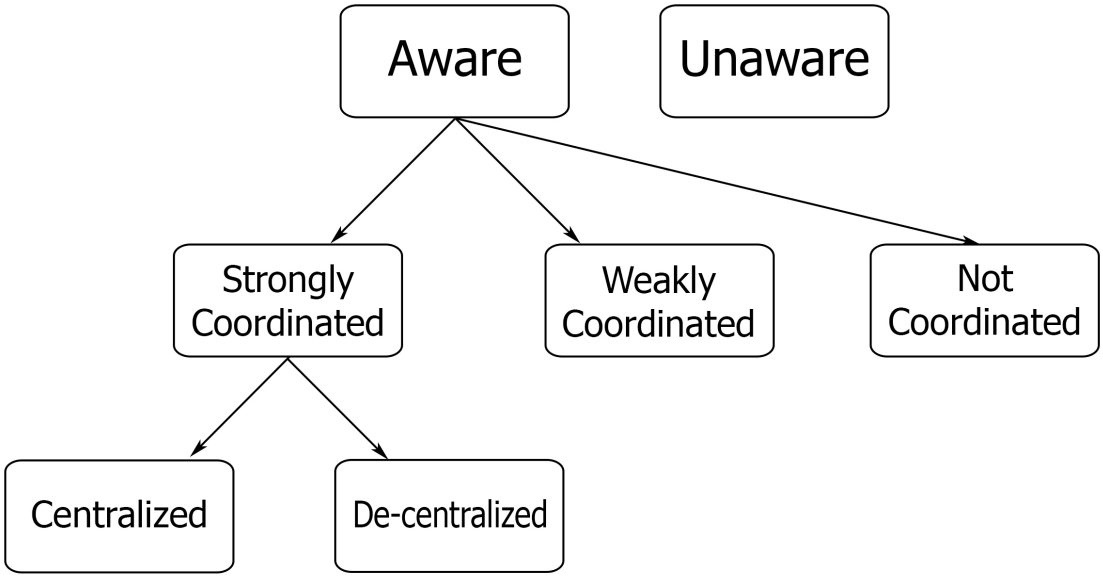

Figure 5.3: MRS R2R communication taxonomy

Chapter 6

Design

6.1 Communication requirements

After the study of the prototypes on the market and the analysis of the due capabilities of the system itself, the three crucial part always present in an autonomous delivery system are:

1. Robot to Robot and Robot to Infrastructure communication.
   Among all the characteristics that a delivery service robot must have, the ability to exchange instructions and information with the outside world, is crucial. In a Internet of Things world where everything is connected to the internet, an autonomous device not able to communicate with the outside is limited and outdated. In a multi-robot-system it is therefore necessary to have an Infrastructure, that provides organizing structure and support, while it is an option to have direct communication between the robots.

2. Fully autonomous robot.
   The starting point to create an automated delivery system is to have such devices that can sense their environment, through the use of radars, lidars, cameras, GPS, and move safely towards a defined destination with little or no human input. SLAM with the mapping and localization, Path Planning algorithm, Autopilot and all the sensors of the robot must then collaborate and exchange crucial information to accomplish the aimed tasks. The robot

internal communication structure is crucial, because an inefficient or erroneous exchange of information between the internal actors of the system can damage the device, objects or humans.

3. User Interface
   Also called UI, is the interaction place between user and the system. Nearly all software programs and services have an User Interface that allow effective operation and control of the machine/system from the human end. The goal of the user interface design is to produce user interface which makes it easy, efficient and enjoyable to operate the service in the way which it produces the desired result.

6.2 Design Architecture

After the definition of the requirements, the next step is to choose how to implement the functionalities of the system. The system structure description will follow the same outline of the three parts mentioned above. First, the Robot to Robot and Robot to Infrastructure communication, then the logical internal structure and after the user interface.

6.2.1 ISO layers

The delivery service has to orchestrate a network of multiple devices, that can work alone or together to execute delivery tasks. The system receives the orders from the user and must consequently send to all the needed robots the instructions to achieve the desired goal. The first design step of the robot communication has been the analysis and study of the different communication protocols and technologies that can be used to connect the single devices to the external world and the internet. For the purpose of the project the kind of communication protocols to take into account are the wireless one, because the robot will be free to move in open space without any physical connection to other devices or infrastructure. Among all the possible choices, the link layer solution adopted was the use of a WiFi connection. Notwithstanding the high energy consumption,

the reduced cost, the simplicity of installation and use are more than sufficient reasons to choose this standard. Another reason why the WiFi standard is the best choice for this project is that the area of the Los Angeles University Citadel is, like most Universities, entirely covered from the university WiFi network. 6.1.

In this way, not only the project doesn't need infrastructure installation, but the robot rarely will miss internet connection. In addition, even if it can happen that the robot exits the WiFi university coverage, it is highly possible that in the area there is an available free WiFi net (that is more and more present in big cities) to which robots can connect. With this network setup, showed in Figure 6.2 the robot can communicate with other robots (M2M) with the infrastructure (Cloud, M2I), and with the user through the internet connection or Bluetooth. To recognize them, every robot has its own unique Identifier (ID number).

The Jetson TX2 uses its own integrated WiFi module to connect to a WiFi network. The module is a Linux-compatible module, able to connect to 802.11a/b/g/n 2×2 867Mbps WiFi networks. The use of this module makes possible the Bluetooth connection too, a good characteristic useful for possible future development. After link layer, the Internet and Transport layer were defined. For the purpose of the project, TCP and RTP were adopted. When turned on, the robots will connect to the server and status, battery level, position and diagnostic data will be sent over TCP communication to the Infrastructure. In the opposite way, instead, the commands will travel from the Infrastructure to the devices. Meanwhile, video streaming will use RTP as transport layer. Regarding the Application layer, HTTP is the standard choice for transmitting hyper-media documents, such HTML. Further details in the code implementation chapter.

6.2.2 Robot to infrastructure

At any moment, a new user can connect to the system and request a new delivery, which the system must serve as soon as possible. For this reason, the system must keep track, minute by minute, about the status of each robot, to be sure of current availability. It is therefore necessary to have the appropriate infrastructure to manage this type of communication. Both distributed approach, with the computation or data management happening on every robot, or a central Infrastructure,

