POLITECNICO DI TORINO

Master of Science in Automotive Engineering

Final Project

Bilby Rover autonomous mobile robot in the context of Industry 4.0

Survey on sensor-enabled applications through Webots simulations and implementation of Hector-SLAM algorithm for autonomous navigation

Supervisor:
Prof. Giovanni Belingardi
Co-Supervisor:
Prof. Maria Pia Cavatorta
McMaster University Supervisor:
Prof. Ishwar Singh

Candidate:
Alessia Valle

Academic Year 2020/2021

"O frati," dissi, "che per cento milia
perigli siete giunti a l'occidente,
a questa tanto picciola vigilia
d'i nostri sensi ch'è del rimanente
non vogliate negar l'esperïenza,
di retro al sol, del mondo sanza gente.
Considerate la vostra semenza:
fatti non foste a viver come bruti,
ma per seguir virtute e canoscenza".

D. Alighieri, Inferno, C. XXVI, vv.112-120

Abstract

In the light of Industry 4.0 revolution, the advances in information technology like Internet of Things, Artificial Intelligence, Cloud Computing, Cyber-Physical Systems and Big Data are changing the use and design of robots in the industry. Robotics, which is a multidisciplinary field that builds on the intersection of mechanical and electrical engineering, computer science, communication and networking, and material technology, is undergoing dramatic developments in recent years, as the landscape of Industry 4.0 is progressively unfolding.
This thesis stems from the collaboration with McMaster's W Booth School of Engineering Practice and Technology and aims at studying and simulating a model of an autonomous mobile robot, called Bilby Rover.

The first section of the project provides a review of literature and research related to Industry 4.0, outlining its enabling technologies, future opportunities and implications in the field of robotics. The mobile autonomous robot Bilby Rover is then presented in the second part, where the mechatronic description and the kinematic model of the system are provided. Additionally, the concept of Simultaneous Localization and Mapping (SLAM) is introduced, proposing a summary of the state of the art of the most widely used technologies. The third section is aimed at exploring through virtual simulations a set of potential industrial applications of the Bilby Rover, enabled by the integrated deployment of specific sensors: for this purpose, Webots robotic simulator is exploited. In the view of transferring this technology into the physical robot, the last section of the thesis is dedicated to a literature review concerning the Hector-SLAM algorithm for Simultaneous Localization and Mapping. Thereafter, the algorithm is tested using the Raspberry Pi single-board computer and the YDLIDAR X4 Lidar sensor, which, for this purpose, are appropriately integrated into the open-source Robot Operating System (ROS) framework.

Contents

List of Figures

List of Tables

Introduction

The content of this paper is closely related to the *Industry 4.0* paradigm, being framed within the major technological advances and breakthrough improvements that the manufacturing industry is currently undergoing across the entire value chain processes. In this regard, in Chapter 1 a literature search will be devoted to the exploration of the origins of Industry 4.0, thus focusing on its structural and organisational characteristics and outlining the so-called nine technological pillars, which can be considered as its main building blocks. Given this general background, attention is then focused on autonomous robots, highlighting the implications of Industry 4.0 on robotics, which can be defined as a multidisciplinary field that builds on the intersection of mechanical and electrical engineering, computer science, communication and networking, and materials technology.

In this context in which robotics is undergoing dramatic developments, as the landscape of Industry 4.0 is progressively unfolding, the aim of this project is to study and simulate a model of an autonomous mobile robot, called *Bilby Rover*, which has been designed and 3D printed by a team of students at McMaster University. A detailed kinematic and mechatronic description of the Bilby Rover robot will be proposed in Chapter 2, highlighting its relevant features.

As far as the content of the project is concerned, this paper can be considered having a twofold purpose, which will be addressed in two different chapters.

The first part is aimed at exploring through virtual simulations a set of potential applications, enabled by the integrated installation of specific sensors, in order to assess the suitability of the Bilby Rover to be engaged in defined tasks, typically in industrial scenarios. For this purpose, *Webots* will be employed, being a robotic simulator that offers a rapid prototyping environment for the creation of 3D virtual worlds. In this context, the work is divided into three macro-phases, namely modelling of the robot, which is time to time integrated with

application-specific sensors, programming, which is aimed at controlling the robot to obtain the desired behaviour through the deployment of dedicated algorithms, and finally simulation, to analyse the obtained results. Chapter 3 will be dedicated to the description of different scenarios that have been simulated on Webots, highlighting the steps to be followed and the technologies leveraged for the specific task.

The second part is aimed at implementing the *Hector-SLAM* algorithm for autonomous navigation. *SLAM*, which stands for *Simultaneous Localization and Mapping*, refers to the capability of autonomous mobile robots to solve the problem of constructing a map of an unknown environment while simultaneously keeping track of their position within the map. The problem is not trivial, actually it is referred to as a chicken-and-egg problem, since the robot needs to keep track of its position to build the map, but, at the same time, a detailed knowledge of the map is required to locate itself in it. Although several approaches are available to solve the problem, the purpose of this project is to test the compatibility and the performance of the Hector-SLAM algorithm, which is a software module designed by the Technical University of Darmstadt to process distance measurements and consequently produce both the map of an unknown environment and the trajectory followed by the robotic platform. To test this algorithm, a LiDAR sensor is employed as an indoor scanner and connected to the Raspberry Pi 4 Model B single board computer. Such architecture is thus able to extract range information and then send it to a remote computer, which is responsible for running the Hector-SLAM algorithm and eventually output the desired results. To provide such functionalities, however, the whole system must be integrated over *ROS*, an open-source middleware used by the community of developers and researchers to create robotic software. Therefore, Chapter 4 will be dedicated to the discussion of the mathematical foundations of Hector-SLAM algorithm and to the definition of the methodology for the execution of experiments. In this context, ROS middleware will be introduced, outlining its overall functionalities and highlighting how it is leveraged in the implementation of the Hector algorithm for simultaneous localization and mapping. Finally, the results of mapping experiments carried out in different environments will be proposed to assess the accuracy and the reliability of the constructed system, in view of its integration on the Bilby Rover robotic platform.

Chapter 1

Industry 4.0

Industrial operations worldwide are driven nowadays by global competition and the need for rapid adaptation of production to continuously changing market trends. These needs can be met only with radical breakthrough improvements in the current manufacturing technology. To address these challenges, over the last decades, a digital revolution has started in the manufacturing industry, mainly related to the adoption of new digital tools for the management of corporate and business processes. This phenomenon is leading to the continuous convergence between the real and virtual world, which can be considered the main driving force for innovation and change in all sectors of the global industrial economy. In such a framework, the progressive consolidation of *Information and Communication Technology* (ICT) and the possibility of using cutting-edge technologies to collect an ever-growing amount of data are dramatically reshaping the concept of the company. Hence, the first major breakthrough in industrial scenarios corresponds to a process of high-level digitization that, in combination with Internet-based and future-oriented technologies, leads to a new paradigm of industrial production called *Industry 4.0* [1].
The aim of this chapter is to provide an overview of Industry 4.0, outlining the origins of this strategic initiative and its structural and organisational aspects: a description of the main pillars and characteristic aspects will be proposed. Finally, a focus on the robotic implications of Industry 4.0 will be proposed.

1.1 Industrie 4.0: the fourth industrial revolution

Industry 4.0 is a term used to refer to a further developmental stage in the organization and management of the entire value chain processes that the manufacturing

industry is currently undergoing [2]. The birth of this term dates back to 2011, when a working advisory group - under an initiative promoted by the German federal government to enhance the competitiveness in the manufacturing industry of the country - presented the results of their work across various domains, including *Industrie 4.0*, which became broadly known, especially after the launching of the *Platform Industrie 4.0* in 2013 at Hannover Messe.
Taking the German *Platform Industrie 4.0* as the official reference, Industry 4.0 is defined as *the intelligent networking of machines and processes for industry with the help of information and communication technology* [3]. It is the information-intensive transformation of manufacturing and related industries in a connected environment of big data, people, processes, services, systems and IoT-enabled industrial assets with the generation and utilization of actionable data and information as a key enablers to realize smart industry and ecosystems of industrial innovation and collaboration. In a nutshell, Industry 4.0 is a broad vision with clear frameworks and reference architectures, mainly characterized by the bridging of physical industrial assets and digital technologies in so-called *Cyber-Physical Systems* (CPS) [4]. CPS are embedded systems with decentralized control and advanced connectivity that are collecting and exchanging real-time information with the goal of identifying, locating, tracking, monitoring and optimizing the production processes [5].
Industry 4.0 has the ambition to revolutionize industry management and business processes, leveraging on CPS, IoT, Cloud Computing and other advanced technologies to enable autonomous decision-making processes, real-time networking of products, processes and infrastructure, and enable connected values creation networks through vertical and horizontal integration. What makes this possible is the availability of all the relevant information in real time thanks to the networking of all the entities involved in the value creation process, together with the ability to use this data to determine the optimal value stream at any given point in time [6].

1.1.1 Through the industrial revolutions

The term Industry 4.0 is interchangeably used with the *Fourth Industrial Revolution* (4IR), to highlight the technological disruptive changes that affect the way people live, work and relate to one another. Building on foundations laid by the first three industrial revolutions, a look at the predecessors provides a perspective to understand how manufacturing has evolved since the 1800s and the innovation scale of Industry 4.0.

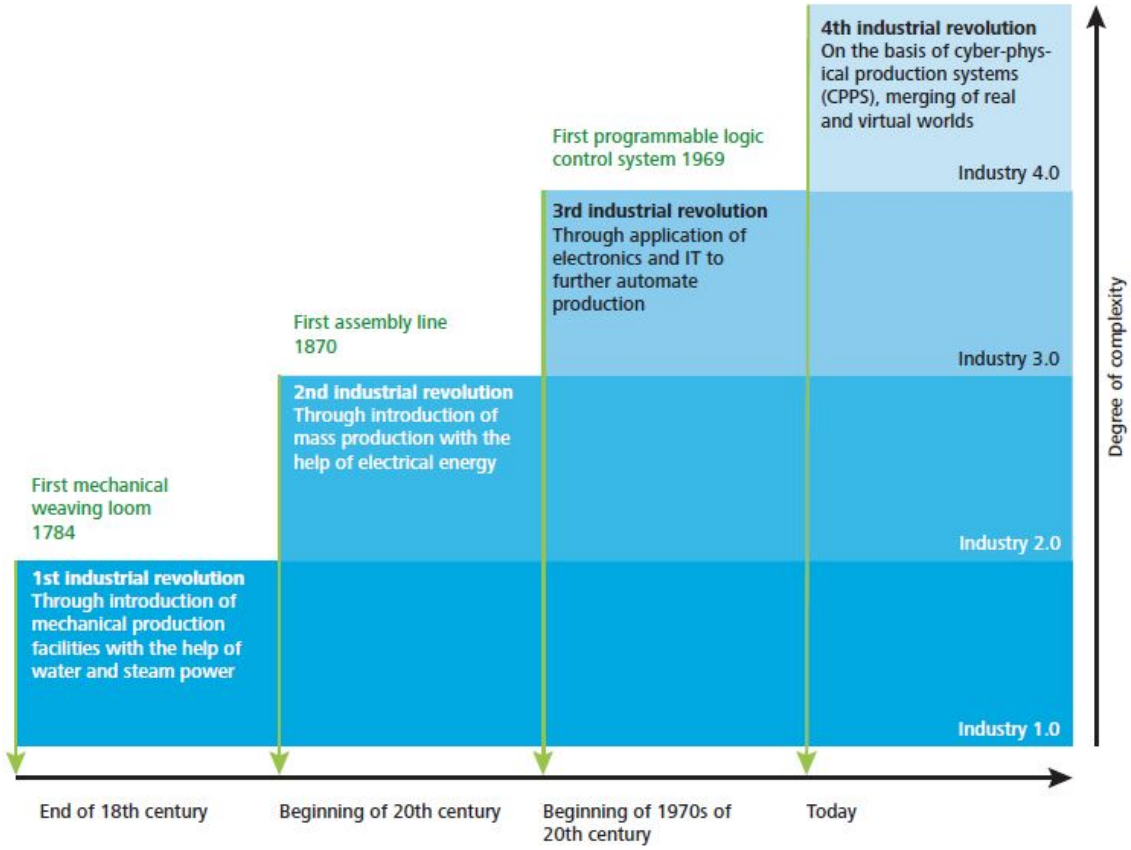

Figure 1.1: The evolution from Industry 1.0 to Industry 4.0 (Source: Deloitte [2])

The first industrial revolution spread in Britain as a result of a series of technological innovations: starting from 1760, a stream of macro-inventions, such as the steam engine and the mechanical spinning machine, spread throughout the country, allowing a shift from manual labour to mechanized production through the use of steam-powered engines and water as a source of power. The steam engine, invented by Watt in 1775, was initially used for water extraction in coalmines, thereafter its use was extended to other sectors, especially in textiles, and eventually the first steam locomotive was built in 1796. This also led to the extension and improvement of the transportation network, which resulted in the reduction of time and costs associated to the circulation of goods. The industrial revolution marked the radical transformation of the working system, with the transition to the modern factory system, which, by concentrating production in a dedicated environment, allowed the control of production processes as well as the quantity and quality of the finished product. Industrialization also entailed a massive social change through a phenomenon of urbanization whereby the population was concentrated in urban areas surrounding the factories.

The second industrial revolution arose in the second half of the 1800s, particularly after 1870, in the more advanced European countries and in the United States. It was characterized by a set of technical innovations revolving around electricity, chemistry and steel, by the spread of large-scale business and standardized production, and by the presence of large working classes. The main innovations in technology were driven by a closer relationship between science and industry. Among the sources of energy, the use of electricity in the industrial and civil fields gained a major role, with the introduction of electrical networks to deliver it. The diffusion of the American system of manufacturing also contributed to boost the rapid development: it was an approach to work management in the factory characterized by the sequential assembly of interchangeable pieces for the realization of complex products, which allowed a continuous cycle processing of single identical high quality parts. Such system is indeed the precursor of mass production and rational work organization, which emerged at the beginning of the 1900s and envisaged a standardization and fragmentation of the production process through the use of electricity-powered assembly lines, paving the way for mechanization.

The phase that began between 1950 and 1970 and that was marked by the rise of disruptive innovations in electronics, information and communications technology is called the third industrial evolution, or the Digital Revolution. As with the first and second industrial revolutions, the third one was triggered by a major technological breakthrough and by its implementation in various fields, from industry to everyday life. A series of investments were undertaken in new technological systems for defense after the Second World War, leading to the invention of the transistor and the integrated circuit, up to the digital computer, thereby contributing to the development of the third industrial revolution. A real turnaround was achieved in the 1970s with the invention of the Intel microprocessor, which allowed the application of automated systems in factories and telecommunications, generating a shift from analog and mechanical systems to digital ones. For the manufacturing industry it was an authentic qualitative leap in production and organizational standards. Actually, continuous cycle machines, specialized in the realization of a single product (used in the "Fordist Factory"), were replaced by more flexible and programmable ones, able to control volumes and characteristics of products and to elaborate, collect and transmit data. The third industrial revolution is tightly related to the birth of computers, the spread of robotics in production processes, connectivity and, obviously, the rise of the Internet, which has revolutionized the way information is handled and shared.

In the past few decades, a fourth industrial revolution has emerged on the way paved by the digital revolution. Industry 4.0 shifts the digital technology focus of the last century to a totally new stage aided by interconnectivity through the Internet of Things (IoT), access to real-time data, and the introduction of Cyber-Physical Systems. Industry 4.0 offers a more encompassing, interconnected and holistic approach to manufacturing, blurring the boundaries between physical and digital spheres. The fourth industrial revolution brings a shift from the Internet and the client-server model to a new paradigm characterized by ubiquitous mobility, bridging of digital and physical domains, convergence of IT and OT, and key-enabling technologies including IoT, Big Data, Cloud computing, with additional accelerators such as advanced robotics and AI. Connecting physical with digital, Industry 4.0 empowers business owners to better control and understand every aspect of their operation, and allows them to leverage instant data to boost productivity, improve processes, and drive growth [4]. Although many view Industry 4.0 as a continuation of the third industrial revolution, where automation was already intensive on many levels, there are three main reasons why today's transformations constitute not a straight extension of previous innovations, but rather the arrival of a distinct one: speed, scope, and system impact [7]. The pace of IT-driven change reflects the Moore's law, according to which the speed and capability of computers is expected to double every two years, as a result of the increase in the number of transistors that can be embedded on a microchip. These trends cannot be compared simply to the increased level of manufacturing automation that has been driven by digital developments since 1970: the progressive adoption of information and communication technologies by manufacturing companies is laying the foundations to revolutionize development, production, and the entire supply chain. Exponentially growing technologies will be the drivers for the transformation to Industry 4.0: 3D printing, sensors, artificial intelligence, robotics, nanotechnology are few examples of exponentially growing technologies that follow Moore's law, radically changing industrial processes, accelerating them and making them more flexible. In this transformation, sensors, machines, products, and IT systems will be connected along the value chain beyond a single enterprise. These connected systems can interact with each other using standard Internet-based protocols and analyze data to predict failures, configure themselves and adapt to changes. This, in turn, will increase manufacturing productivity, change the economy, drive industrial growth, and reshape the profile of the workforce, radically changing the competitiveness of companies and regions [8].

1.2 Building blocks of Industry 4.0

The new industrial paradigm of Industry 4.0 has the ambition to digitize the manufacturing sector, with sensors embedded in almost every product component and production equipment, ubiquitous cyber-physical systems, and analysis of all significant data. In the view of Boston Consulting Group [8], this transition is enabled by the convergence and application of nine industrial technologies, called *pillars*, which can be deemed as the main building blocks of Industry 4.0.