Figure 6.1: California State University Wireless coverage

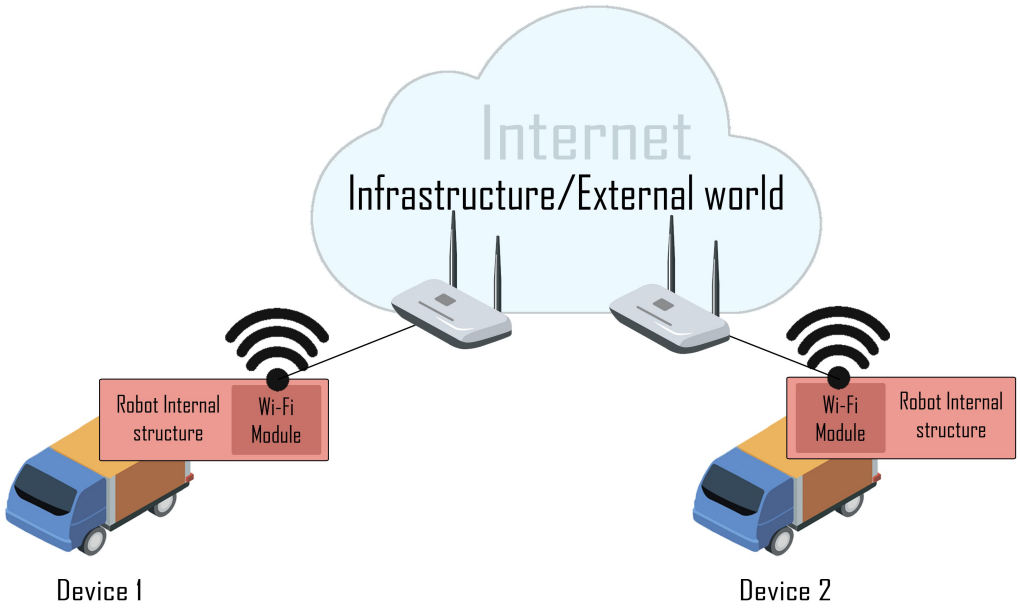

Figure 6.2: Wi-fi Connection

with a unique server maintaining all the connection and collecting all the useful data, are acceptable. Between the two, a study of the University Citadel internal connectivity and a discussion with the InnoTech Engineers, lead to the adoption of the centralized Infrastructure solution, with the implementation of a Server. After an extensive and accurate study of similar systems currently available on the market, the capabilities of the Infrastructure should be:

- **Allow the connection of devices**: through a direct TCP or RTP connection between robot and Server, to have a direct and bidirectional connection channel. Through this channel all the crucial information and data are sent, battery level, availability, time left to the termination of the current task, video feed etc.
- **Keep track of the service robots currently connected**: storing the updating information of the individual devices connected to the network and keeping track of them during the time in the device catalogue.
- **Expose the Web page to the internet**: directly providing all the data

necessary for the users' requests, avoiding the overloading of single devices. This part will be better discussed in the following chapter [indicate web page chapter].

- **Create a interface between the robots and the internet**: with the infrastructure located in the middle, as a bridge for commands and status data, the delivery system has enhanced security and robustness. Any communicative error or intrusion is managed and solved without consulting the robots.

- **Collect diagnostic and congestion information of the environment**: the mapping, localization, congestion and logistic data can be useful information stored inside the Infrastructure cloud, to diagnose and solve problems inside and outside the system.

The server, always running, will be available on a fixed IP on the internet and every delivery robot, once powered on, will connect to it, establishing a TCP/RTP socket connection.

6.2.3 Internal structure

With the previously mentioned architecture (Chapter 1 – Project – System architecture) we shown the physical connection among all the robot parts. Instead, between the computational units, such SLAM, PathPlanning and Autopilot, we must define the way how these parts exchange information and data. To create such communication, adopted solution was the use of ROS (Robot Operating System), a widely used framework for writing robot software. After taking in consideration all the needed information exchange among the robot internal actors, the structure of the topics was defined, using ROS predefined messages. Topics detail description in the following chapter. While SLAM and Path Planning can communicate directly with the creation of a topic between them, the communication with the RCU has to overcome the hardware separation of the I2C link. For this reason the `"i2c_node"` ROS node was created, with the purpose of interfacing correctly the RCU with the other actors inside the Jetson. While the ROS communication

is entirely managed by the open-source software and there is nothing to worry after the definition of the topics and types of messages, the i2c connection needs a common message interface to send significant information. For this reason a dedicated communication interface has been created. For proprietary reasons the implementation structure of this part can't be shown. In Figure 6.3 the high level logical internal structure of the topics that run inside the robot.

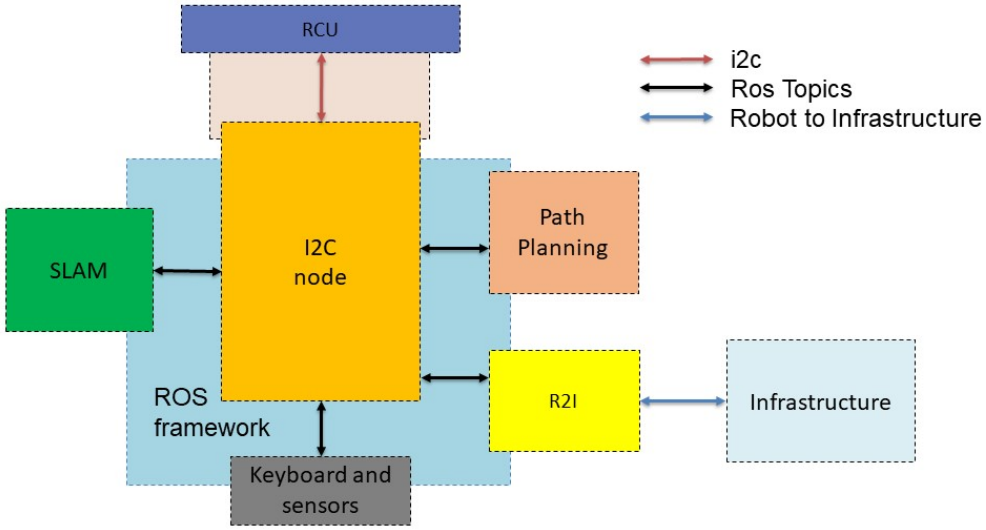

Figure 6.3: Internal robot topics

6.2.4 User interface

The User interface, in an autonomous delivery system, is mandatory. Without an easy and intuitive way to request a delivery, the entire system is pointless. The attention on the design of the user interface, therefore, aims to maximize user experience, with the purpose of making user interaction as simple and efficient as possible. Aiming to versatility and compatibility, the use of a Web-page based interface was the option selected for the system. Nowadays, every device has a browser installed, and there is no need of other application or setup. The core element of the web page, written in HTML, will be then supported by the use of

a JavaScript code, to transmit information from/to the server. Because of the limited time, in this first prototype phase the web page has a really basic set of functionalities and misses authentication/security mechanism, but in the future further functionalities and a security layer will be added . The web page, as its current status, gives users the possibility of:

- Check robot availability.
- Select one of the devices connected to retrieve its status, informations and video feed.
- Control the selected device (Forward, Backward, Left, Right).
- Request a robot movement from point A to point B, like from a store to the final destination.

In Figure 6.4 the proposed design of the page.

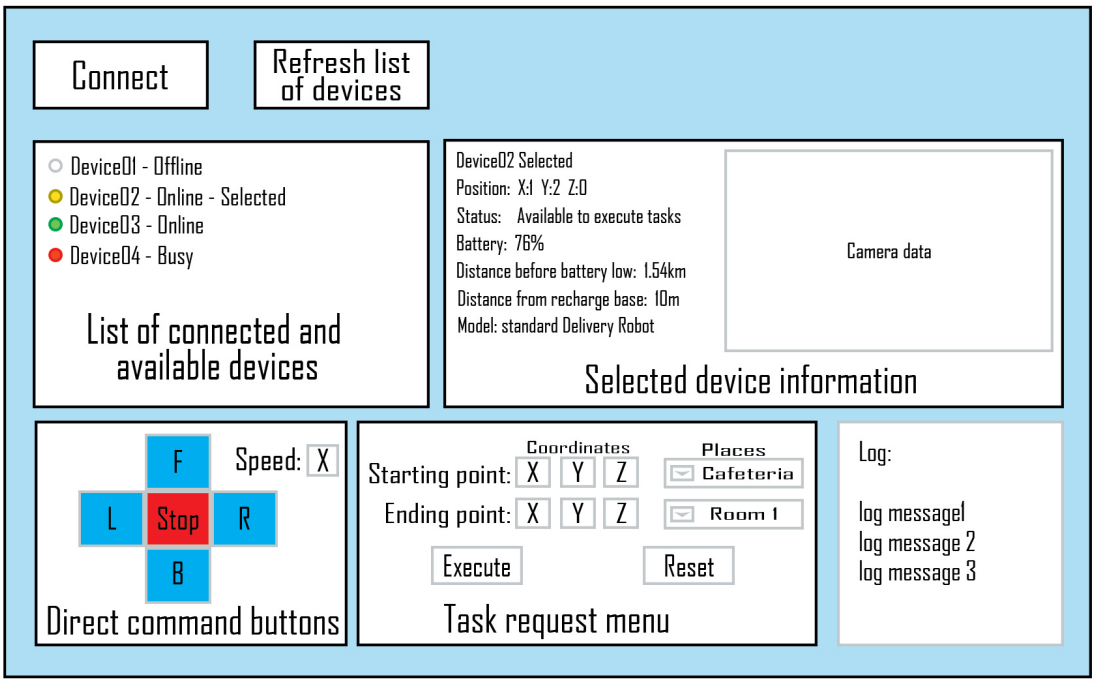

Figure 6.4: Designed Web Page

6.2.5 Design Architecture

Now that every design decision was defined, a simplified point of view of the entire system will be given, having the aim of clarifying the overall system structure.

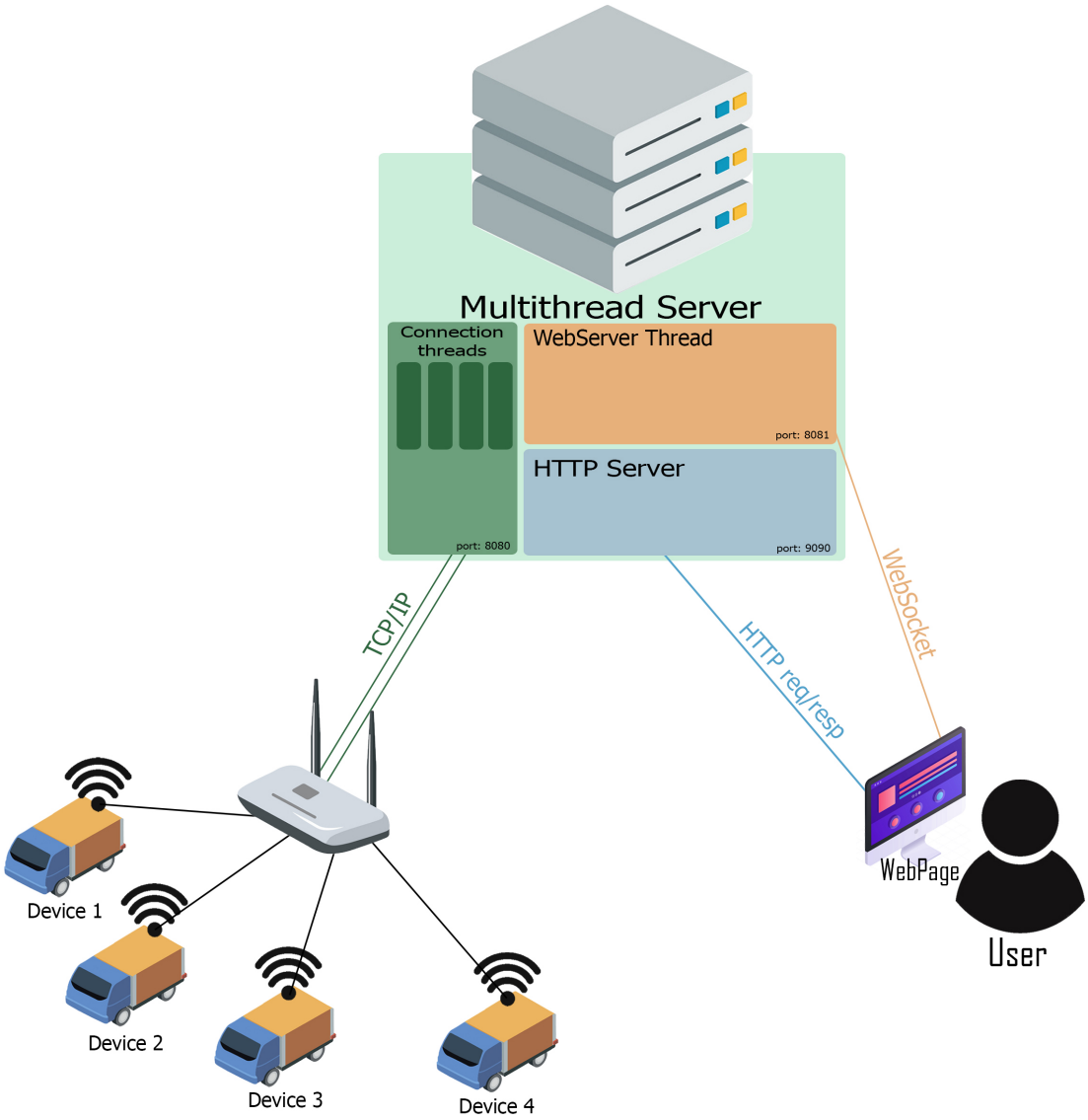

Figure 6.5: Overall structure

Preliminary setup

Server side:
The server is turned on. With a multi-threaded structure, the server exposes the page to the internet, waiting for user's web-page connection. In addition, it waits for robots TCP connection and initialize the device catalog of the system. The server is now ready to manage the requests coming both from users and from devices.
Device side:
When a device is turned on, the robot client-side part of the code is launched. The device makes a TCP connection with the infrastructure and the communication between the server and single robot is established, through a bidirectional socket connection.
User side:
Once the user opens the system site, the browser requests the page, and it is provided back from the server. The browser and the server establish a WebSocket connection and the list of current devices available on the system is sent and showed to the user.

A simplified communication flow is now presented (Figure 6.6):

1. With the selection of one of the devices, the user's web-page requests the device information and status.

2. The server then sends back the requested data of that device, and now the user can move the robot, with a path or direct movement command.

3. Once the user confirms a task to execute, the web-page sends a message to the server, with a special pre-defined format.

4. The server receives and checks the command, if it is correct, it is forwarded to the designed device through the TCP connection.