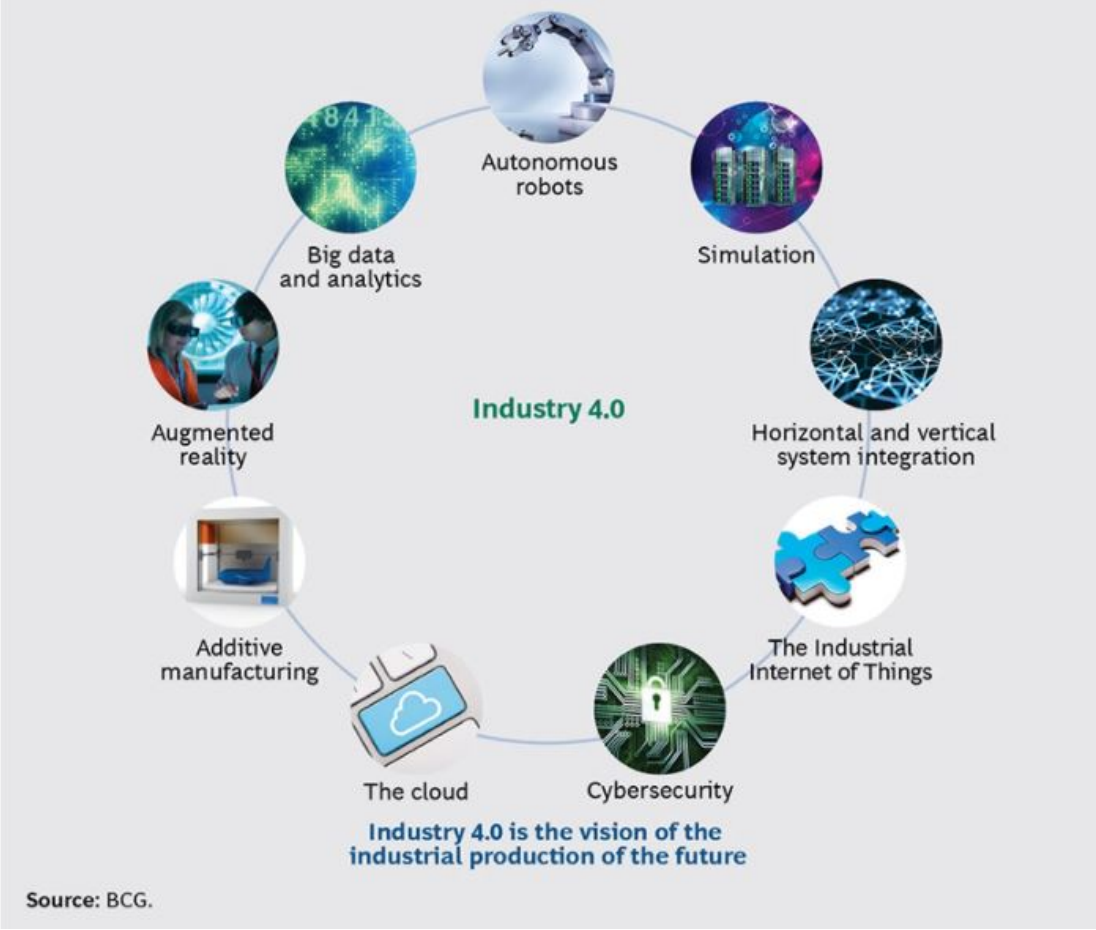

Figure 1.2: The nine technological pillars of Industry 4.0 (Source: BCG [8])

The integrated application of such technological advances - which will be outlined in the following sections - will enable the transformation of isolated cell manufacturing into a thoroughly optimized and automated production flow, leading to increased efficiency and changing the traditional relationships between suppliers, manufacturers and customers.

1.2.1 Big Data and Analytics

Big data refers to broad and diverse series of increasingly growing information, usually retrieved through data mining techniques. It encompasses the volume and value of information, the velocity at which it is created, and the variety or scale of data points that are covered (4 Vs of Big Data [10]). Analytics based on big data sets results in optimized production quality, energy savings, and improved equipment service. In an Industry 4.0 framework, the comprehensive collection and assessment of data from many distinct sources - from manufacturing equipment systems to customer management platforms - will become standard to aid real-time decision making.

1.2.2 Autonomous Robots

Since the spread of the Digital Revolution, robots have been largely employed to tackle complex tasks: at the beginning, relatively simple robots were employed to replace humans in the heaviest and most hazardous operations, and then fully automated workshop departments were set up. Nowadays, robots are evolving for even greater utility, becoming more autonomous, flexible, and cooperative. With high-end sensors and control units, autonomous robots allow for close cooperation with humans and improved interconnection with other robots in the surrounding environment, offering enhanced capability, versatility and safety.

1.2.3 Simulation

Simulations will be used more extensively in plant operations, leveraging large amounts of available real-time data to mirror the physical model in a real-time digital twin of industrial scenarios, including machines, products and people. Simulations, which can be aimed at assessing cycle times, energy consumption or even ergonomic aspects, not only lead to reduced downtime and changeovers, but also to decreased production failures during start-up. Through virtual simulation, operators can test and optimize machine settings in the virtual world before the change is actually applied to the physical industrial scenario, thereby achieving a boost in quality.

1.2.4 Horizontal and Vertical System Integration

Industry 4.0 is intended to tackle the problem of a poor connection between companies, suppliers and customers. Most of today's IT systems are not fully

integrated. Through integration, the new industrial paradigm aims to achieve an ecosystem-wide flow of information between systems and across all processes, using data transfer standards and creating the baseline for an automated supply and value chain [4]. Industry 4.0 is underpinned by three dimensions of integration:

- *Horizontal integration*, which refers to the integration of IT systems and information flows across manufacturing and business planning processes, involving both internal and external stakeholders and processes inside and outside the value and supply chain - from suppliers of materials and utilities to internal processes, distributors, and customers. Simply put, horizontal integration is about the digitization of the entire value and supply chain, made possible by data sharing and connected information system, enabling horizontal coordination, cooperation, cost savings, value generation and speed.

- *Vertical integration*, which addresses the integration of IT systems at various manufacturing and production hierarchical levels. The ultimate goal is to achieve a unique global solution that includes the field level (interfacing manufacturing processes via sensors and actuators), the control level (tuning of machines and systems), the production level (monitoring, controlling and supervising production lines), the operations level (concerning production planning and quality management), and the business planning level (dealing with order management and processing, enterprise production planning and business process management) [4].

- *Through-engineering* across the entire value chain and lifecycle of both products and customers. Engineering occurs seamlessly throughout the design, development and production of new products and services, enabling new synergies between product development and production systems. The hallmark of end-to-end engineering is that data and information is also available at all stages of a product's lifecycle, allowing new, more flexible processes [2].

1.2.5 The Industrial Internet of Things

Today, just a fraction of a manufacturer's sensors and machines are networked and employ embedded computing. With the Industrial Internet of Things (IIoT), a greater number of devices will be equipped with embedded computing and connected with standard technologies. The Internet of Things denotes the worldwide network of interconnected objects that communicate through standard protocols

[9]. IIoT is characterized by three main attributes: context - which refers to the ability for advanced object interaction within an established environment and immediate feedback if something changes - ubiquity, which stands for the ability to provide information about an object's location and physical condition, and optimization [10]. This allows field devices to both communicate and interact with each other and with more centralized controllers.

1.2.6 Cybersecurity

Industry 4.0 brings a sharp focus on security. Given that many enterprises still rely on unconnected or closed management and production systems, as industrial assets and critical infrastructure become increasingly connected, and as attacks are growing in typically rather isolated industrial environments, the stakes and threats of violations are enormous in Industry 4.0. The need to protect industrial systems and production lines from cybersecurity threats increases dramatically, requiring an end-to-end security design approach [4]. As a result, secure and reliable communications as well as sophisticated identity and access management of machines and users are paramount [8].

1.2.7 The Cloud

The straightforward networking of cloud-based platforms opens up outstanding opportunities to store and efficiently leverage the big data generated by Industry 4.0. Cloud-based IT platforms will become increasingly crucial as they serve as the technical backbone for connecting and communicating across sites and company boundaries. There are particular benefits for smart manufacturing systems operating in a decentralized network, where massive computing power will enable cloud-based applications to provide universal, anytime access to all key data. This makes it easier to collect, monitor, distribute and analyze data not only among factories, but also across the entire global value chain network, allowing for more data-driven services. This will form the foundation for providing global market solutions that smoothly integrate all steps from the value chain and suppliers to end customers and enable innovation beyond products [2].

1.2.8 Additive Manufacturing

As customer needs are constantly evolving, companies are facing the challenge of increasing product individualization on the one hand and reducing time-to-market on the other. These challenges are becoming even more demanding due to growing

levels of digitization, IT penetration and networking of products and manufacturing resources. Shrinking product life cycles in combination with the rising demand for customized products require further transformation towards organizational structures that lead to a higher degree of complexity [11]. With Industry 4.0, additive manufacturing, such as 3-D printing, will be widely exploited to produce small batches of customized products, thus offering constructive advantages, such as complex and lightweight designs [8].

1.2.9 Augmented Reality

One of the great promises and areas of interest of Industry 4.0 is the bridging of the virtual and physical worlds, where augmented reality-based systems are playing an important role. Despite being at an early stage, Virtual Reality (VR) and Augmented Reality (AR) are already being used in different fields and contexts, from consumer applications to manufacturing processes, and companies will make much wider use of augmented reality to provide workers with real-time information to improve decision making and work procedures [8]. The use of VR and AR in the manufacturing industry is accelerating as its benefits are becoming increasingly visible in various application areas. Use cases include virtual design, machining and production, training and collaboration, factory planning, assembly, safety, testing, and maintenance [4].

1.3 Industry 4.0 design principles

Industry 4.0 creates new design principles along which industry can organize itself, serving as part of the Industry 4.0 vision, and as a reference to make clearer guidelines for companies aiming to understand, identify and implement Industry 4.0 projects. These include: (a) increased interoperability between manufacturing networks through increased connectivity, (b) virtualization of manufacturing process by linking sensor data with virtual plant and simulation models, (c) decentralized decisions making, (d) real-time capability to collect and analyse data and provide insights, (e) flexible adaptation to changes by reconfiguring virtual modules, and (f) increased service orientation. Such principles allow shaping a vision of an industrial paradigm able to deliver opportunities for new business models, solution offerings, and new products [12]. As it will be explained below, each design principle delivers its own value, which is amplified when multiple design principles work together in synergy to support new and more efficient processes. Intelligence must be decentralized across various machines and devices, and all

process components must be monitored through virtualization and interoperability, allowing to respond in real-time to changes and requests. Moreover, processes and infrastructure must be modular to support customization and centered on service orientation to make services easily accessible inside and outside the factory.

1.3.1 Interoperability

One of the goals of Industry 4.0 is to integrate data science and analytical models to interpret real-time data provided by multiple sources - such as machines, processes and production systems - and combine this information together in an automated manufacturing system. To successfully implement this vision, a core requisite is to achieve interconnectivity across a diverse set of devices, integrating the massive amount of data that comes from them to support decision-making processes [13].

In the realm of manufacturing, interoperability between multiple systems, within or across industries, has been gradually accepted as a critical feature throughout a product's lifecycle [14]. The Institute of Electrical and Electronic Engineers (IEEE) defines interoperability as *the ability of two or more systems or components to exchange information and to use the information that has been exchanged* [15]: in other words, in the manufacturing industry, interoperability is defined as the electronic communication and management of data through the Internet of Things (IoT). Interoperability implies real-time access to data, paving the way for a new approach towards enhanced manufacturing operations for companies. It enables manufacturing partners (including customers, suppliers and other departments) and their machines to share information accurately and quickly, resulting in more effective and reliable operations [16]. The Industry 4.0 idea of interaction between physical and digital systems is pursued at the cyber level and at the physical machine level. The means of interaction and the mechanisms for providing functional compatibility of CPS at the physical machine level and at the cyber level are significantly different. CPS interaction at the physical machine level is provided through kinematics, sensors, and software to support the Ethernet communication environment. CPS interaction at the virtual environment level is established by an artificial intelligence embedded in the enterprise cloud storage that guides the technological processes of self-organization of cyber and physical systems [17].

The implementation of interoperability presents a challenge in establishing an efficient and reliable information management infrastructure with standard protocols of information exchange, physical machine interaction interfaces, event processing, production data representation formats, cloud operations, data management ana-

lytics, which support CPS. Although the challenge of establishing interoperability is still underway, despite the obstacles, interoperability between devices and assets is being implemented by a growing number of factories. Because of its intrinsic potential, many companies are upgrading their plants and facilities to have standardized methods of communication, data analysis, and security. Leveraging this technology as a management tool can extend beyond just operational efficiency on the shop floor to other areas such as inventory management and supply chain optimization. Manufacturing companies that have created the infrastructure to enable data interoperability are better able to serve their customers and streamline their operations [16].

1.3.2 Virtualization

As industrial automation systems and integration has become more complicated and information management systems have expanded dramatically, offering a reliable, scalable, and flexible dynamic computing environment is a key-challenge for companies operating in the present industrial scenario. With the development of virtualization technology, virtual instruments and machines are more and more widely used in the industrial area, appearing a promising path forward. In a nutshell, virtualization refers to the capability of information systems and cyber-physical systems to simulate and create virtual copies of physical world elements through the creation of digital models that are fed by merging sensor data acquired from monitoring physical processes and equipment [4]. In such a framework, the concept of digital twin gains particular relevance: it consists of a near real-time digital replica of a physical object or process that helps optimize business performance [18]. The introduction of digital twins has been possible due to significantly lower costs and improved power and capabilities, which are leading to exponential changes, enabling companies to integrate information technology and operational technology. Digital twin provides companies with a comprehensive digital footprint of their products from design to the end of the product lifecycle. A digital twin is built on a large set of cumulative, real-time measurements across multiple dimensions. They, in turn, produce an ever-changing map of the object or process in the digital world that can provide important insights into system performance, suggesting possible adjustments to be introduced in the physical world. This model, therefore, allows companies not only to understand the performance of the finished product, but also the capabilities of the entire system dedicated to the manufacturing of the product itself. The digital twin can enable companies to solve physical problems faster by

detecting them earlier, predict outcomes with a much higher level of confidence, design and build improved products, and finally serve their customers better. Thanks to the close interaction between physical and digital domains, digital twins represent significant key enablers of efficient models that provide more accurate and holistic measurements of unpredictability [18], opening new perspectives in easy reconfiguration of production lines, effective detection of failures, autonomous maintenance triggering and prompt reaction to unexpected changes in production [19].

1.3.3 Decentralization

One of the main goals of Industry 4.0 is to bring autonomy and autonomous decision-making to machines and cyber-physical systems [4]. In the new industrial paradigm, everything is connected: machines, humans, products, and IT tools communicate with each other and are arranged with the aim of improving overall production performance, inside and outside the factory. One of Industry 4.0 key features is the ability to decentralize control and decision-making as it aids changes in the manufacturing process helping to meet the growing demand for mass customization [19]. Industry 4.0 is inherently a decentralized system, with intelligence in independent entities. Actually, Cyber-Physical Systems (CPS) and Cyber-Physical Production Systems (CPPS) have their own intelligence: a smart product or CPS knows its state, location, target, and flow alternatives. Similarly, an intelligent resource or CPPS knows its state, history, maintenance plan, capacity, and its range of possible configurations. Such systems together with IoT enable the creation of an intelligent network of machines, ICT systems, products and people across the entire value chain and product lifecycle [20]. In a centralized system, a single decision center is provided with all the information of the system and is responsible for planning the entire network, thus managing the operations performed by all the nodes in the network. The central node makes decisions in terms of optimizing the goals of the entire network [19]. However, centralized system architectures, where the business logic is contained in a central information system that supports or controls the operation of various subsystems, have limitations in areas critical to Industry 4.0, in particular offering limited scalability [21]. In decentralized decision-making models, each independent network entity makes its own decisions, seeking to optimize its goals. More than one decision maker is identified and, depending on the degree of collaboration, nodes take into account the decisions of other nodes [19]. In a fully decentralized architecture, all business logic is embedded in the subsystem or component so

that it has all the intelligence it needs to perform its function, coordinating its activities with other subsystems to handle complex tasks. These decentralized systems also offer the ability to evolve over time and adapt to change, creating a more flexible environment for manufacturing [21].

1.3.4 Real-time capability

According to Industry 4.0 guidelines, a factory must be able to collect, store, and analyse real-time data, and consequently make decisions according to the new findings. For a factory to be considered *smart*, machines, devices and the entire system overall must be aware of what is currently happening, within and beyond plant boundaries [4]. Real-time capabilities – enabled by Interoperability and visualized through Virtualization - ensure that the industry has the best possible response time to internal and external stimuli by sharing, receiving, and analyzing data and information in real time [22]. This technology is supported by the extensive advancement of sensors and other devices that, embedded in all machines, material handling equipment, robots and forklifts, are capable of providing continuous streams of data and accurate algorithms [21]. By leveraging IoT and advanced analytics in the manufacturing process, smart machines embedded with specific software will automatically adapt to production needs by making independent decisions. Some companies are already using this technology to provide comprehensive financial market information, such as stock quotes, indices, and economic indicators. Real-time capabilities are not only restricted to market research, but also to internal processes, such as for detection of a machine failure in the production line: smart objects must be able to identify the defect and distribute tasks to other operating machines. This also contributes greatly to flexibility and production optimization. Constant monitoring of product quality, and of the system in general, will allow taking decisions at any time by exchanging data in real time. This interconnection will minimize resource misuse, waste, material scrap and increase energy efficiency by providing immediate results to react faster to problems or even prevent them, as in the case of proactive maintenance [22].

1.3.5 Service orientation

One of the goals of Industry 4.0 is also to transform production systems according to a customer-centric perspective, in order to meet constantly changing customer demands. Moreover, factories must not only consider external customers, but

support services that must be offered both inside and outside the organisation. In such a framework, the Internet of Services (IoS), and the service orientation that it enables, gain important relevance. The Internet of Services operates in a way that is comparable to the Internet of Things, but it creates a network of services - provided by humans and intelligent systems both inside and outside an organisation - allowing them to be delivered more efficiently and combined to increase their value. Here, the term 'services' has a broader meaning, to include all operations, from the movement of goods from one place to another to data analysis services designed to solve a particular business challenge [21]. In other words, people and smart objects need to be able to connect efficiently through the Internet of Services to create products based on customer specifications, leveraging internal and external services across the value chain to enable a seamless adaptation to change as it occurs. Service orientation enables the services of companies, cyber-physical systems and humans to be made available so that other cyber-physical systems, humans or companies can use them [21]. Everyone can have access to useful services, products, and information about the industry using virtual and digital platforms available at all times [22]. This not only increases the accessibility of these services, but also enables the creation of new types of services. The implementation of the service-oriented approach also brings several advantages for companies. In contrast to the use of traditional large applications, which cannot easily exchange information with each other, service orientation allows a more unrestricted flow of information within and among companies. This makes it easier to expose its functionality to the outside world, increasing its value and offering greater flexibility for abrupt changes [21].