5. The robot receives the command and now it can execute the task.

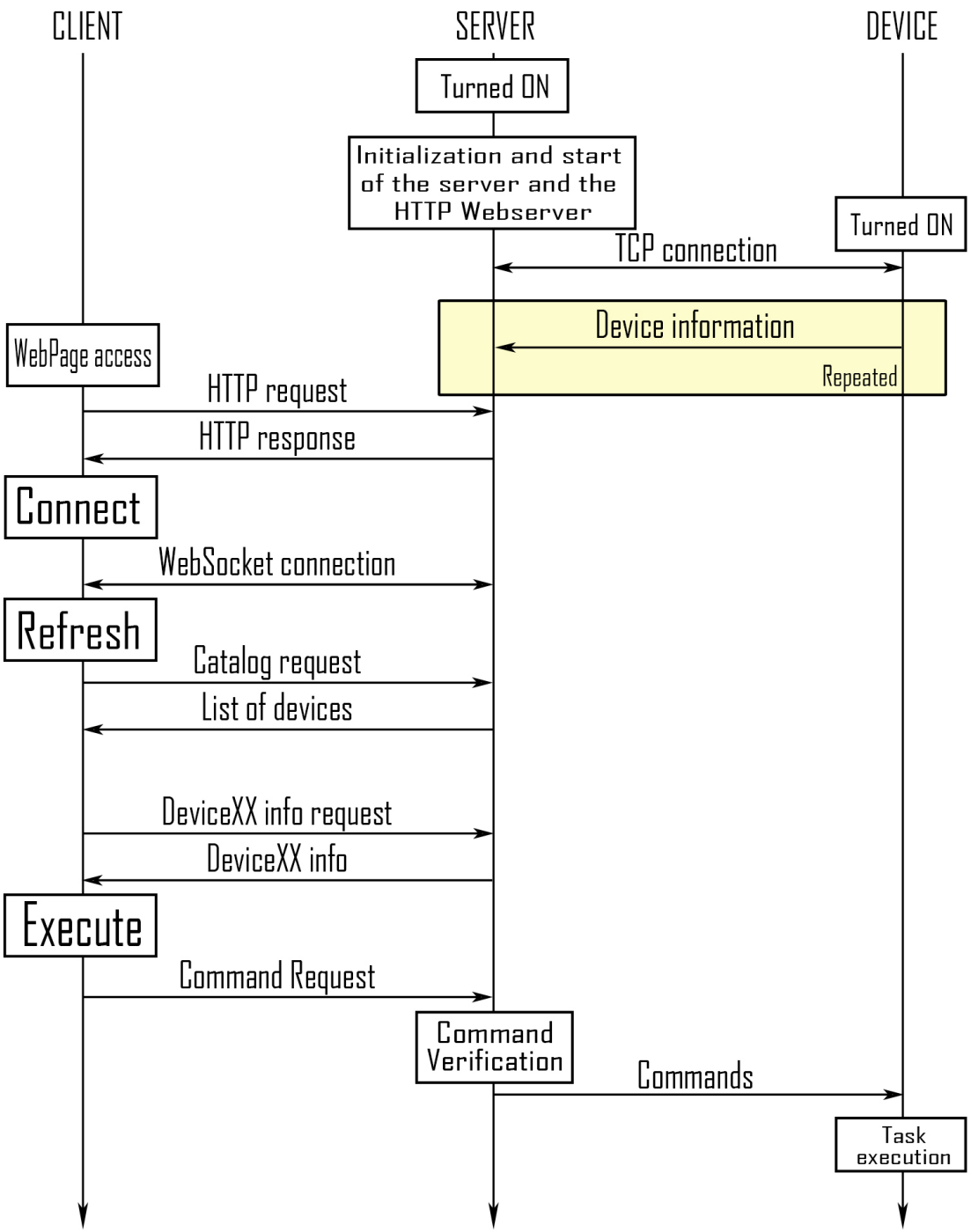

Figure 6.6: Overall system message flow

Chapter 7

Implementation

7.1 Robot Internal Communication

The kind of information exchanged between the robot internal actors are mainly of three kinds:

- Points in the space (robot localization, mapping, path planning)
- Images (Map of the environment created from SLAM)
- Simple numbers (Speed, status, operative modes)

Using both standard ROS messages and creating new ones, for each kind of information above, a different ROS message type was chosen.

The topics codes were then created, taking in consideration every information exchange between the internal components of the single device: map, localization information, images, movement commands from the RCU to the motors etc.

R2I and Path Planning

Between R2I and Path Planning codes the information exchanged consists in the start and end points of the path that the robot receives from the Infrastructure. Once the user uses the WebPage to request the execution of a task, the R2I code receives the starting and ending point (and middle points if needed) of the path that must be followed by the robot.

Path Planning and I2C/RCU

The topics that Path Planning and RCU share, with the i2c node in the middle, permit the exchange of two kind of data. Robot speed and status, sent from the RCU to the Path Planning, and Path Planning status and Pose, in the opposite way.

Slam and I2C/RCU

With the same logic of the Path planning to RCU topics, the SLAM exchange Pose, Status, Speed and Mode messages with the RCU, through the i2c node. The only difference is that the SLAM has two separated topics, one for Mapping pose and one for Localization, to sent to RCU.

7.2 Robot to Robot

For the purpose of Object Transportation tasks, the coordination among the devices can be either unaware, simple but inefficient approach, or Aware Coordinated, complex but effective. For the specific scope of the InnoTech project, the coordination approach is the aware-coordinated-distributed one. The robots know about the presence of other devices in the environment, explicitly taking in consideration their action to accomplish different tasks together. Despite this, every robot takes its own decisions in the micro level of movement, avoiding obstacles and defining its own path to the destination. Because of the prototyping state of the project, the real implementation of these codes was skipped, InnoTech system plans to complete this kind of communication once the single robot system is correctly working. Anyway, from the research and study of these communication, the kind of information that the devices exchange to coordinate is of different types:

- **Status information**: Devices send their status information (position, battery level, assigned activity running) periodically to all robots in the network in order to have a clear and updated description of the surrounding area.

- **Help and recharge request information**: In the case a robot is unable to move, it is in a emergency status or it is unable to reach the recharge station without running out of power, the device sends an help request to other robots. Among all devices in the network, the task-free, closest device will then start the recognition procedure.

- **Congestion avoidance and logistic information**: Logistic data, like temporary jam in a location or dead-ends, are shared among near devices, with the objective of optimizing paths creation.

- **Forwarded data when outside network scope**: The critical information that must be sent or received to the robot while it is outside the Wi-Fi scope, is forwarded through neighbour robots instead, which work as a "bridge" for the communication.