1.3.6 Modularity

Another core design principle adopted by the Industry 4.0 paradigm is that of modularity, namely the ability to flexibly adapt to changing requirements by replacing or expanding individual modules [23]. In a dynamic market, the ability to respond to ever-changing trends and demands is essential for companies. To remain competitive, companies must embrace the latest technologies that prove to be beneficial in terms of efficiency and productivity. However, most companies may be facing the problem of reorganising a pre-existing rigid structure, finding themselves locked into an inflexible technology that cannot match the disruptive changes occurring in the industries [21]. Therefore, many organisations are revising their structure from a rigid and inflexible system to innovative modular platforms. Then, these modular systems can simply be readjusted in case of seasonal fluctuations

or changes in production requirements, such as the inclusion of new technologies. Thus, through software and mechatronic modularity, production can always be tailored to the environmental, systemic and changing demands of customers without errors, loss of productivity or customer dissatisfaction [22]. Modular systems are inherently scalable, making it possible, for example, to react to rapid changes while retaining the flexibility to add modules in the future. Robots, for instance, can thus evolve in a constantly changing environment, dynamically adapting to various tasks and modifying their operations as needed, without further programming or integration, to comply with new conditions [21]. In summary, modular platforms are essential as, in combination with the previous design principles, they allow to put into practice and exploit the benefits coming from the new paradigm of Industry 4.0: their synergistic integration will empower companies to leverage these cutting-edge technologies and exploit their intrinsic potential to the fullest.

1.4 Robotics: from the early stages to the IoT

In the light of Industry 4.0 revolution, the advances in information technology like Internet of Things, Artificial Intelligence, Cloud Computing, Cyber-Physical Systems and Big Data are changing the use and design of robots in the industry. Robotics, which is a multidisciplinary field that builds on the intersection of mechanical and electrical engineering, computer science, communication and networking, and material technology, is undergoing dramatic developments in recent years, as the landscape of Industry 4.0 is progressively unfolding. The sector has grown relatively steadily since the 1960s. After the first industrial robots appeared in the 1960s, a real surge of growth occurred when automotive OEMs started automating their welding departments. A new wave began to spread during the 1990s through the integration of innovative technologies borrowed from the disruptive digital revolution. Steep advances then occurred from the first decade of the 2000s, spurred on by a series of fundamental changes in the industry and economic environment: dramatic developments in technology and new applications, as well as trends of rising labour costs in conjunction with labour shortages, increased turnover, and falling equipment costs [24]. Nowadays, robotics technology is making an irreplaceable contribution to modern industry, being recognised as one of the main pillars on which Smart Factories are built. Advanced robotics systems are a key component of the highly automated, self-controlled factory of the future. New disruptive technologies and anticipated cost reductions will enable advanced

robots to perform tasks that conventional robots cannot automate economically, becoming very important productivity drivers [25] and promising to meet the dynamic demands of collaborative and smart manufacturing in the context of Industry 4.0.

The aim of this section is to shortly outline the evolution of robotics, with a brief excursus on the different technologies that have been integrated over the years, up to the present state of the art and challenges. Attention will be devoted to modern robotics as such, focusing on the key enabling technologies that are exploited in the context of Industry 4.0. Finally, a brief description of the possible applications of robotics in the industrial scenario will be provided.

1.4.1 The evolution of industrial robotics

Robotics and its applications have transformed the industrial world in different phases: indeed, it is possible to identify three main macro-phases in the timeline of the robotics evolution.
The first wave, known as Robotics 1.0, dates back to the 1960s, when the first robots were introduced, such as Unimate, an industrial robot that was installed by Unimation in 1961 on the production line at GM's Ternstedt plant in Trenton, NJ [26]. At this infant stage, robots had a bare perception of their surroundings, thus they were placed in a designated space and relied on to perform the same sequence of repetitive and labour-intensive tasks. Robots were similar to numerically controlled machines, programmed with a language similar to G-Code. The inventions of servo-motors, controllers and motor drivers were the central subjects in R&D operations [27]. However, such robots had several limitations, related to costs and, in particular, reconfigurability: being designed, built, and equipped to perform a specific sequence of operations, robots made it difficult to adapt them for a new production line.
A significant step forward was made in the era of robotics 2.0, between the 1990s and 2000s, as a result of the massive adoption of information acquisition and computing technology. At this stage, the use of automatic feedback systems gave an impulse to the development of industrial robots for increasingly diverse applications in manufacturing processes. During this period, research intensified in several fields, leading to the testing of different sensors, including torque/force sensors, tactile sensors, proximity sensors, acceleration sensors, angular sensors, thermal sensors and also 2D and 3D cameras. Other enabling technologies were Ethernet, embedded systems, real-time data acquisition and processing, and teleoperation

developments. The combined leveraging of sensors and these new technologies made it possible to extend the robots' ability to perceive their surroundings and be more aware of their state and position. Nevertheless, in many scenarios robotic systems were still lacking the capacity to interact and collaborate with other robots, objects, and workers in the factory, working mostly as isolated robotic systems [27].

Since the first decade of the 2000s, the introduction of advanced technologies has given a significant boost to robotics, especially after the spread of the Industry 4.0 paradigm, paving the way for the so-called Robotics 3.0. With further explorations and integration of cutting-edge technologies, the main goal was to address the pending needs left unsolved from the previous era, namely the need to improve connectivity, perception and data flow [27]. By integrating disruptive innovations, including IoT, real-time image recognition, deep learning, innovative machine-human-robot interfaces, system interoperability, natural language processing, digital twinning for CPS, to name a few, the purpose is to enhance the awareness and interaction capabilities of robots. Through the combination of machine-to-machine communication, secure networking and big data analytics, robotic systems are upgraded with increased efficiency, productivity and optimised performance. Robot autonomy is attained by using the new information technologies, as these enable robots to detect and monitor production processes, the work environment, human workers, and share through the network the industrial Big Data collected from sensors [28]. Therefore, advances in sensor technology and networking have been crucial to provide this capability. Increasing robot autonomy also revolves around decentralization: in this framework, a cloud-based controller serves as a computing centre, called Brain-on-Cloud, then each robot has an independent control system that allows it to make independent decisions, depending on its current state, to respond to the surrounding environment.

In simple terms, innovative technologies and solutions are being harnessed to build smarter industrial robots, enabling higher degrees of flexibility, the ability to learn tasks without redundant programming, and to collaborate independently with other autonomous devices and human operators, via the increased functionalities offered by evolving communication networks [26]. The technical features of Industry 4.0, in a synergistic combination with these equipment-specific innovations, will aid in unlocking the potential of advanced robotic systems. Hence, advanced robots will offer significant improvements, outperforming conventional robotic systems in perception, integration, adaptability and mobility [25]:

- *Perception*: considerable refinements in natural language processing, com-

puter vision and sensors are expected to offer greater autonomy and accuracy. Such increased functionality, in turn, will allow robots to perform more sophisticated tasks.

- *Integration*: new service-oriented architectures, superior connectivity, access to comprehensive data models, and progressive optimization in interface and programming will speed up the configuration process and minimize the effort required to instruct robots on tasks.
- *Adaptability*: advances in data processing technologies and access to cloud services will allow robots to learn and adapt autonomously to complex and changing environments.
- *Mobility*: through machine learning and computer vision, advanced robots will be able to autonomously drive themselves and navigate in complex environments.

1.4.2 Applications of Robotics in Industry 4.0

The intelligent capabilities derived from advanced robotics are driving new applications in factory operations. In Industry 4.0 factories, robots, equipped with superior functionalities, owing to information, sensor and networking technologies, can work hand-in-hand with human operators (collaborative robots) as well as with other robots in the surrounding environment (cooperative robots) in interconnected tasks [28]. Robotics is widely deployed in different factory departments nowadays, as it is expected to become a key productivity driver in manufacturing, logistics, quality and maintenance by 2025, shaping the factory of the future [25]. In the following, an overview of the main applications of robotics is proposed.

Manufacturing

Conventional manufacturing processes have been transformed by the adoption of intelligent robots at the shop floor level. Machines such as stand-alone robots were already in place a few decades ago to relieve human workers in demanding tasks and to achieve faster and more accurate production. However, stand-alone industrial robots were usually fixed stations, programmed for a specific application and, in addition, required the presence of safety equipment to operate without direct contact with human workers [24]. The intense surge in information and sensory technology has recently made it possible to build more efficient types of collaborative and cooperative robots. The main difference between collaborative

and autonomous robots revolves around safety: as a matter of fact, the former are embedded with an on-board safety mechanism that allows these robots to operate safely and directly alongside human workers, without the need for additional safety equipment. Sensors are therefore selected and tuned according to safety concepts, monitoring force, speed and distance to ensure collision avoidance. In a nutshell, these improvements will allow some types of advanced robots to autonomously perform processes at the same high speed as conventional industrial robots. Furthermore, the safety features of advanced robots - including the ability to work at lower speeds than conventional robots, when requested in case of danger - mean that they can share workspace with humans and assist them without the need for protective guards [25]. In addition to working cooperatively with humans, advanced robots also need to be able to work cooperatively with other robots on the shop floor, connecting together and sharing information over a network. This capability is made available by rapid advances in wireless technology: the software provides for cloud computing and big data analysis, and virtualizes the network of servers, where robots are the clients. In such an infrastructure, robots can communicate with each other, transferring streams of big data collected from their tasks to the cloud, and, eventually, to other devices. This improves the robots' learning, prediction, flexibility and decision-making over the entire production process [28].

Assembly

In the Industry 4.0 paradigm, manufacturers also use advanced robots to perform assembly tasks autonomously, in particular in the automotive industry. As robots are able to adapt their duties based on the perception of the surrounding environment, they are well suited to perform complex assembly operations, for example those involving flexible and ductile parts. In such a framework, assembly operations are aided by connectivity and intensive communication between robots and workpieces, allowing machines to understand the sequence of processes to be followed without prior specific instructions or programming. This is also made possible by simplified robot-object interfaces and the availability of existing information from which to deduce process parameters [25].

Logistics

Autonomous mobile robots can be used in a wide range of applications, including warehouses and distribution centres, manufacturing intralogistics, and in other fields such as agriculture and logistics in hospitals and retail [24]. In the indus-

trial sector, autonomous mobile robots are becoming increasingly important in warehouses and in-plant logistics for moving parts of all types and will eventually replace both stationary conveyors belts and conventional AGVs that rely on magnetic bands for guidance. Different types of autonomous mobile robots can perform both material handling and loading tasks. Computer vision, laser sensors and machine learning allow robots to self-navigate in the environment and reach the targeted destination. In addition, advanced robots are suitable for picking, packing and pelletizing operations, supported by intelligent sensors that allow them to identify, pick and handle parts, making them a useful lever to increase productivity in large-scale distribution [25].

Maintenance

Advanced robots will also have a broad application in the field of maintenance. Mobile robots can facilitate maintenance activities by transferring spare parts to their intended location and even performing mobile inspections in places where access is difficult or dangerous [25]. For example, mobile robots can support manufacturers in the visual assessment and maintenance of tanks, vessels and pipes. Another key benefit offered by intelligent robots is their self-maintenance capability. Prognostics and Health Management (PHM) have been introduced to allow robots to monitor the status of a system, detect possible errors and even predict the occurrence of machine failures. By processing information streams from Big Data, industrial robots are able to monitor their health, and by leveraging IoT, they can efficiently use PHM, providing rapid decision making and consequently ensuring greater reliability of the entire system [28].

Quality

Advanced intelligent robots are also employed in quality control operations. Industrial robots, thus endowed with technical innovations, can monitor and control all physical processes in real time, and automatically adjust equipment parameters - to which they are digitally interconnected - in response to perceived quality. In addition, vision systems enable the detection of defects and recognition of damages on various parts, thus ensuring optimised quality of final products [29].

Chapter 2

Introduction to Bilby Rover

The present work is framed within the context of Industry 4.0, with a major focus on the applications in the field of Robotics. The project stems from the collaboration with McMaster's W Booth School of Engineering Practice and Technology and aims at studying and simulating a model of an autonomous mobile robot, called Bilby Rover, which has been designed and 3D-printed by a team of students at the McMaster University.

Figure 2.1: Bilby Rover Autonomous Mobile Robot

The proposed robot, equipped with a LiDAR, is expected to be able to move autonomously in an unknown environment through the implementation of the

Simultaneous Localization and Mapping (SLAM) technology. It will also be able to identify obstacles in its path, and modify its behaviour to respond to the surrounding areas. This result, as will be explained in detail below, will be enabled by the integration of different technologies, in full compliance with the previously introduced Industry 4.0 guidelines. Moreover, although the implementation of SLAM represents a central topic of this project, other applications may be proposed, owing to the versatile structure of the robot, which makes it suitable to be equipped with different types of sensors, thus serving multiple purposes, as it will be seen in Chapter 3.

The chapter is structured as follows. A general mechatronic description is outlined in the first section, presenting the mechanical, electronic and sensor systems. In this regard, an overview of the Bilby Rover chassis is provided and the kinematic model of the system is briefly presented. In the second part, the aim is to explore the concept of Simultaneous Localization and Mapping (SLAM), proposing a summary of the state of the art of the most widely used technologies. In particular, the focus will be placed on the two major applications, namely Visual SLAM and LiDAR SLAM, briefly describing the main deployments and sensors used.

2.1 Bilby Rover description

This section outlines the hardware components and the design considerations to develop the proposed robot, covering the systems that are actually present on the Bilby Rover, according to its original design presented in the drawing by McMaster University. A 3D-printed four-wheel drive (4WD) chassis is used as the base on which the following hardware components are fit:

- Raspberry Pi 4 Model B 4 GB for CPU and GPU computations
- YDLIDAR X4 laser rangefinder
- 2 DC motors
- Motor drivers
- Powerbank to supply power to Raspberry Pi and sensors
- Batteries to supply power to motors
- Jumper wires to connect individual components

2.1.1 Robotic body

Body frame

The body of the Bilby Rover robot has been purposely designed and 3D printed in the laboratory of W Booth School of Engineering Practice and Technology. CAD/CAE tools have been employed to design the prototype, taking into account several requirements, such as the environment conditions, the appropriate position of sensors on the platform, and the allocation of the control system. The overall dimensions of the robot body are reported in the figure below.

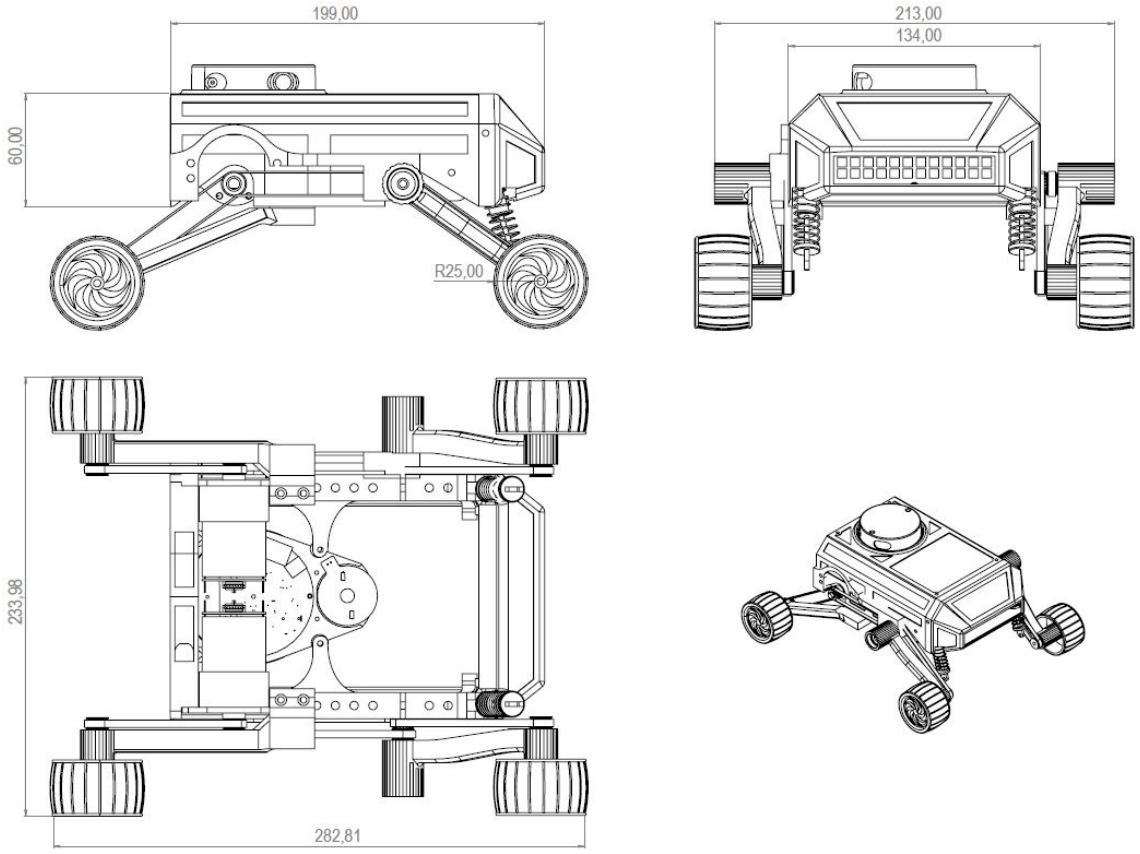

Figure 2.2: Robot overall dimensions

Wheels configuration and robot kinematics

As it can be observed in the Figure 2.2, the Bilby Rover robot is designed having four conventional fixed wheels, with both centre and axis fixed to the frame.

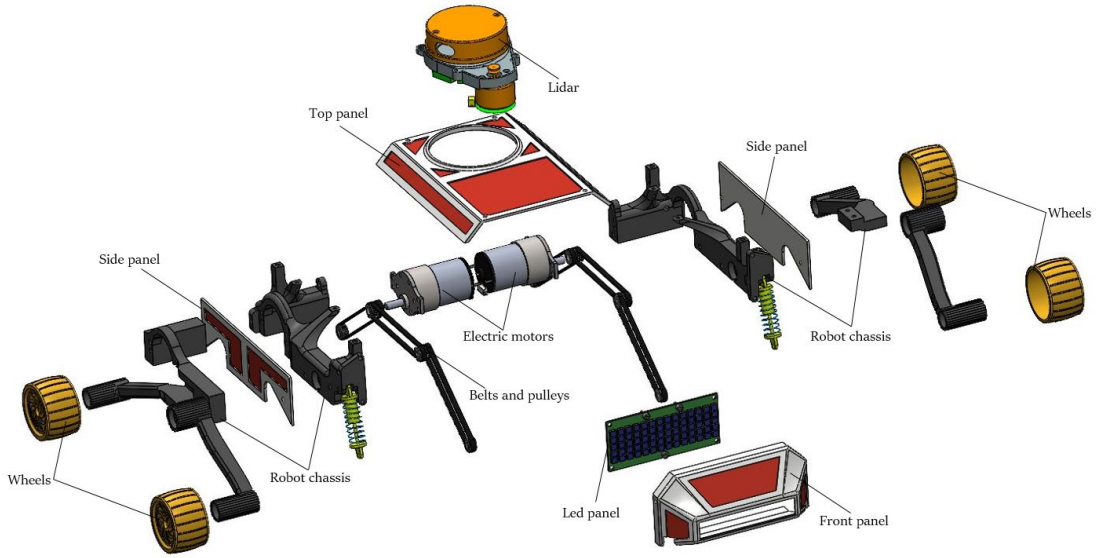

Figure 2.3: Exploded view of the Bilby Rover

All four wheels of the robot are active: this is made possible by a system of belts and pulleys. Actually, in the exploded view of the robot depicted in Figure 2.3, it is possible to observe two rotational motors, one per side, connected to pulleys. These, in turn, drive a system of belts, which allow the rotary motion to be transmitted from the motor to the wheels on the same side of the robot. Consequently, the wheels located on the same side of the robot will run at the same speed as they are driven by the same motor.

Before discussing the wheels configuration in further detail, it is worth introducing the concept of *holonomy*. In robotics, holonomy refers to the relationship between controllable and total degrees of freedom of the specified robot. A system is said to be holonomic if the number of controllable degrees of freedom equals the total number of degrees of freedom [30]. A robot moving in a plane (i.e. on smooth ground) has 3 degrees of freedom: given a global reference system, the x and y coordinates define the robot's position, while the θ coordinate sets its orientation with respect to the aforementioned reference frame.

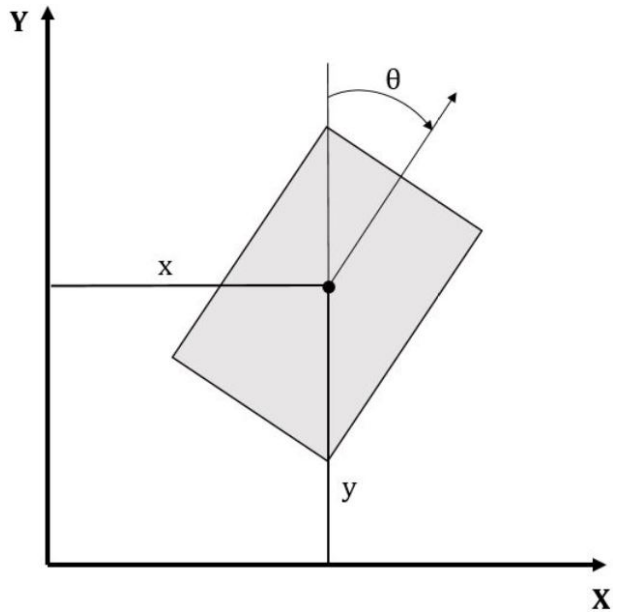

Figure 2.4: Schematic of the reference frame

In this framework, a robot is referred to as holonomic if it is able to change its position or orientation independently, as in the case of mobile platforms equipped with omnidirectional wheels (see Figure 2.5). Actually, omnidirectional wheels contain rollers along their circumference that are rotated with respect to both the plane of the wheel and the axis of rotation. The combined action of these wheels allows the platform to move in any direction.

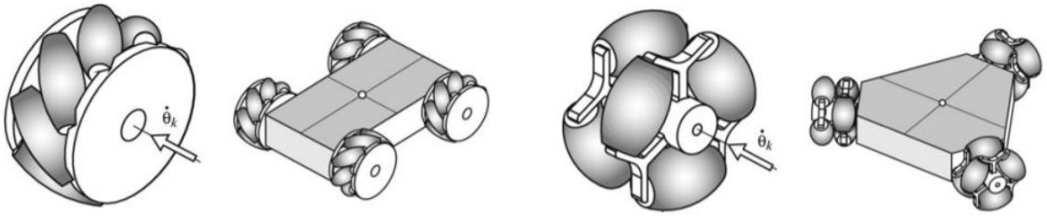

Figure 2.5: Omnidirectional wheels with peripheral rollers (Source:[30])

In analytical terms, a system is said holonomic if all the constraints it is subjected to are expressible as a function of the form:

$$f(x_1, x_2, ..., f_N, t) = 0 \tag{2.1}$$

where x_i are the system coordinates. Thus, the constraints depend only on time and coordinates x_i [31]. A system having constraints that cannot be ex-

pressed in the previous form is said to be *non-holonomic*. For instance, a car is a non-holonomic system, as it can go straight and steer, showing two controllable degrees of freedom: direct sideways movement is not allowed, on the contrary such motion can be performed through a combination of different manoeuvres, as it happens for parallel parking. In robotics, non-holonomic robots represent a very common solution owing to their simple design and ease of control. Indeed, the Bilby Rover is an example of a non-holonomic robot. Relying on a chassis with four conventional, non-steering wheels, the robot has only one controllable degree of freedom (position along the x-axis), since the fixed wheels do not allow independent steering manoeuvres, as with vehicles. Accordingly, the robot will not be restricted in moving forwards and backwards, while sideways and rotational movement will be executed as results of combined and more complex manoeuvres.

For this purpose, the Bilby Rover can be considered as a *Skid-Steering Mobile Robot* (SSMR), namely a category of mobile platforms for which the slippage mechanism is fundamental because of the drive mechanism. The steering of an SSMR is achieved by differentially driving wheel pairs on each side of the robot. Although the steering scheme yields some mechanical benefits, such as the robustness of the mechanical structure, controlling an SSMR is a challenging task because the wheels have to slip sideways to follow a curved trajectory [32], making path planning particularly difficult especially in narrow environments.
To enable a better understanding of the robot, it is suitable to formulate the kinematic model of the Bilby Rover. In this regard, the formulation of the model is carried out by making reference to [32]. For the sake of simplicity, the robot is assumed to be placed on plane surface, where a global reference frame (X_g, Y_g, Z_g) is fixed. A local reference frame denoted by (x_l, y_l, z_l) is set jointly to the robot center of mass (COM). On the base of this arrangement, the COM can be expressed with respect to the inertial frame as (X, Y, Z); however, assuming that the robot is engaged in a planar motion, the Z-coordinate of the center of mass is considered constant. In such configuration, the linear velocity expressed in the local frame is defined by the vector $\mathbf{v}=[v_x\ v_y\ 0]^T$, while the angular velocity is defined by $\boldsymbol{\omega}=[0\ 0\ \omega]^T$. Instead, the state vectors $\mathbf{q}=[\mathrm{X}\ \mathrm{Y}\ \Theta]^T$ and $\dot{\boldsymbol{q}}=[\dot{X}\ \dot{Y}\ \dot{\Theta}]^T$ describe respectively the generalized coordinates and velocities of the robot with respect to the global reference frame.

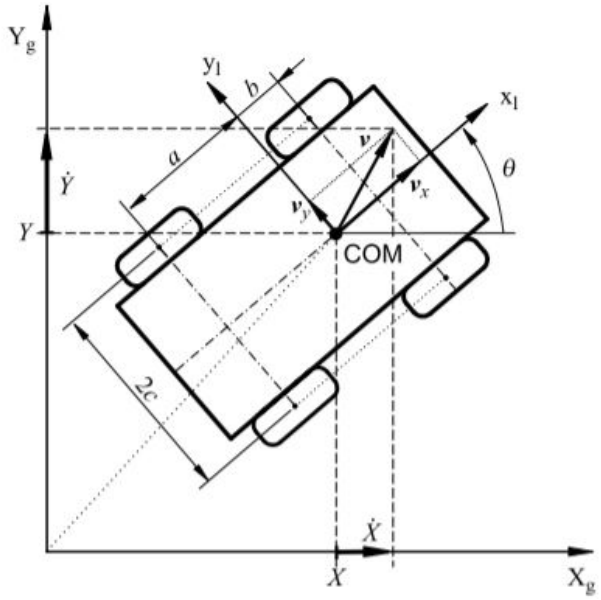

Figure 2.6: Robot velocity diagram (Source:[30])

Then, is possible to prove that the local velocities are related to the global ones according to Equation (2.2):

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} \qquad \dot{\theta} = \omega \tag{2.2}$$

At this point, it is necessary to define the relationships between the wheels velocities and the local velocities. In this regard, it is assumed that each i-th wheel moves at a rotational speed ω_i and, for simplicity, the thickness of the wheel is considered negligible to ensure a point contact with the plane. As a simplifying hypothesis, the grip between the wheels and the ground is enough to ensure the absence of slipping in the longitudinal direction (pure rolling). According to these assumptions, the longitudinal component of the velocity vector v_i of the i-th wheel in the local frame results from the formula in Equation (2.3):

$$v_{ix} = r_i \omega_i \tag{2.3}$$

where r_i is the effective rolling radius of the wheel.
As it is possible to see from the velocity diagram reported in Figure 2.7, the lateral component of the velocity v_{iy} is usually nonzero. This is due to the mechanical configuration typical of Skid-Steering Mobile Robots, for which lateral skidding

is a necessary condition to change the orientation of the robot. Therefore, the lateral component of the i-th wheel velocity is null only if ω referred to the COM is zero, namely when the robot moves along a straight line [32].

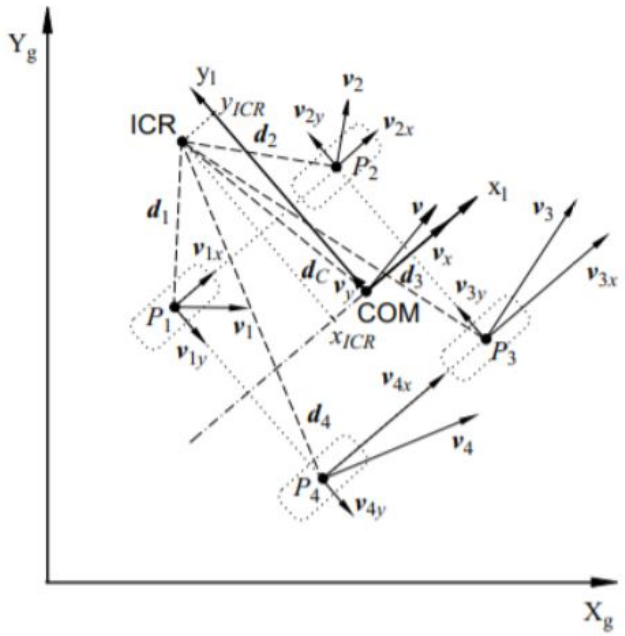

Figure 2.7: Wheels velocity diagram (Source:[30])

Considering a situation in which the robot is turning, it is possible to identify an Instantaneous Center of Rotation (ICR) that can be shortly defined as the virtual point on the plane around which the rigid body is rotating in a specific moment. In such a configuration, the radius $\boldsymbol{d_i}=[d_{ix}\ d_{iy}]^T$ and $\boldsymbol{d_C}=[d_{Cx}\ d_{Cy}]^T$ can be defined with respect to the local reference frame, running respectively from the i-th wheel and the COM to the ICR. On the base of these, the following relations can be deduced:

$$\frac{v_{ix}}{-d_{iy}} = \frac{v_x}{-d_{Cy}} = \frac{v_{iy}}{d_{ix}} = \frac{v_y}{d_{Cx}} = \omega \tag{2.4}$$

From the definition of ICR coordinates $(x_{ICR},y_{ICR}) = (-d_{xC},-d_{yC})$ in the local frame, Equation (2.4) can be rewritten as:

$$\frac{v_x}{y_{ICR}} = -\frac{v_y}{x_{ICR}} = \omega \tag{2.5}$$

Considering the geometric configuration and the following relations:

$$\begin{aligned} d_{1y} &= d_{2y} = d_{Cy} + c \\ d_{3y} &= d_{4y} = d_{Cy} - c \\ d_{1x} &= d_{4x} = d_{Cx} - a \\ d_{2y} &= d_{3y} = d_{Cx} + b \end{aligned} \tag{2.6}$$

it is possible to obtain the correlations between wheel velocities as expressed in Equation (2.7):

$$\begin{aligned} v_L &= v_{1x} = v_{2x} \\ v_R &= v_{3x} = v_{4x} \\ v_F &= v_{2y} = v_{3y} \\ v_B &= v_{1y} = v_{4y} \end{aligned} \tag{2.7}$$

Where v_L and v_R represent the longitudinal component respectively of left and right wheel velocities, while v_F and v_B are the lateral component of front and rear wheels.
Combining Equations (2.4)-(2.7), it is possible to define the transformations that link together wheel velocities with the robot speed:

$$\begin{bmatrix} v_L \\ v_R \\ v_F \\ v_B \end{bmatrix} = \begin{bmatrix} 1 & -c \\ 1 & c \\ 0 & -x_{ICR} + b \\ 0 & -x_{ICR} - a \end{bmatrix} \begin{bmatrix} v_x \\ \omega \end{bmatrix} \tag{2.8}$$

Assuming the radius of each wheel to be r, it is possible to express in matrix form the rotational speed of both left and right wheels as follows:

$$\omega_{wheels} = \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} = \frac{1}{r} \begin{bmatrix} v_L \\ v_R \end{bmatrix} \tag{2.9}$$

Finally, the relations between the robot speed and the wheels velocity can be therefore developed from the inverse formula:

$$\begin{bmatrix} v_x \\ \omega \end{bmatrix} = r \begin{bmatrix} \frac{\omega_L + \omega_R}{2} \\ \frac{-\omega_L + \omega_R}{2c} \end{bmatrix} \tag{2.10}$$

From Equation (2.10), it is possible to understand that the pair of velocities ω_L and ω_R can be treated as control kinematic input as well as v_x and ω. Actually, since the wheels are directly actuated by the motor, by setting the desired linear

speed v_x and ω as control inputs of the robot platform, the values of ω_L and ω_R can be retrieved from the inverse formula of the Equation (2.10) and sent as commands for the motors connected to the wheels.

2.1.2 Sensing unit

Autonomous control and operation of the mobile robot relies heavily on external sensor information. Actually, sensors are devices capable of converting a physical event or quantity into an electrical signal, allowing measurements to be made [33]. Therefore, robotic sensors provide robots with the ability to detect and sense, thereby acting in a way similar to sensory organs for living organisms. Leveraging sensors, the robot becomes aware of its surroundings.
Addressing the specific application of the project, the Bilby Rover is equipped with a *YDLIDAR X4* sensor, installed in a dedicated housing on the top panel of the robot body, as it possible to see in the exploded view in Figure 2.3.

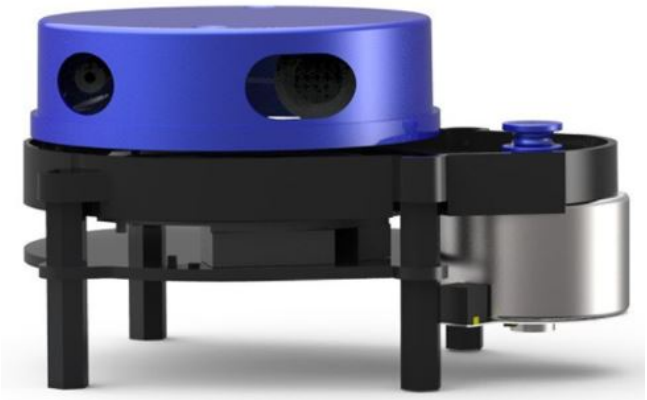

Figure 2.8: YDLIDAR X4 (Source: www.ydlidar.com)

Before discussing the technical specifications of YDLIDAR X4, it is worth outlining briefly the operating principle of LiDAR sensors.

LiDAR technology

LiDAR, acronym for *Light Detection And Ranging*, is a remote sensing technology that uses light in the form of pulsed laser beams to measure distances to a selected target. LiDAR uses electromagnetic waves in optical and infrared wavelengths and is an active sensor, implying that it sends out an EM wave and receives back the reflected signal [34]. On the one hand, its operating mechanism is similar to that of radar and sonar sensors, apart from the fact that it works at much shorter lengths than radio and sound waves. On the other, its working wavelength is

similar to that of passive electro-optical sensors, except that it provides its own radiation rather than using already existing ones.
LiDAR sensors feature three main components: the transmitter, the receiver and the detector. Figure 2.9 provides a basic schematic of the LiDAR architecture.