7.3 Robot to Infrastructure

7.3.1 Cloud server

From the previously mentioned requirements, this chapter describes the details of the Server implementation. In the system, the server handles all control requests from clients, processes them and forwards the translated data to the robot. For the feedback data, the server periodically retrieves information about the status of the robots and it transmits these data to requesting clients. The status data include the battery level, robot position, speed, current task and congestion data when a problem occurs. In the system, the server is implemented as a multi-threaded TCP server written in C++. The communication between the server and client was established through sockets, an abstraction that represents a terminal for communication between processes across a network. Because of the prototyping phase of the project, the coding approach used aims to maximize future update and troubleshooting possibility. For this reason, with the use of multi-threading programming, the code threads manage separately every part of the communication:

- Catalogue manager thread
- TCP/IP Connection handling thread
- Data to/from devices thread
- Video feedback thread
- Web socket and User request thread

The code will be now presented briefly with some coding details. Because of the use of WebSockets, TCP Sockets and multi-threading structure, some special libraries were included. Then, the ports were defined, one for the TCP connections, one for the data communication and another for the WebSocked connection. To store the device information, a struct has been created, containing all the crucial information of a single device. With a vector of this struct the system will keep updated all the devices information, store battery Level, status, position and connection information. The main code of the server will launch all the above mentioned threads:

- WebSocket Thread:
  This thread set up the Websocket connection for the WebPage, creating the Web connection and managing: direct movement commands from the user, data request, catalog request, video camera request, delivery requests. This part of the code is in charge to send to the User the list of connected delivery devices and, for every device, to provide status information. On the other side, once the WebPage requests are sent to the system, this part of the code verifies and evaluates the messages, forwarding such commands to the devices. The thread contains the try/except procedure too, to avoid any anomaly in the connection between server and Webpage.
- Create connection Thread:
  This thread manages the connections between the server and the single device. This kind of connection exists to ensure that the devices are correctly connected to the server, waiting for commands. The code manages every TCP device connection to the system, creating a new thread for every

communication. Every device, after every setting-defined number of seconds, uses the created connection to refresh its entry on the catalog with battery level, current status, position and current task. With this procedure, the catalogue contains the most recent status received from the device. Once the connection is established, this thread stores the socket information on the catalogue, maintaining the unique correspondence [Device ID – socket], to make the server-device communication easier to implement. The catalog can be stored simply in an execution variable, allocated and deallocated when the server starts and stops, or in a designated file, in the case this information is needed by other entities inside the system. All the catalog write and read procedure are then secured with semaphore/mutex synchronization techniques to avoid errors or inconsistencies in the data structure.

- Create Data Connection Thread
  This is the Data thread. The working principle is the same of the Alive thread. The RTP connection is managed inside this part of the code, with a different socket port. Once the user requests the video feedback of a particular device, the connection is established and maintained. This connection makes possible the video feed exchange.

- Catalog Refresh Thread
  This thread manages the deletion of old entries in the device catalog. When one entry inside the catalog is older than a certain period (adjustable from the setting file) it is discarded.

Client side of the code

The client side of the code, that will run in every serving device, is a multi-thread C++ code. The coding strategy used aims to maximize future code updates. The code consists of two threads. One that sends status information in a continuous way, once the TCP connection with the server is established, to signal the device presence to the server, and the other one which reads all the TCP socket connection messages received and acts accordingly with them. This code acts as a bridge between the external world and the internal part of the robot. The incoming

messages are analyzed and correctly sent to the internal components of the device. The code will be now presented in its parts with some coding details. After the includes, that are equal to the server code, the communication port is defined. The device ID is hard coded in the device setting file. This identifier is set by the system owner and it is stored in an internal variable. The core code part consists in the launch of two threads:

- Create and refresh connection thread.
  The TCP socket connection is created, binded to the designed port and launched. Once the connection is established, with a while loop the thread sends the device status information to the server. An interactive exit command permits to close the connection without creating any dangerous uncatched exceptions.

- Read incoming messages thread.
  This thread waits continuously on the read function of the socket connection together with the server. When a message arrives, it is evaluated and executed. In the case of command messages, they are forwarded to PathPlanning algorithm through the designed ROS topic.

7.3.2 User interface, Web page implementation

The proposed webpage solution is composed by the HTML and JavaScript files, that aim to satisfy every requirement previously mentioned. For the prototyping phase, the webpage is exposed with a simple HTTP python-based server module and runs in a virtual environment.

Codes

With a simple row division, the page presents to the user the list of the connected devices and the connect/refresh buttons. The HTML index code does not contain any special communication detail of the system, but when the user makes an action on the page, the corresponding JavaScript method is invoked. The code manages the connection to the server, sending and receiving messages through

a WebSocket connection. The communication between the server and the page happens sending messages with a defined format.

The methods of the JavaScript file are:

- **Startup function**. Initialization routine that prepares the webpage for the correct use of it. This method is invoked with the body "onload" function <body onload="app.startup()"> at the very beginning load of the page.
- **Connect function**. The method, invoked when the "Connect to the Server" button is pressed, starts the webSocket connection with the server. Once the connection is established, the device selection menu is enabled and usable.
- **Refresh function**. It is a method invoked to refresh the list of currently connected devices on the systems. The function sends a request to the server, asking for the Catalog information of the system. Once the data is received, the connected devices are shown.
- **Selection function**. This method is invoked when the user selects one of the devices on the list of available robots. The function sends to the server a request, asking for the status information and data of the selected device. The information are then showed in the "Device information" section of the webpage. Once the device is selected, the task request part of the webpage is activated and ready to use.
- **Movement functions**. "forward", "stop", "backward", "turnLeft", "turnRight" methods used to send the direct movement messages to the server.
- **Path function**. After the user inserts the XYZ coordinates of the starting and ending point, this method sends the path information to the server. In the case the user inserts just the destination coordinates, the starting point of the path is assumed to be the current robot position. This functionality simulates the need of the user to send a robot to position XYZ to grab food or drink. In the future phases of the project these coordinates will be substituted by real locations like bars and restaurants.

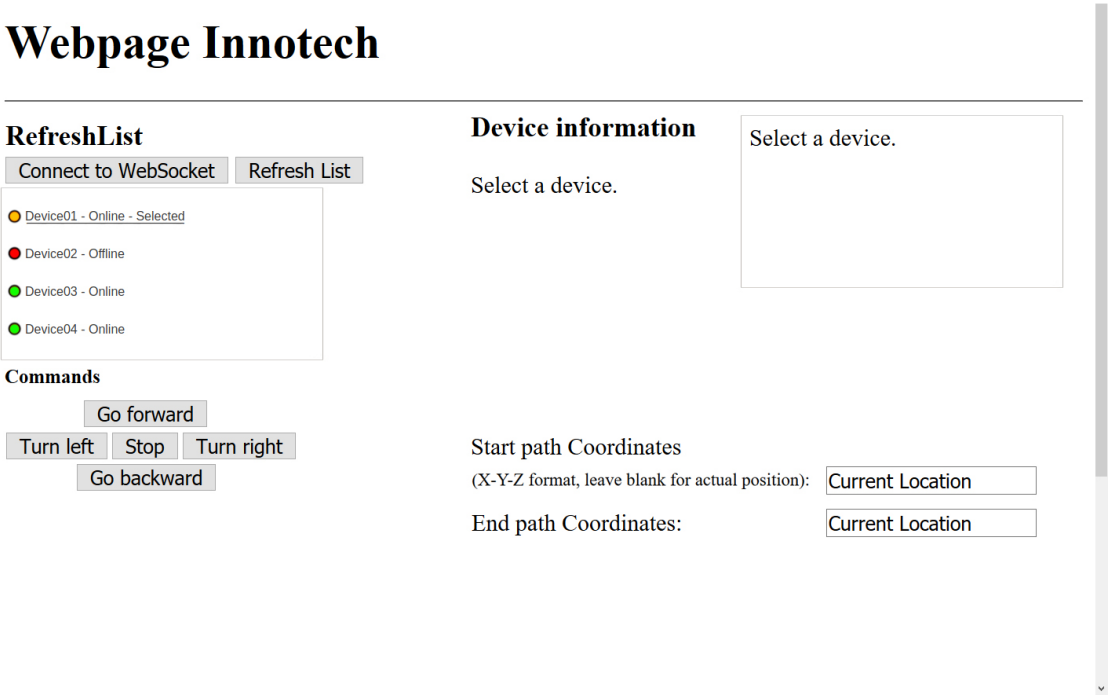

Figure 7.1: Webpage implementation

Chapter 8

Testing and Simulation

8.1 Simulation

The simulation will take place in the gazebo environment, where turtlebot3 will be used as delivery device inside the "turtlebot3_basic" world. With Server, WebPage, Slam, Path Planning and client codes running, the user connects to the system and requests a delivery. Once the request is received from the server, it is forwarded to the device and the internal communication among the internal robot actors starts, moving the device autonomously from the starting to the destination point. The turtlebot3 gazebo simulation will be used only as a visual feedback to correctly time the whole operation, from the user request to the completed delivery. The system architecture used to conduct these types of tests is presented in Figure 8.1

The server, along with the HTTP server too, will run in one computer connected to the internet with a fixed public IP. The device codes will run in the Jetson module, connected to the internet with a Wi-Fi connection different from the server one. To recreate the device internal ROS communication, inside the Jetson module the robot movement will be simulated using the Gazebo environment, with the turtlebot3 "burger" robot moving in the "turtlebot3-world" map. Unfortunately, during this part of the project the Path Planning and SLAM algorithm were not completed. However, the communication system can be simulated and tested anyway using the standard turtlebot3 navigation algorithms.

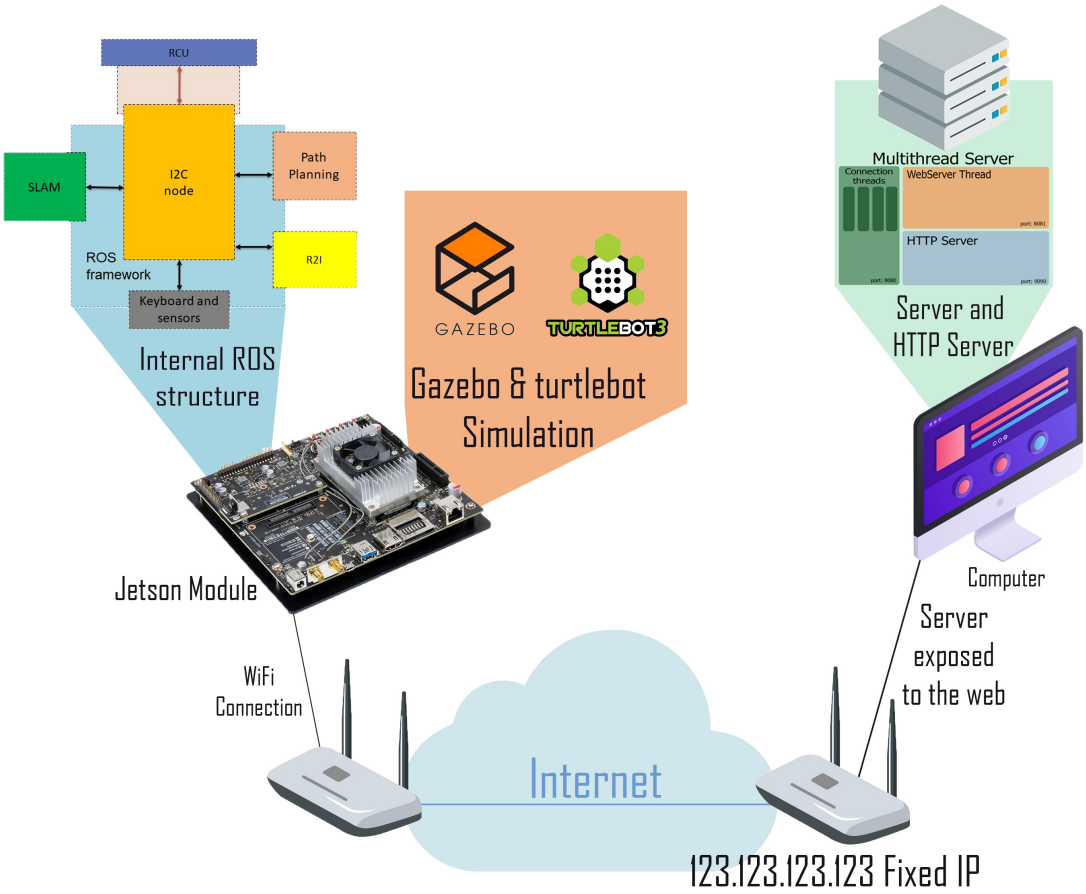

Figure 8.1: Testing structure

The timing of the simulation and testing procedures will be performed using the std::time function. The function, defined in header "ctime", returns the current calendar time as seconds since the Epoch, aka 1/1/1970. With the help of a script that automates the whole procedure and with two timing functions placed before the user click and after the completing of the requested task, the difference of the two times we get will be used to evaluate different scenarios. The setup procedure is the following:

- Server side:

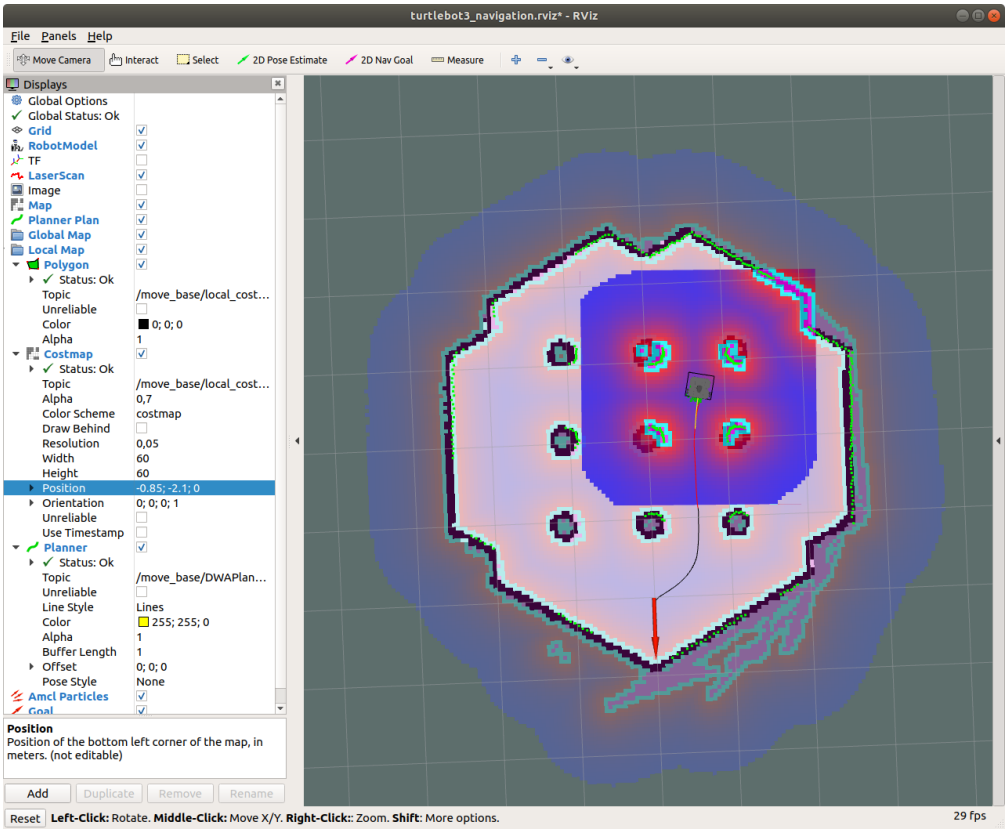

Figure 8.2: Navigation example in the Gazebo and Rviz environment

```
$ sudo ./server
```

HTTP server start in virtual environment

```
$ source venv/bin/activate
$ cd ../webpage
$ python3 index.html
```

- Client side:

Client code start

```
1        $ sudo ./client
2
```

Gazebo environment launch

```
1        $ roscore
2        $ roslaunch turtlebot3_bringup turtlebot3_robot.launch
3        $ export TURTLEBOT3_MODEL=burger
4        $ roslaunch turtlebot3_navigation turtlebot3_navigation.
    launch map_file:=$HOME/map.yaml
5
```

The system is now ready to accept and complete the user requests coming from the webpage.

8.2 Testing

Once the communication system is fully working, a verification step will be performed, to underline possible weakness or operative situation difficulties of the system. The type of verification performed is focused on the server testing under stress conditions. From the protocol stack used for the implementation of the system, the size of the message header sent between the server and the user are the standard TCP-IP-Ethernet 20+20+24 Bytes. With the payload, the total message size goes around 80 and 100 Bytes per message.
The kind of verification performed are:

- Latency testing with a single user and a single device connected
- Latency testing with multiple users connected to the system.
- Latency testing when multiple robots are operating in the system.

Latency testing with a single user and a single device connected

The first step to evaluate the system operative conditions was to simulate the easiest scenario possible. With only one device available and one user connected to the system, it has the less computational stress possible and the latency of the procedure can be tested. The test will consist in a request from the user to move the robot from its current location to another. With the aid of the timing functions, the time between the user request and the task execution will be calculated. In this way, the latency of the system will be evaluated.

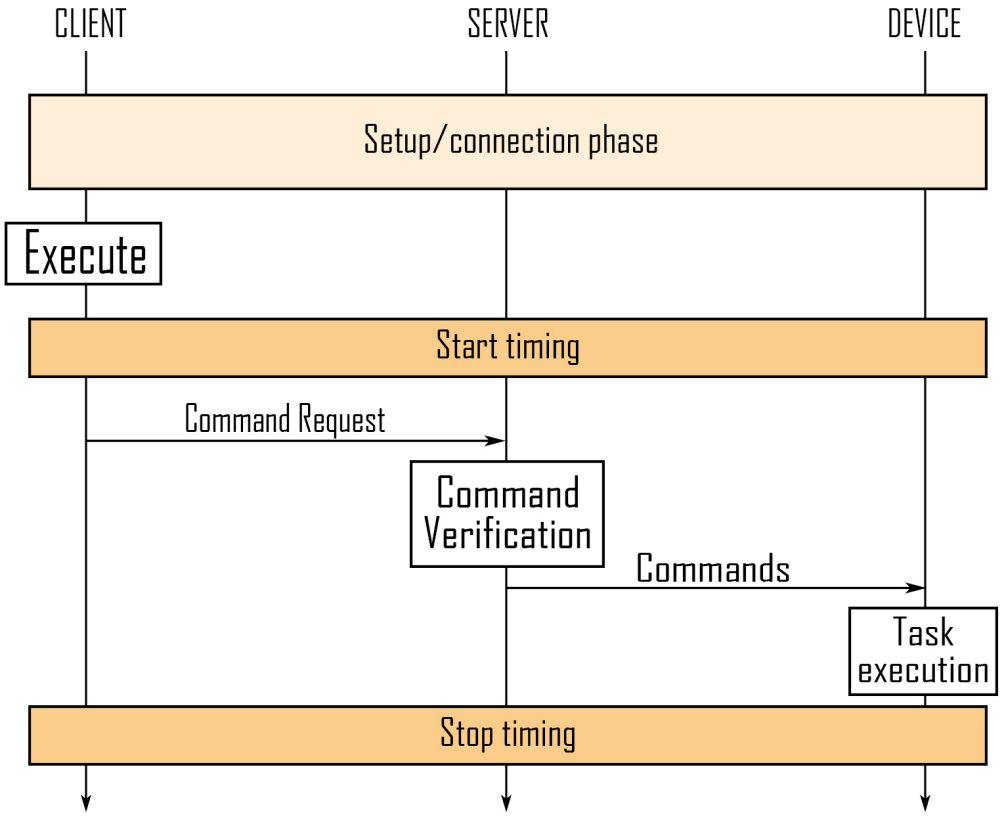

Figure 8.3: Timing description of the test

In Table 8.1 the results.

The latency results are consistent, with the results not moving too much away

Latency testing with a single user and a single device connected			
Test n.	Latency (ms)	Test n.	Latency (ms)
1	70	11	70
2	74	12	72
3	83	13	74
4	71	14	78
5	71	15	71
6	80	16	79
7	79	17	72
8	77	18	82
9	72	19	87
10	77	20	68

Table 8.1: Latency testing with a single user and a single device connected results.

from the average. Avg = 75.35ms . In this prototyping phase of the project it is clear that the use of TCP communication, reduces the system performance. However, the latency is acceptable for the delivery purpose nature of the system because no hard real-time procedures are performed.

Latency testing with multiple users connected to the system

The tested scenario is out of the control of the system owner. At any time, a high number of users can connect to the system and request a task to execute, resulting in an overcharge of the server itself. One simple solution to this problem could be to limit the concurrent number of users that can connect to the system to avoid a-priori some unwanted behaviours. For the purpose of this test, the scenario of 10, 50, 100 users connected concurrently was studied, with the following results.