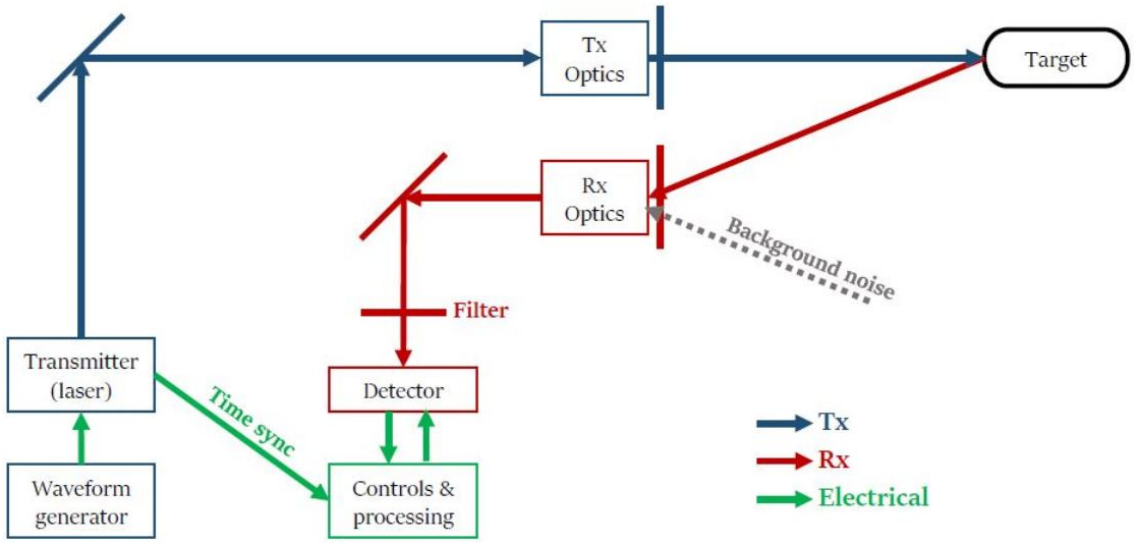

Figure 2.9: LiDAR conceptual diagram (Source: adapted from [34])

A waveform generator produces the laser beam needed to retrieve the range information, usually harnessing a diode laser or a diode-pumped solid-state laser [34]. The LiDAR needs a transmitting optical aperture to emit the light towards the target and a receiving optical aperture to properly capture the reflected EM wave. In this regard, there are two possible configurations available: monostatic LiDAR, in which a single aperture is used for both the transmitter and the receiver, and bistatic LiDAR, in which the transmitting and the receiving part have dedicated openings. The LiDAR operating principle is relatively straightforward: the sensor emits laser beams towards a selected target and measures the time it takes for the EM wave to bounce off the target and return to its source (*Time of Flight*, ToF). The time between the emitted and the reflected pulse allows calculating with accuracy the distance to each object [35]: since electromagnetic waves travel at the speed of light (approximately 299 792 458 m/s), the distance to the target can be measured as follows:

$$d = \frac{c \cdot ToF}{2} \tag{2.11}$$

where d is the distance to the target, c is the speed of light, and ToF is the Time of Flight, namely the time elapsed between the emission of the laser beam and the reception of the reflected beam at the receiver.

LiDAR sensors are often used as mapping and imaging devices. By capturing millions of precise distance-measuring points every second, called *point clouds*, LiDARs can provide a complete digital map of the surrounding environment, from which information on the position, shape and behaviour of objects can be retrieved. Such a property makes LiDAR a compelling solution for a broad range of applications and across various domains, such as archaeology, agriculture, seismology, automation, smart retail, as well as other military and civil applications.

YDLIDAR X4

YDLIDAR X4 is a two-dimensional 360-degree rangefinder developed by Shenzhen EAI Technology Co. Ltd and equipped with the necessary optics, electricity and design algorithm to achieve high-frequency and high-precision distance measurement.
This particular LiDAR is a bistatic sensor (meaning that transmitter and receiver have a dedicated aperture) using a point-pulsed infrared laser which, in combination with a specific optical lens, is deployed for the transmission and reception of the signal to achieve high-frequency ranging during operation. Thanks to a 360-degree rotating mechanical structure, driven by an electric motor connected to it, the sensor continuously produces angular information and point cloud data of the scanning environment during the measurement. The optical parameters are available on the manufacturer website and are summarised below:

Table 2.1: YDLIDAR X4 Optical parameters

Parameter	**Min**	**Mean**	**Max**	**Unit**
Wavelength	775	785	795	nm
Laser Power	-	3	5	mW

The technical specifications of YDLIDAR X4 are enlisted below:

- 360-degree scanning field of view for distance measurement
- 10 m ranging distance
- Ranging frequency up to 5 KHz
- Small distance error, stable performance and high accuracy
- Strong resistance to ambient light interference
- Motor frequency ranging between 6 Hz-12 Hz

- Low power consumption

The overall dimensions of the LiDAR device are reported in Figure 2.10.

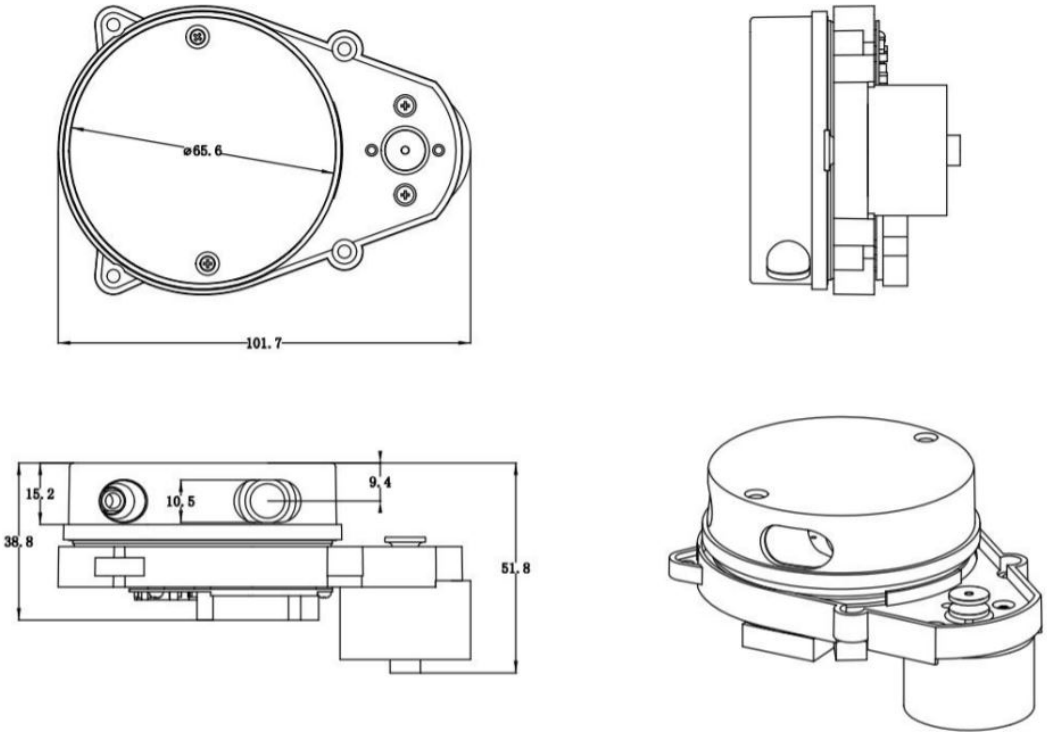

Figure 2.10: YDLIDAR X4 dimensions (Source: www.ydlidar.com)

2.1.3 Processing Unit

Coordination and control of the robot's operations is handled by a Raspberry Pi 4 Model B board, which will be installed on the Bilby Rover chassis.
The Raspberry Pi 4 Model B is a credit-card sized, fully programmable single-board computer (SBC), and is the latest product in the popular Raspberry Pi computer range, offering considerable enhancements in processor speed, multimedia performance, memory, and connectivity over its predecessors. Key features of this product are documented in the reference manual [36] and include a high-performance 64-bit quad-core processor, dual-display support at resolutions up to 4K via a dual micro-HDMI port, hardware video decoding up to 4Kp60, 4GB of RAM, dual-band 2.4/5.0 GHz wireless LAN, Bluetooth 5.0, Gigabit Ethernet, USB 3.0 and PoE capability [36].

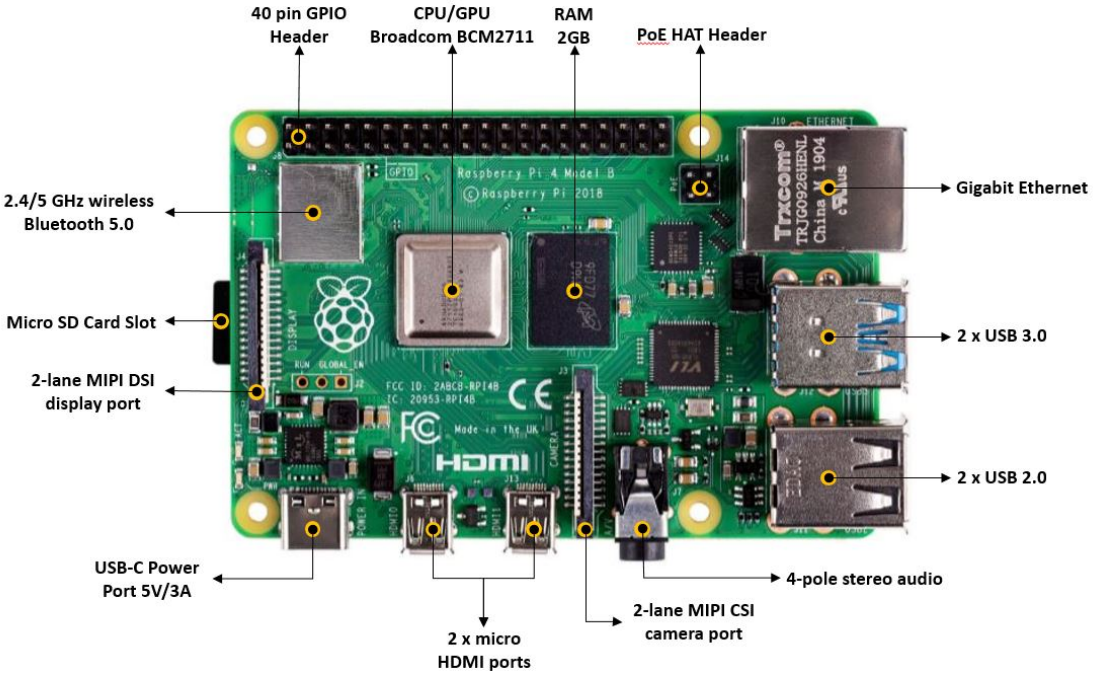

Figure 2.11: Raspberry Pi Model B (Source: www.raspberrypi.org)

Raspberry Pi comes with a fully functional operating system called Raspberry Pi OS, a Debian-based distribution developed specifically for the single-board computer and loaded onto an SD card flashed in a dedicated slot in the computer board. Based on an ARM processor, which can run the Linux-based operating system, Raspberry Pi can be adequately used as a desktop when connected with peripherals such as a screen, keyboard and mouse. Nonetheless, what sets Raspberry Pi computers apart from other desktops is the presence of a GPIO connector, namely 40 general-purpose input/output pins that can be controlled by software [37]. Such feature allows the Raspberry Pi to be interfaced with electronic circuits; therefore, units such as buttons, LEDs and motors can be controlled using the Raspberry Pi board as in a microcontroller. In addition, thanks to internet connectivity, Raspberry Pi offers a viable solution for remotely monitoring and controlling the robot on which it is mounted. In this way, the device fulfils the purpose of connecting the robotic platform to a remote control station (i.e. a computer) via Wi-Fi network, thus enabling the following functionalities:

- Receive data and commands from the control station
- Convey instructions to the microcontrollers
- Retrieve information from devices and sensors connected to the single-board computer.

2.1.4 Overall system architecture

Although the assembly of the physical robot is outside the scope of the project and will be left as a future development, it is worth providing an overview of the overall system architecture to build a clear picture of all the components, mostly to support the robot assembly task that will be carried out by future students. Actually, in addition to the components described in the previous sections, it will be necessary to integrate the robot with specific devices that will enable the locomotive capabilities of the platform, namely motors and motor drivers.

As general guidelines, the idea is to employ two DC motors, one per side, which must be appropriately connected to the system of belts and pulleys to enable the transmission of rotational movement to the four wheels of the Bilby Rover across the entire kinematic chain. While the transmission of movement downstream of the motors is ensured by the correct coupling of the various components up to the wheels, the drive control of the motors themselves is carried out via specific commands that are managed by the Raspberry Pi SBC, which to all effects serves the function of a microcontroller. In this connection, a motor driver must be employed, acting as an interface between the motors and the control circuits. Since motors typically require a high amount of current, while control circuits work on low current signals, the motor driver comes in handy, being an integrated circuit chip that is designed to take the low current control signal and then transform it into a higher current signal that can drive motors. In simple terms, despite the different possible topologies of the device, a motor driver can control the direction of the motor according to the commands and instructions it receives from the controller, thus allowing the conversion of a signal into the effective movement of the actuators.

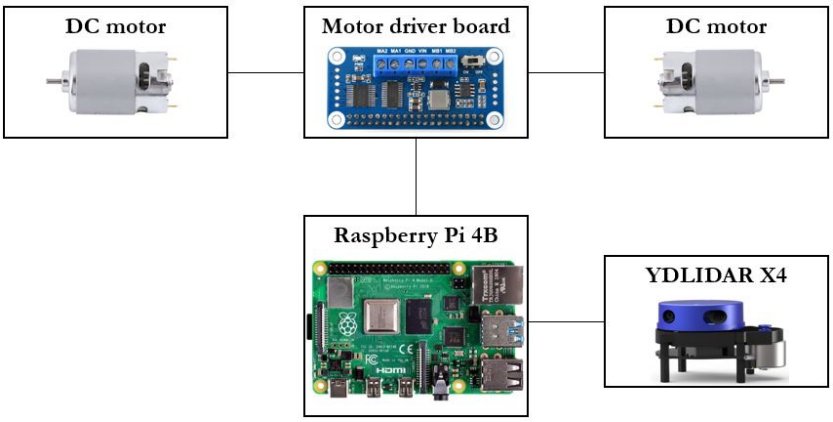

Figure 2.12: High-level system architecture

In view of these considerations, Figure 2.12 presents an overview of the system architecture, including the connections between the main devices deployed on the robotic platform of the Bilby Rover.

2.2 SLAM

An autonomous mobile robot needs to tackle two critical issues to navigate its surroundings: mapping the environment and tracking its relative position within the map. However, real-world applications in GPS-denied areas, such as manufacturing sites, require robust mapping and perception techniques to enable mobile systems to navigate autonomously in complex environments [38]. The close interdependence between localization and mapping operations dramatically increases the complexity of the problem, thus requiring both tasks to be addressed at the same time. Actually, on the one hand, in order to move accurately, a mobile robot must have an accurate map of the environment; on the other hand, in order to build an accurate map, the tracking positions of mobile robots must be known precisely [39]. In this context, the research community has pointed out that *Simultaneous Localization and Mapping* (SLAM) is a promising technology to solve this problem. SLAM is a method that involves the simultaneous estimation of the state of the robot, equipped with on-board sensors, and the construction of a model of the environment (namely a map) that the sensors are perceiving [40]. Starting from an undefined position in an unknown environment, the robot locates its position through repeated observations of its surroundings during motion, and simultaneously builds an incremental map of the environment based on its position, thereby achieving the goal of simultaneous positioning and map construction [39]. Therefore, the term SLAM, which is used for both the problem itself and the algorithmic solutions that attempt to solve it, refers to the chicken-egg problem of building a map - which requires localization - while using this map for localization [41].

The aim of this section is to provide an overview of the literature on Simultaneous Localization and Mapping, with a brief reference to the analytical formulation of the problem. In the following, the two main SLAM techniques, Visual SLAM and LiDAR SLAM, will be presented, focusing on their characteristics and structural features.

2.2.1 Problem formulation

The purpose is to provide a summary of the SLAM problem statement, referring to the main variables at stake. However, the discussion will not go into much detail because, as it will be seen in Chapter 4, the SLAM algorithm will be implemented automatically by a dedicated middleware software (ROS), which provides built-in libraries to tackle the problem using data coming from sensors. Therefore, only general guidelines for understanding the subject matter are hereby presented.

In essence, Simultaneous Localization and Mapping refers to the probabilistic method used to construct a 2D or 3D map of an unknown environment under imperfect localization. Although, in general, the robot at any given time t is defined by 6 degrees of freedom, from the perspective of a considerable number of applications, a substantial part of SLAM research focuses only on 2D space, that is with 3-DoF robots defined by (x_0, x_1, θ), as seen in Figure 2.4. One or multiple sensors installed on the robot platform provide at every time t sensor readings z_t, which are processed into a map m_t. There are two different ways in which environment observations can be leveraged for SLAM implementation:

- External sensors can detect and locate landmarks, namely distinguishable features that can be identified in fixed positions: two consecutive observations of the same landmark at times t and $t + 1$ allow for the assessment of the relative movement δx_t of the robot, that is, the relative change in the robot's position, which can be denoted as $x_{t+1} = x_t + \delta x_t$ (see Figure 2.13)
- Registration can be employed, meaning a method that creates spatial constraints by spatially aligning sensor data based on their readings. For instance, a LiDAR scans a part of the environment to generate a profile of distances to the nearest obstacle. Two successive scans can be registered by finding the spatial transformation - and hence the relative movement δx_t of the robot - that leads to the best fit of the two scans (see Figure 2.14).

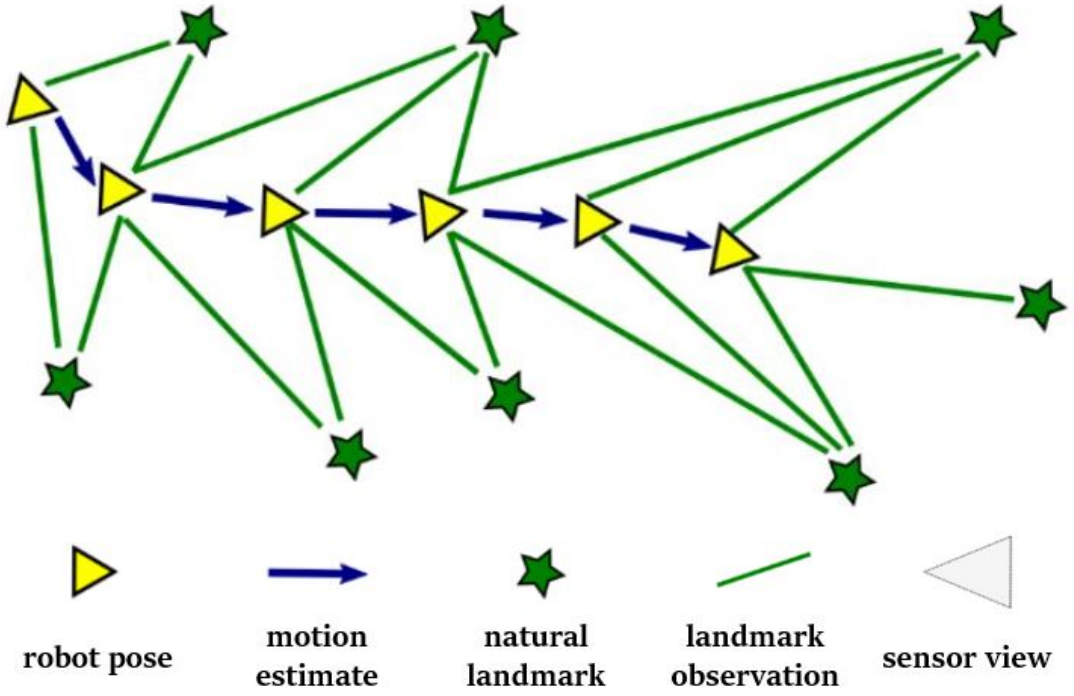

Figure 2.13: Landmarks recognition

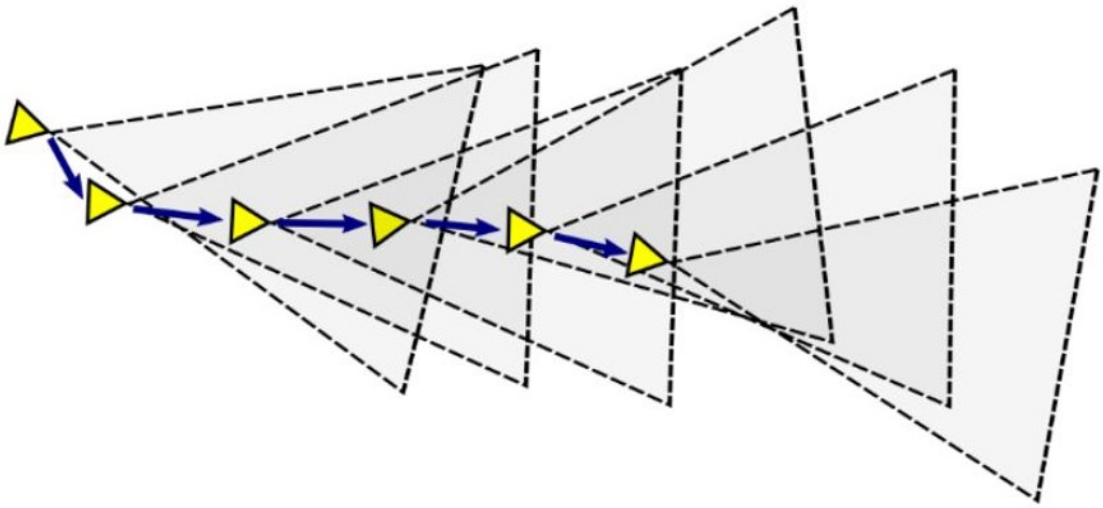

Figure 2.14: Constraints generated through registration

In addition to sensing the environment, the robot can use its internal sensors and control inputs in combination with appropriate models to estimate its poses. This information is denoted by u_t, just like the standard notation for the control inputs in a system. For example, motor control inputs for a robot with a two-wheel

differential drive can be used in combination with a forward kinematic model to estimate its motion [41].

A critical factor in the SLAM problem is the uncertainty associated with the motion estimation, and consequently related to the prediction of robot poses. It is therefore possible to assume a probability distribution $P(x_t)$ for each time step t, where x_t defines the robot position at a given time. The movement of the robot in time leads to a sequence of poses or trajectory that is denoted by $x_{0:t}$. Likewise, the related sensor observations, control inputs and maps will be denoted accordingly as $z_{0:t}$, $u_{0:t}$, and $m_{0:t}$. Each motion estimate introduces an extra uncertainty and the cumulative error grows indefinitely. Purely sequential motion estimates would lead to an unlimited drift in pose estimates since they add up errors in the local coordinate systems of the robot, as it can be seen in the following figure that provides an illustration of the covariance in x_0 and x_1 coordinates (θ coordinate is omitted for the sake of simplicity).

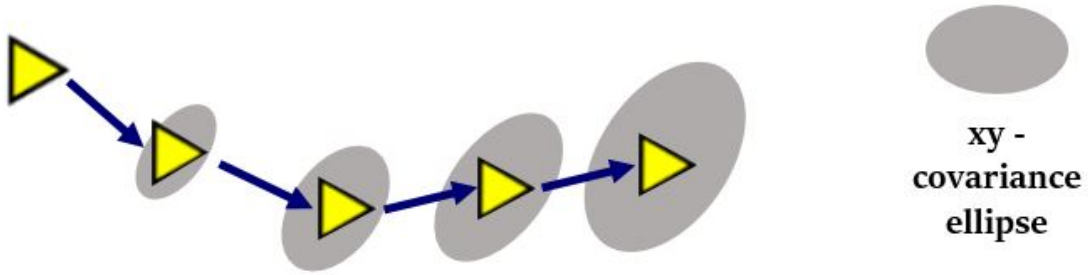

Figure 2.15: SLAM uncertainty: drift in pose estimate

This drift in estimation errors can be bounded through the exploitation of loop closures, which consist in generating spatial constraints between a pose estimate x_n and a previously visited location x_{n-k} $(0 < k < n)$. Based on non-sequential observations of the same landmark or non-sequential registration of sensor data, loop closures create anchor points in the environment coordinate frame that help to limit the incremental error in pose estimation. SLAM implementations usually try to exploit different types of loop closures. Actually, loop closures can be set between pose estimates that are close in time (for instance, between x_n and x_{n-2}), which is also referred to as tracking; more importantly, loop closures can be established when the robot returns to a previously visited location after travelling for some time. All in all, loop closures are crucial in SLAM as they represent a

key element in bounding uncertainty and cumulative error in sequential motion estimates [41].

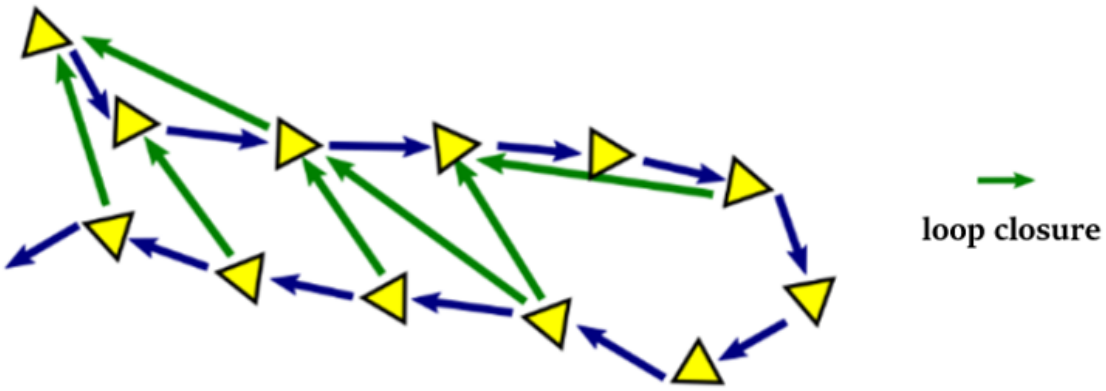

Figure 2.16: SLAM multiple loop closures

On the base of these considerations, it is possible to formulate the SLAM problem from a more formal point of view. Loosely speaking, given a set of sensor observations $z_{0:t}$ and control inputs $u_{0:t}$, SLAM is aimed at finding the most probable trajectory (as a discrete set of robot poses) and environment map as:

$$x^*_{0:t}, m^* = \arg\max_{x_{0:t},m}[p(x_{0:t}, m|z_{0:t}, u_{0:t})] \tag{2.12}$$

Equation (2.12) expresses the probability distribution of the actual robot pose and map conditioned by the coordinates of landmarks and history of control inputs.

The implementation of the SLAM algorithm is usually organised in two main parts: the *front-end* and the *back-end*. The front-end abstracts sensor datasets into models that are suitable for estimation, while the back-end performs inference on the abstracted data provided by the front-end [40]. In other words, the front-end is responsible for applying specific methods that generate spatial constraints (feature recognition and loop closure). These results, which are correct in principle but affected by noise, are refined in the back-end with generic probabilistic optimization algorithms, which aim to find as many matches as possible among data, while rejecting the one that seems to be similar but originates from different environmental areas. Furthermore, the back-end can provide feedback to the front-end for loop closure detection and verification [41]. The basic architecture

of SLAM is outlined in Figure 2.17

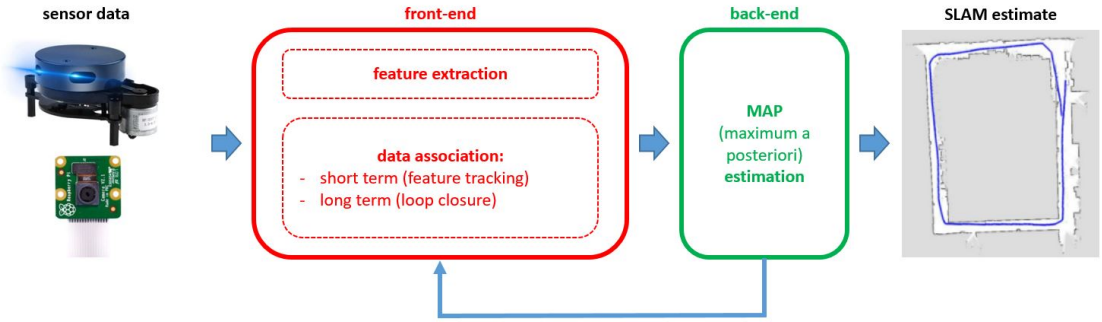

Figure 2.17: SLAM architecture

2.2.2 Review of SLAM state of the art

The SLAM algorithm exploits information from sensors that allow perceiving the surrounding environment: it is usually operated with photographic sensors like cameras, ranging sensors like LiDARs, or depth cameras. Depending on the sensors applied, it can be classified as Visual SLAM or LiDAR SLAM, which are the two most mainstream practical approaches [42].

Visual SLAM

As the name suggests, Visual SLAM (or vSLAM) uses images captured by cameras and other imaging sensors. Visual SLAM can use simple cameras (grand-angle, fish-eye and spherical cameras), compound-eye cameras (stereo and multi-cameras) and RGB-D cameras (depth and ToF cameras) [43]. The large volume of information provided by vision sensors can be leveraged to detect natural landmarks in the surrounding environment. Landmark detection, coupled with graph optimisation techniques, allows achieving flexibility in SLAM implementation. Monocular SLAM (also known as Mono-SLAM), proposed in 2007, is regarded as the origin of many Visual-SLAM techniques: in this practice, a single camera is used as the only sensor [44]. Reflecting the dualistic anatomy typical of SLAM systems, Mono-SLAM makes use of an Extended Kalman Filter as a back-end to track the feature points abstracted in the front-end. Uncertainty is expressed by a probability density function. From the observation model and recursive computation, the mean and variance of the posterior probability distribution are obtained [44]. In short, the robot motion is estimated by matching image features extracted in

different poses to construct a feature map, which can be represented as:

$$M_{feature} = \{f(x, y, z)\} \tag{2.13}$$

where $f(x, y, z)$ denotes that on the world position (x, y, z), there is a feature $f(x, y, z)$ [44].
Unlike feature-based methods such as Mono-SLAM, direct visual SLAM methods have been developed in recent years, with the aim of estimating robot motion directly through pixel values, taking into account relative brightness. LSD-SLAM (Large-Scale Direct Monocular SLAM) applies this direct method, estimating the depth values of each pixel in the image to construct a map of the environment [43].

On the other hand, SVO (Semi-Direct Monocular Visual Odometry), also called *sparse direct method*, tries to combine feature points with direct methods to track some key points in the environment, and then assesses the movement and position of the camera based on the information gathered around the key points.

RGB-D SLAM makes use of RGB-D cameras to provide both colour and depth information retrievable from the field of view, acting as a combination of a camera and a LiDAR sensor. Hence, depth images acquired through the camera are used to measure the minimum distance of each pixel in the frame; these results are then combined to obtain global map information. The camera pose is obtained by minimising the error function, which combines the intensity and depth error of the pixels [44].

LiDAR SLAM

LiDAR SLAM uses distance readings acquired by sensing devices, namely Light Detection and Ranging sensors. Compared to visual sensors, LiDAR sensors have the advantage of being less sensitive to light interference and changes in environmental conditions [45]. In addition, cameras are directional, so they detect objects placed in the direction in which they are oriented; LiDAR sensors, on the other hand, generally offer a 360-degree field of view, obtaining a complete point cloud model from the surrounding environment. Therefore, the laser sensor provides 2D or 3D point cloud data that ensures high-precision distance measurements, thus working very effectively for map construction. On the other hand, a disadvantage of LiDAR sensors is that the point density decreases with distance due to the divergence between the laser beams [46]. In addition, matching laser

point clouds generally requires high computing power, so processes need to be optimised to improve speed. Due to these challenges, LiDAR SLAM may involve the fusion of other measurement results such as wheel odometry and IMU data [43].

Early SLAM research often used LiDAR as the main sensor and the Extended Kalman Filter (EKF), which was applied to estimate the pose of the robot. However, EKF shows significant limitations in the accuracy of the results: especially for highly non-linear systems, this method produces many truncation errors, which can lead to improper localization. Particle filter approaches have been introduced to tackle such limitations in non-linear systems, however resulting in an increase in computational power proportional to the increase in the number of particles.

In 2007, Gmapping method was introduced, becoming a milestone for LiDAR-SLAM. This approach, based on an improved particle filter, allows enhancing the positioning accuracy while reducing computational complexity leveraging on adaptive resampling techniques.

As an alternative to probabilistic approaches, optimization-based methods have been introduced in the last decade: Karto-SLAM uses sparse pose adjustment to solve the problem of non-linear optimization, while Hector-SLAM – which will be discussed in Chapter 4 in further detail – uses the Gauss-Newton method to solve the problem of scan matching [44].

In general terms, with LiDAR SLAM, motion is evaluated sequentially by matching point cloud readings from laser sensors. The computed motion (robot trajectory) is then exploited to locate the robot within the scanned environment and the 2D or 3D point cloud information can be represented in an occupancy grid map [43]. Typically, the LiDAR SLAM system consists of a LiDAR odometry front-end to perform scan matching and a back-end that performs optimisation algorithms to obtain a globally optimised map and the complete trajectory of the robot. In parallel to the pre-processing of the raw front-end data, loop closure detection plays an important role, providing spatial constraints between poses captured at the same location but at different times. For the front-end, Iterative Closest Point (ICP), Normalized Distribution Transform (NDT) and Correlative Scan Matcher (CSM) are typical algorithms commonly used for matching 2D LiDAR scans: these methods aim at estimating the rigid body transformation between two laser scans, thus allowing the prediction of the robot pose [44].

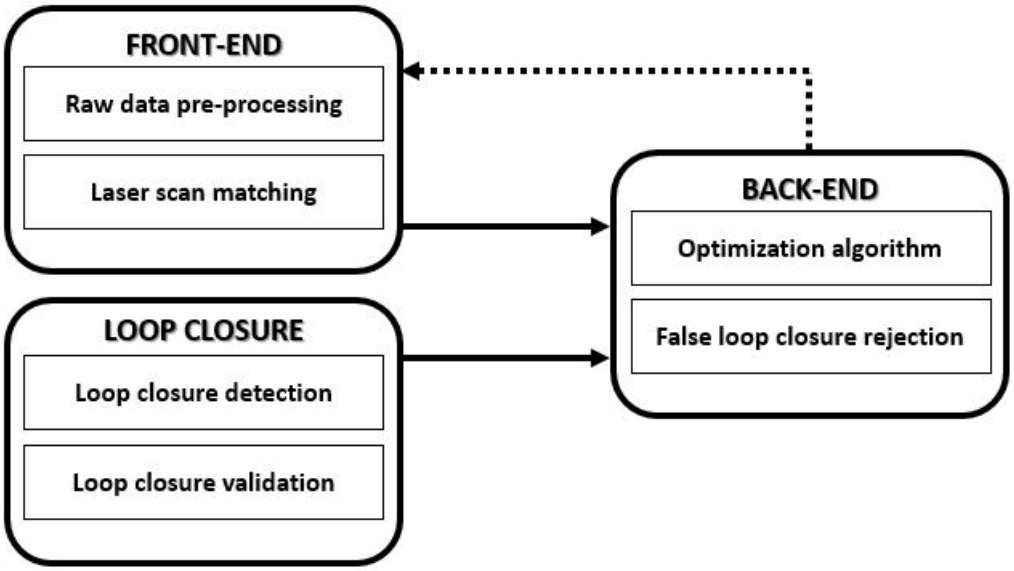

Figure 2.18: Overview of the 2D SLAM system

The main challenges in the implementation of a 2D LiDAR SLAM system revolve around three main requirements. Firstly, the front-end should be designed to be computationally efficient and robust to initialization errors, generating acceptable data models for loop closure detection and system optimization. Secondly, an efficient loop closure detection algorithm must be employed to limit incremental drifts when loops are detected. Finally, an accurate and robust optimization back-end is needed to reject false loop closures: several back-end algorithms exist, such as *incremental smoothing and mapping* (iSAM), which is a graph-based optimization method that, based on the results generated by the front-end and loop closure detection, provides a consistent trajectory as well as an optimized map.

In this regard, Occupancy Grid Maps (OGM) are the most widely used output of LiDAR SLAM techniques. It features a 2D map that represents the obstacles on the LiDAR plane:

$$M_{grid} = \{m_g(x, y)\} \tag{2.14}$$

where $m_g(x, y)$ denotes the probability of a map cell to be occupied or not. Generally, the value of $m_g(x, y)$ can be 1 if the grid cell (x, y) is occupied or 0 if the same cell is not occupied.