Avg = 75.55ms. The results prove that the overcharge expectation of the server is not really present in the testing environment. The small messages sent through the TCP connections do not create heavy loads and the system is able to manage them correctly. This result is in line with the computational capabilities of the computer where the server codes are running. The computer has a strong computational power. For this reason it can manage every connection, thread, computation, without running out of memory or CPU power. In a real-world

Latency testing with multiple users connected to the system			
Test n.	Latency (ms)	Test n.	Latency (ms)
1	82	11	70
2	85	12	66
3	88	13	67
4	79	14	75
5	84	15	83
6	83	16	82
7	69	17	83
8	65	18	70
9	82	19	66
10	69	20	65

Table 8.2: Latency testing with multiple users connected to the system results.

scenario, however, with the server deployed on a cloud structure, for example, the computational throttling of the Platform as a Service (PaaS) can be an upper limit for communication performance, resulting in slower service.

Latency testing when multiple robot are in the system

The last testing scenario is the least dangerous possible. The devices inside the system are directly under the control of the system owner, so there will never be an unexpected high number of devices concurrently connected to the system. However, thinking with a scalability approach, this latency test is really useful to test the limits of the system itself. The studied scenario will consist in 10, 50, 100 devices connected to the system. With a script, all the client codes will be launched inside the Jetson board and the simulated devices will connect to the system. The script launches the single client codes with an homogeneous period, because it is really unlikely that all the robots will be turned on in the same instant. For the purpose of this test, because the simulation of the Gazebo environment is a really heavy computational task, only the communication time will be measured, from the web-page task request, to the robot message reception, ignoring the movement time from point A to point B.

Avg = 79.55ms. The results are satisfactory. The system can manage the heavy test load, mainly consisting in all the multiple status messages sent from the

Latency testing when multiple robot are operating in the system			
Test n.	Latency (ms)	Test n.	Latency (ms)
1	83	11	85
2	71	12	71
3	90	13	87
4	87	14	73
5	74	15	71
6	83	16	86
7	72	17	84
8	89	18	76
9	76	19	77
10	83	20	71

Table 8.3: Latency testing when multiple robot are in the system results.

devices to the server. To stress the system even more , a >100 devices simulation can be conducted, but because it's a rare situation in a campus to have such high number of autonomous devices concurrently it won't be performed.

Chapter 9

Final remarks

The final chapter is divided two main sections. The former focuses on the future steps of the communication system development, central topic of this dissertation. The latter concludes the discussion of the works of the project.

9.1 Future works

Latency

As the testing results proved, the latency of the current communication system implementation is in line with the requirements of this kind of system. Because all the autonomous movement decision happen inside the robot (collision avoidance, path planning, localization and mapping) there is no hard real time information that need to be exchanged between the system and the single robot. The average 77ms time from the user's task request to the actual command execution start is an acceptable time for the delivery purpose of the system. However, if in the future the company wants to have better latency performance, a valid option can be the use of the mobile data network. With the wide spread of 4G antennas and the newborn 5G coming in the next years, the use of a mobile network to connect to the internet can be the best solution to be adopted to minimize latency. The robots can have even more flexibility to move, not being forced do be connected to a WiFi network.

Robot to Infrastructure Communication

Regarding the Robot to Infrastructure communication, another option for the project is to remove the server and use a decentralized infrastructure. This strategy can be a good solution to enhance the system robustness. The mapping, localization, congestion etc. information can travel directly from one robot to another. In this way, for example, the SLAM can be implemented at an higher level and the path planning can be more efficient because of the shared information. Anyway, this kind of system needs a lot of effort to be implemented. The amount of work needed to create these kind of systems is way higher in respect to the simple centralized infrastructure one.

Webpage

The webpage presented in this paper is a really basic page. The number of functionalities that gives to the user is limited and all it does is to take every request and to execute it without any authentication step. For this reason, once the system will be complete, an authentication layer will be added to the system. In addition, a security layer will be added too, to avoid malicious external attacks. The visual appearance of the page will be improved, to give a pleasant view to the user.

Drones

The discussion presented in this work refers only to ground devices. The implemented solution is instead robustly coded in a way to permit the 3D movement of a drone with the same programs and system architecture.

9.2 Conclusions

The work of this thesis regarded the design and implementation of internal and external communication systems for a fleet of autonomous robots. Thanks to the continuous interaction with the InnoTech colleagues during the duration of the work, the project is growing and the company aims to create the first

working prototype for early 2021. The difficulties caused from the Covid-19 pandemic, with the impossibility to travel, made the coordination difficult. Every decision and discussion had to overcome the time zone between Italy and America. However, with the hard work of every person in the team, the achieved results are satisfactory. It is worth to say that the development phases described in this thesis represent only the initial stage of a design that needs much more time and effort, considering the fact that the company had an idea and started developing the robot basically from scratch. The decision adopted for the Robot to Infrastructure communication, together with the codes implemented, aimed just to implement a working prototype as soon as possible. A lot of features will be added in the near future, to make the system robust and fully functional.

Bibliography

[1] Erico Guizzo. *What Is a Robot? Top roboticists explain their definition of robot.* URL: https://robots.ieee.org/learn/what-is-a-robot/. (accessed: 09.20.2020) (cit. on p. 1).

[2] *Robots and robotic devices vocabulary.* URL: https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:en. (accessed: 09.20.2020) (cit. on p. 1).

[3] *Executive Summary World Robotics 2019 Service Robots.* URL: https://ifr.org/downloads/press2018/Executive_Summary_WR_Service_Robots_2019.pdf. (accessed: 09.20.2020) (cit. on p. 1).

[4] *Starship website.* URL: https://www.starship.xyz/. (accessed: 09.20.2020) (cit. on p. 3).

[5] *KiwiBot website.* URL: https://www.kiwibot.com/. (accessed: 09.20.2020) (cit. on p. 4).

[6] *Robomart website.* URL: https://robomart.co/. (accessed: 09.20.2020) (cit. on p. 5).

[7] *Teleretail website.* URL: https://teleretail.com/. (accessed: 09.20.2020) (cit. on p. 5).

[8] *Amazon Robotics website.* URL: https://www.amazonrobotics.com/#/. (accessed: 09.20.2020) (cit. on p. 6).

[9] *Fedex Roxo website.* URL: https://www.fedex.com/en-us/innovation/roxo-delivery-robot.html. (accessed: 09.20.2020) (cit. on p. 7).

[10] *InnoTech website.* URL: https://www.innotech-sys.com/. (accessed: 09.20.2020) (cit. on p. 7).

[11] «IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture». In: *IEEE Std 802-2014 (Revision to IEEE Std 802-2001)* (2014), pp. 1–74 (cit. on p. 18).

[12] *User Datagram Protocol*. RFC 768. Aug. 1980. DOI: 10.17487/RFC0768. URL: https://rfc-editor.org/rfc/rfc768.txt (cit. on p. 25).

[13] *Specification of Internet Transmission Control Program*. RFC 675. Dec. 1974. DOI: 10.17487/RFC0675. URL: https://rfc-editor.org/rfc/rfc675.txt (cit. on p. 25).

[14] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550. July 2003. DOI: 10.17487/RFC3550. URL: https://rfc-editor.org/rfc/rfc3550.txt (cit. on p. 25).