Chapter 3

Webots Simulations

Before addressing the implementation of the SLAM algorithm, which is the ultimate objective of this dissertation, it is also useful to explore other applications of the Bilby Rover, which are made possible by the combined installation of sensors other than the LiDAR. The structure of the robot, as a matter of fact, makes it amenable to a broad array of potential applications, notably in an industrial setting. However, it goes without saying that a detailed understanding of the sensors employed is essential: the correct execution of a given task is the product of an accurate choice of the most suitable sensors for the intended application, in conjunction with a precise calibration of the sensors involved.

In this context, a key tool that is exploited for the proper installation of sensors and for predicting the optimal behaviour of the robot and the suitability of its application is virtual simulation. Especially in industrial engineering, simulation has recently gained in prominence due to the spread of the Industry 4.0 paradigm, based on Cyber-Physical Systems, becoming progressively central to improve decision-making processes [47]. Simulation is a valuable working tool in industry, serving to study and test the behaviour of systems, thus providing a cheap, safe and fast analysis instrument for multiple applications in many fields and across multiple domains [48]. Simulation has attained a level of technology that delivers high flexibility and integration capabilities needed for product design, development and production efficiency towards a fully digital simulation environment. Simulation can be used to cover the lifecycle of a product, from conceptual design to final product assembly and testing without the need for any physical application. Industrial demand is actually increasing due to the significant benefits brought by simulation, such as cost savings, process efficiency, and shorter delivery times [49]. Simulation has been recognized as a powerful tool for visualization, planning

and strategy in several areas of research and development, playing an important role in the unfolding of the "digital factory". Simulation opens up a wide range of options for creative problem solving, allowing industrial systems to be examined and tested although they do not physically exist. The use of simulation tools makes it possible to avoid hazards, accidents, unnecessary design changes and long cycle times. All these factors make the production process smoother and, if used correctly, make the process more cost-effective [50].

Research and work on the development of autonomous robots has always been a very complex activity, involving long development times and monetary efforts. In addition, during physical tests, robots have often proved to be inconsistent with their theoretical models, showing many unexpected variables. Robotic simulation has emerged as a solution to provide a low-cost and easily accessible environment for robot development. The benefits of robotic simulation technology have been recognized by scientific and engineering communities, with applications ranging from simple robot path simulation to full robot cell layout modelling. Robot simulation is a key feature of modern, agile manufacturing systems, as it allows a robotic system to be visualized and tested, even if it does not physically exist. Fast-growing industries will increasingly rely on advanced robotics technology, which allows the prediction of the performance of robot programs generated off-line. The rising demand for short production cycles as well as high product quality requires greater flexibility in the production process. This flexibility can be reached, for example, through the integration of robots into the production shop floor. The greatest advantage of robots is their flexibility and modularity, namely their ability to be adapted to new production tasks. Effective, intuitive and fast programming are essential to take advantage of robot flexibility. Today, most industrial robots are still programmed online, but this method is not a suitable approach for modern production systems. Conversely, an off-line programming approach could be leveraged to significantly shorten programming and set-up times. In such a way, any robotic program written off-line can be tested before launching the actual production system. Robotic simulation techniques allow robotic programs to be created and optimised without interfering with production, thus reducing downtime. Furthermore, off-line programming also offers the opportunity to exploit simulation technology to test a variety of scenarios and processes. In general, simulation aids the designer of a robotic system to determine the appropriate choice of the robot and its components, including power supplies, tools, equipment and sensors [50].

In this chapter, the objective will be to simulate the Bilby Rover engaged in specific tasks, thanks to the combined installation of different sensors. For this purpose, Webots - a software dedicated to robotic simulation - will be used.

Therefore, the chapter is structured as follows: the first section is dedicated to the description of Webots software and its main functionalities. In the second section, the steps for constructing the Bilby Rover model and for importing it into Webots environment are outlined. Finally, the following sections will be devoted to illustrate some examples of specific applications of the robot, describing the enabling technologies underlining the execution of a certain task.

3.1 Webots simulator

Webots is an open-source three-dimensional mobile robot simulator developed by Cyberbotics Ltd. in collaboration with the Swiss Federal Institute of Technology (EPFL) in Lausanne. It was originally introduced as a research tool to investigate various control algorithms in mobile robotics. Since December 2018, Webots is released as an open source software under the *Apache 2.0* license.
Webots is used in thousands of universities and research centres around the world and is promoted as a reliable, thoroughly tested, well-documented technology in continuous evolution, with the aim of significantly reducing design, implementation and testing time, thereby fostering an agile development process. It is a complete development environment created with the objective of guiding the modelling, programming and simulation of mainly mobile and autonomous robots. Webots offers a rapid prototyping environment that allows the user creating 3D virtual worlds and arbitrarily complex scenarios that may include more than one robot as well as simple passive objects [51]. For each object in the simulation, it is possible to specify a series of properties, such as shape, colour, texture, mass, friction, etc. In addition, each robot can be equipped with a large number of sensors and actuators, available for use in the platform. Webots offers the possibility to program robots to exhibit the desired behavior, simulate them in realistic ways, and eventually transfer the resulting programs to real robots [52]. Actually, Webots contains a number of interfaces to real mobile robots, so that once the simulated robot behaves as expected, it is possible to transfer its control program to a real robot, like *e-puck*, *DARwIn-OP*, *Nao*, and others [51]. It also offers the possibility of reproducing high-quality simulations with an appealing graphical interface, as well as taking snapshots of the current situation or recording videos

during the execution of a simulation.

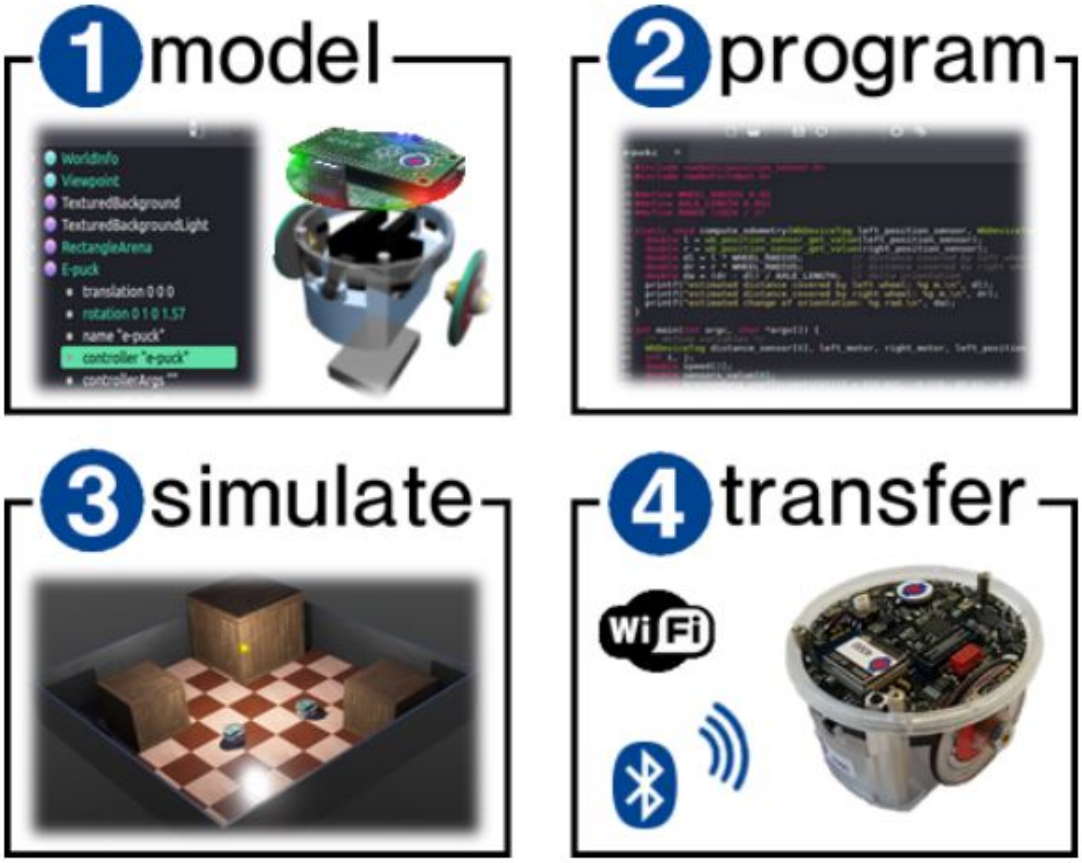

Figure 3.1: The four development steps in the Webots platform

Webots runs on Windows, Linux and Mac OS X and is suitable for research and educational projects related to mobile robotics. Many mobile robotics projects have relied on Webots for years in the following areas:

- Prototyping of mobile robots (such as in academic research, automotive industry, aeronautics, vacuum cleaner industry, hobbyists)
- Robot locomotion research (i.e. study of the performance of legged, humanoids, quadrupeds, and flying robots)
- Adaptive behaviour research (genetic algorithms, neural networks, and Artificial Intelligence)
- Multi-agent research (focused on swarm intelligence, collaborative robot groups)
- Robotics teaching (robotics, C/C++/Java/Python programming lessons)

3.1.1 Modelling

Webots provides users with the ability to create complex environments for mobile robot simulations, using advanced hardware-accelerated OpenGL technologies. In Webots, a world can be defined as a 3D representation of the properties of the robots and their environment, containing a description of each object in it - whether passive or active - specifying position, orientation, shape, type, physical properties. Worlds are organised as hierarchical structures outlined in the scene tree of the simulation environment.

Moreover, Webots offers a wide range of pre-built robot models available in its embedded library, but it also allows designing new robots freely by combining together elementary objects. In addition, Webots supports importing complex 3D models into its scene tree from the majority of 3D modelling software. There are many compatible formats, including VRML, STL, Collada, Blender, Wavefront, 3D Studio mesh.

The Webots platform comes with a library containing a wide range of sensors, which can be installed on the robot model and calibrated according to needs (including range, noise, frequency, field of view, to name a few parameters). The library includes distance sensors (i.e. infrared, ultrasonic, laser sensors), light sensors, touch sensors, global positioning sensors (GPS), force sensors, gyroscopes, compass, cameras, accelerometers, radio receivers (for inter-robot communication), incremental encoders for wheels, and others. Similarly, a library of actuators is also supplied, including motors (both linear and rotational), brakes, displays, connectors, LEDs, and speakers, to mention a few [52].

3.1.2 Programming

After the world and robots to be examined have been configured, the controller must be programmed to make the robot display the desired behaviour. A controller is a computer program that manages a specific robot in a world file. The controller can be written in one of the various languages available on Webots: the APIs originally provided by Webots are those designed for the use of the C language, but, based on these, APIs have also been provided for C++, Java, Python, Matlab. The controller can be implemented using Webots built-in development environment or any other third-party tool. However, it is worth stressing that, despite the syntactic differences between the languages mentioned above, they all share the same low-level implementation. Therefore, as long as the sequence of functions

and method calls does not change, each programming language will lead to the same simulation results.
Usually a controller process runs in an endless loop until it is terminated by Webots on one of the following events:

- Webots quits
- The simulation is reset
- The world is reloaded
- A new simulation is loaded
- The controller name is changed

In addition to the regular controller, Webots features another important element called *supervisor*. The supervisor generally performs operations that are carried out by human operators rather than by robots. Thus, in general, it is not a robot physically present in the scene but just an abstract process that has the possibility to intervene in the simulation in a similar way as a human supervisor could do during a real test. The supervisor can be written in one of the languages listed above but, unlike regular robot controllers, this type of process has access to more extensive APIs and to privileged operations. These operations allow to obtain information about any object in the scene and to modify it in an automated way. A supervisor can perform operations such as instantly moving a robot to any position, stopping and restarting the simulation, taking snapshots, recording videos of the simulation, and so on. The main task entrusted to a supervisor is therefore to manage the simulation and to process specific data in favour of the whole development process: actually, a supervisor is often employed to retrieve statistics on the controller performance to be used in optimisation algorithms.

3.1.3 Simulation

Webots relies on the concept of *virtual time*, which makes it possible to reproduce a simulation at a much faster speed than the real one, up to, in the extreme case, fully exploiting the capabilities of the hardware on which it is running. The elementary time interval used in the simulation process is called the *simulation step* and can be configured according to needs, depending on whether greater precision or greater speed in completing the simulation is being sought [52].

Webots employs the *Open Dynamics Engine* (ODE) for the simulation of physical

laws and interactions between objects in the scene. A wide range of objects can compose the scene to be simulated including robots, light sources of various types, floors, walls, obstacles, and many other entities. The properties of each object, such as shape, appearance, mass and friction coefficients, are individually configurable and are then used during the simulation to reproduce the desired characteristics of the corresponding real objects. If the standard operation of the ODE is not sufficient for certain purposes or if it is necessary to introduce special physical laws (e.g. viscous friction in fluids), the physics simulator can be extended by means of plug-ins that can be programmed by the developer and tailored for the specific task [51].

In Webots, the operations of the controller are based on a cyclic process in which, at each iteration, the inputs provided by the sensors are collected and then processed to generate outputs sent to the actuators. As far as simulation is concerned, the controller and simulator run as two independent processes that communicate with each other. At each repetition, the controller must actively request the intervention of Webots in order to carry out the simulation of a time interval, called control step, which substantially represents the time duration of the controller iteration. During this operation, Webots will actually put the controller program into practice by updating the status of the actuators and evaluating their interactions with other objects. The effective duration of this operation depends on the speed at which Webots is performing the simulation: it can be much longer, (approximately) equal or even shorter than the control step, depending on whether the requested simulation is particularly computationally demanding or not. At the end of each operation step, the simulation status is refreshed and the updated value assumed by the sensors is transferred to the controller program.

3.1.4 Transfer to real robots

Once the simulated robot behaves correctly in the virtual world, the controller can be transferred to real mobile robots. Webots was originally designed with this distinguishing transfer capability in mind, providing a programming interface easily portable to physically existing robots. Webots already comprises transfer system for a series of robots including *e-puck*, *DARwIn-OP*, *Khepera*, and *Hemisson*. Obviously, since the simulation is only an approximation of the physics of the real robot, some tuning is always necessary when developing a transfer mechanism, which will affect the simulated model so that it better matches the behaviour of the real robot [51].

Often, the simplest approach to transferring the controller program to a real robot is to develop a remote control system. In this case, the control program is executed on the computer, but instead of sending commands and reading sensor data from the simulated robot, it sends commands and reads sensor data from the real robot. The development of such a remote control system can be achieved by writing the implementation of the Webots API functions as a short library. Webots already provides some facilities to implement a remote control library: this can be exploited as a plugin to a controller, which, once associated with the Robot node, will be executed automatically when running the controller [51].

Cross-compilation is the main alternative to the remote control method. The development of a cross-compilation system will allow the Webots controller to be compiled into a processor embedded in the robot platform. Thus, the source code written for the Webots simulation will run on the real robot itself, and there will be no need to have a permanent PC connection to the robot, as with the remote control system. However, this method is only possible if the robot processor can be programmed in C, C++, Java or Python. Unlike the remote control system, the cross-compilation method requires the source code of the Webots controller to be recompiled using the cross-compilation tools specific to the processor installed in the robot [51].

3.2 Bilby Rover modelling on Webots

As discussed in Section 3.1, the first task to fulfil before running a Webots simulation is to correctly setup the robot model. Owing to the fact that the Bilby Rover model is a custom design developed by the W Booth School of Engineering Practice and Technology at McMaster University, the robot is not available and ready to use in the standard robot library on the Webots simulator platform. To tackle this issue, the software offers two main solutions to design a customised robot: one is to model the robot by aggregating simple elementary components together to obtain the desired shapes (elementary components such Box, Sphere, Cylinder, and Capsule), and the other is to import the robot model from third-party 3D modelling software. The first alternative is undoubtedly the most convenient solution if the structure of the robot is fairly simple and does not feature elaborate geometries. A representative example is the robot shown in Figure 3.2.

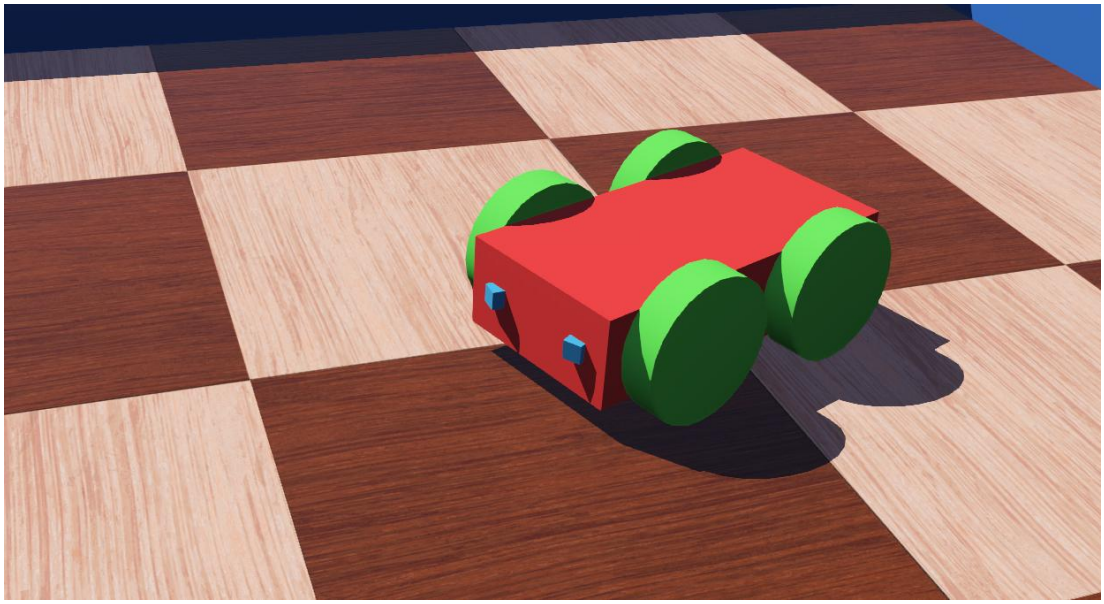

Figure 3.2: 3D view of a sample four-wheeled robot built with elementary solids

As it can be observed, the robot is made up of a box-shaped body, four cylindrical wheels and two distance sensors mounted in the front part. This robot, which is mainly used for educational purposes and to practice with the simulator platform, does not pose any particular criticality in its structure, being made up of elementary geometric solids, such as boxes and cylinders. It goes without saying that for the construction of similar models, the simplest solution is to aggregate elementary components directly by modelling the robot in the Webots scene. Conversely, when it comes to modelling more sophisticated robots, this solution presents important limitations, at the expense of the accuracy of the resulting model and of the time required to build a robot using only elementary geometric shapes. To give a rough idea, one might think of constructing a vehicle suspension or a gearbox by progressively aggregating elementary solids. However, although feasible in theory, the design of these components using such approach would result, in practice, in a model with poor structural accuracy and, above all, would take much longer than using 3D modelling software specifically dedicated to mechanical design. Therefore, in the case of robots with a particularly complex structure, the most convenient solution is to design the model in dedicated third-party software and then import it into the Webots platform. In the light of these considerations and bearing in mind the complexity of the Bilby Rover structure, the procedure to follow is definitely the latter.

Before importing the robot, it is necessary to create on the Webots simula-

tor a new world, which is a file containing information such as where the objects are located, what they look like, how they interact with each other, what the background is, how gravity, friction, masses of the objects, and other parameters are defined. In general, the world file defines the initial state of the simulation. The different objects present in the world file are called *nodes* and they are organized hierarchically in a tree structure named *Scene tree*; on their turn, nodes may contain many sub-nodes. The graphic interface can be thus divided into two main sub-windows: the 3D window, which is the 3D representation of the Scene tree, and the scene tree view, that is the hierarchical schematization of the Scene tree, where the different nodes can be visualized and modified. After world setup, it usually lists the following nodes:

- *WorldInfo*, containing the global parameters of the simulation
- *Viewpoint*, defining the main viewpoint camera parameters
- *TexturedBackground*, defining the background of the scene
- *TexturedBackgroundLight*, defining the light associated to the background
- *RectangleArena*, defining a floor surrounded by walls

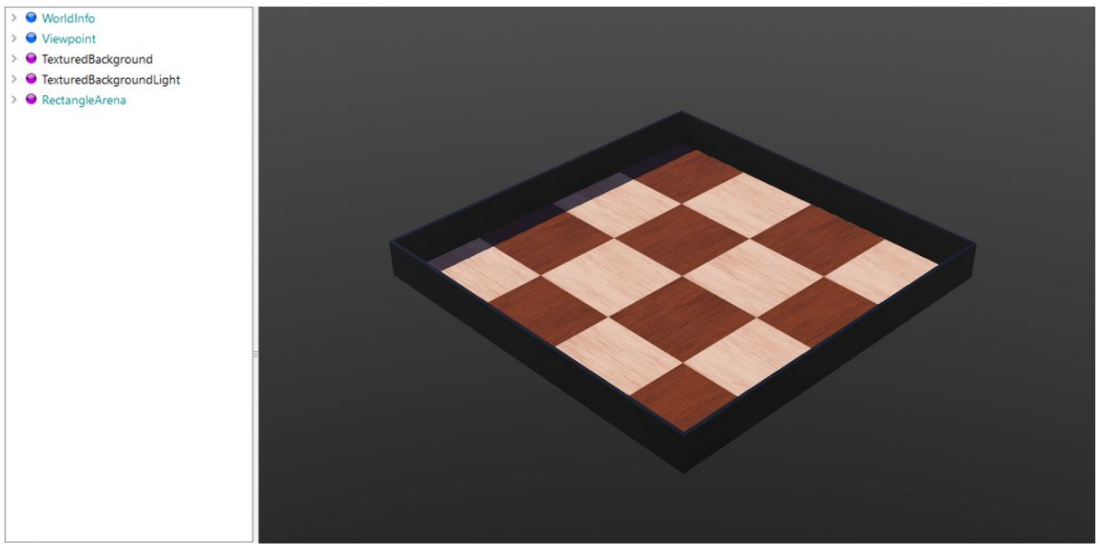

Figure 3.3: Webots Scene Tree and 3D Window showing a rectangular arena

Once the empty scene is created, it is necessary to import the robot model into Webots platform. However, before proceeding with this operation, it is necessary to clarify a number of rules that must be observed to configure the hierarchical

structure of the robot on Webots.

Webots platform comes with a library of different nodes, corresponding to different real or virtual entities in the structure of a robot. A *Solid* node represents a rigid body, namely a body whose deformation can be neglected. Soft and articulated bodies are not rigid bodies. For instance, a wheel rim is a rigid body, whereas a tyre or an articulated robotic arm is not. Webots physics engine is designed to simulate only rigid bodies. Therefore, an important step when planning a simulation is to decompose the various entities into separate rigid bodies [51]. To define a rigid body, it is thus necessary to create a Solid node, within which it is possible to define several sub-nodes corresponding to the characteristics of the rigid body in question. The graphic representation of the Solid node is specified by the *Shape* nodes that populate its list of children. The collision boundaries of the body are set in its *boundingObject* field. Finally, the *Physics* field defines whether the object belongs to the dynamic or static environment [51].

Two distinct solids within the robot structure are connected by *Joint* nodes, which are used to add one or more degrees of freedom between its parent and its child. Therefore, nodes derived from Joint enable various types of constraints to be created between connected Solid nodes. The most widely used in robotics is the *HingeJoint* node, which allows configuring rotational motors including wheels. In addition, a Joint node can be monitored or actuated by adding a *PositionSensor* node or a *Motor* node to its device field. Finally, devices such as sensors and actuators should be direct children of a Robot, Solid or Joint node.

Having these rules in mind, it is possible to start designing the node hierarchy used to model the robot. The first step is to determine which part of the robot should be modelled as a Solid node. In this specific project, the operation is almost straightforward and the Bilby Rover can be divided in six solid nodes: the body (including covers, the underbody frame and driveline made up of belts and pulleys, which will not be considered as active components for the sake of the simulation), the LiDAR, and the four wheels. The second step is to determine which Solid node is configured as a Robot node (parent entity) in the Scene Tree. Such choice is usually arbitrary, even if a solution is often much easier to implement. In this case, the Bilby Rover body is assigned to the Robot node: such choice allows exploiting the symmetry of the structure, aiding the computation of joint parameters and the estimation of the devices position during their installation. Based on this choice, joints (for wheels) and devices (i.e. LiDAR and other sensors) will be set

as children of the Robot node. Figure 3.4 and Figure 3.5 depict respectively the high-level and low-level representation of Bilby Rover solid nodes hierarchy.

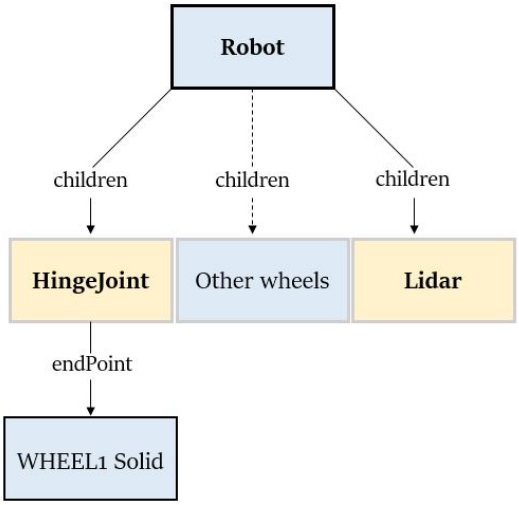

Figure 3.4: High-level representation of Bilby Rover nodes hierarchy

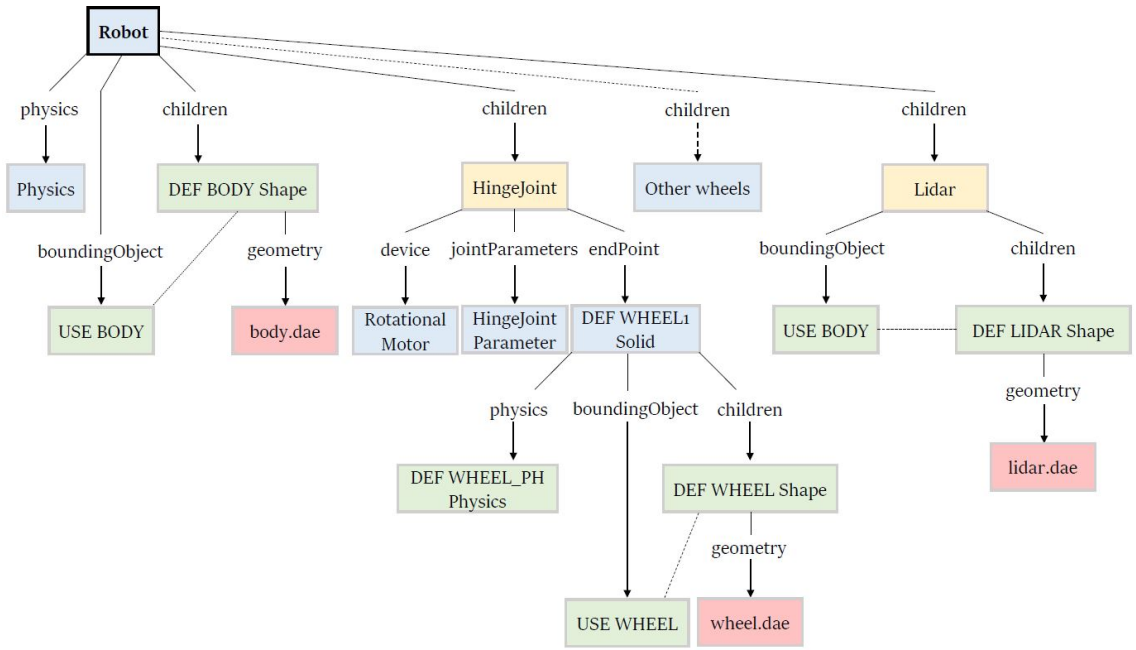

Figure 3.5: Low-level representation of Bilby Rover nodes hierarchy

Based on the graph shown in Figure 3.5, the same structure must be replicated in the scene tree on Webots. For this purpose, the structure of the robot must be broken down into the six distinct solids (the body, the LiDAR and the four wheels) on SolidWorks, where the model has been designed by the McMaster's W Booth School of Engineering Practice and Technology.

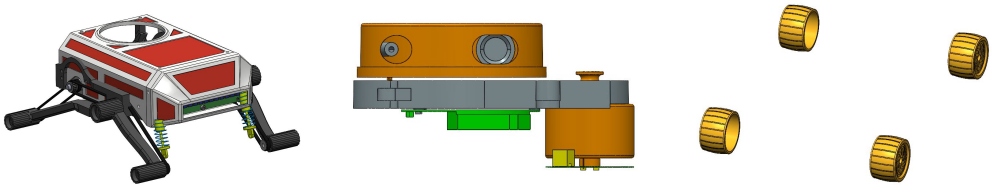

Figure 3.6: Isolated components (body, LiDAR, wheels) imported into Webots

Subsequently, these main components are exported separately as Collada files and then imported into Webots using a designated function in the file menu. Once imported, the files are then nested inside the relative node within the robot hierarchical structure, coinciding with the position indicated by the reddish rectangle in Figure 3.5. In this regard, it is necessary to specify the coordinates of the position of the wheels and the LiDAR sensor with respect to the reference system attached to the root node, namely the body frame.

At this point, the last step is to establish the parameters associated with both the HingeJoint and the Lidar node. For each HingeJoint there are three fields that must be filled in:

- *jointParameters*, in which a *HingeJointParameters* node is created, allowing the anchor and axis of rotation of each wheel joint to be defined
- *device*, in which a *RotationalMotor* node is introduced to provide wheels actuation. The *RotationalMotor* nodes must be labelled with a specific name for each wheel, so that they can be called and instantiated in the controller codes to forward commands.
- *endPoint*, where a Solid node is added, containing the wheel shape imported from the 3D modelling software, according to the procedure described above.

As far as LiDAR is concerned, the relative node comes with a series of fields that allow the setting of the technical specifications of the sensor to mirror the actual functionalities of the real device. In the light of the YDLIDAR X4 data sheet (see Section 2.1.2), some relevant fields of the Lidar node are configured, such as field of view, horizontal resolution, minimum and maximum range and measurement resolution. Again, the Lidar node is labelled with a name to be referenced by the controller.
Once these steps are implemented, the Bilby Rover setup will be properly configured on Webots and will result as depicted in the Figure 3.7.

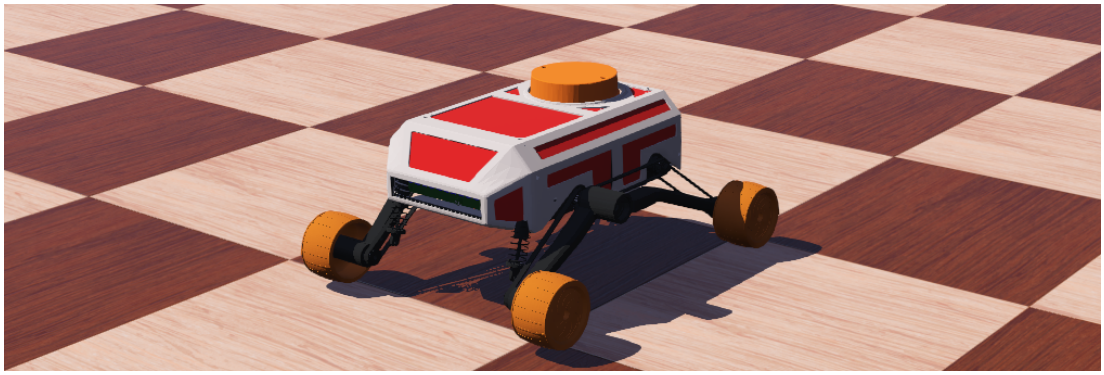

Figure 3.7: Bilby Rover in Webots simulator

To enable the desired behaviour of the robot, a controller must be assigned to it. The controller can be programmed using different languages, such as C, C++, Java and Python and the source code is shown and can be edited in a window in the Webots simulator interface.

The robot node has a field called controller that defines which controller is currently associated with the robot. Therefore, the procedure is to generate the programming code according to needs and depending on the sensors involved in the particular application and store it in the project folder. Then, the same controller must be associated via the appropriate field in the robot hierarchical structure, allowing the system to behave as desired.

3.3 Bilby Rover applications

The objective of this section is to describe the development of a set of simulations aimed at studying applications made possible by the integrated use of sensors installed on the Bilby Rover. Although the ultimate goal is to implement the SLAM algorithm for the physical robot, virtual simulations of other applications are a useful tool to give an overview of the range of functionalities of the robot, highlighting its potentials and limitations.

Therefore, in the following paragraphs a number of examples that have been developed on the Webots simulator will be proposed. Although the range of applications could be manifold, the focus is on some functionalities and behaviours that could be useful in an industrial scenario, such as for shop floor operations.

3.3.1 Keyboard Teleoperation

The purpose of this application is to simulate the teleoperation of the Bilby Rover. In simple terms, teleoperation refers to the task or activity itself of operating a vehicle or system remotely [53]. The research conducted in the area of robotic teleoperation is extensive and has been explored since the mid-twentieth century. The research is strongly interdisciplinary and tackles a broad variety of aspects regarding human-robot interaction, mobile robotics and visualization. Originally, research on teleoperation was conducted to enable human operators to work remotely in hazardous environments. Telerobots are generally mobile platforms that accept instructions from a remote human operator. Furthermore, in addition to receiving commands from the operator, robots can be equipped with specific sensors and actuators that allow them to perform complex tasks, such as object recognition, mapping, exploration and localization. Due to the variety of applications, teleoperation has been widely used in recent decades in different fields, such as space exploration, mining, medical, surveillance, inspection in environments restricted to humans and rescue [54].

A teleoperation setup is usually a master-slave system: in such a configuration, the master device is controlled by the operator and the slave device is placed in the remote side [54]. The key components needed to develop telerobotics applications are control (algorithm and real time implementation), sensors (i.e. world sensing and information processing) and wireless communication, which are interconnected as showed in Figure 3.8.

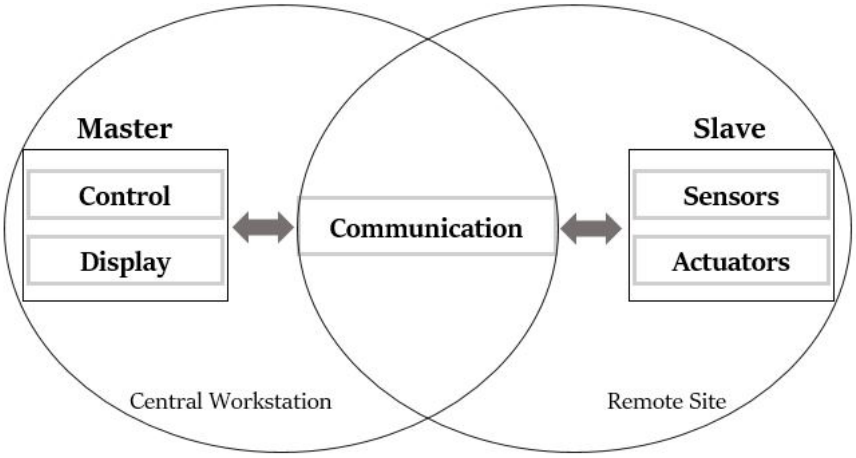

Figure 3.8: General block diagram of a mobile robot teleoperation system

By definition, the human operator manually interacts with the teleoperation

system via the input device, whereas the telerobot senses and performs actions in the far distant environment. There are many devices used to control the remote system in teleoperation, such joystick, keyboard, mouse and touchscreens.

In the present simulation, the intention is to implement a simple algorithm to enable teleoperation with a low Level of Automation (LOA). The low LOA implies that the human operator has direct manual control of the robot by providing the input command through a dedicated peripheral device [53]. More precisely, the user will be able to command the movement of the Bilby Rover via the keyboard: by pressing the arrow keys on the keyboard, the user will send commands to the robot, which will move in the specified direction.

To develop the teleoperation system, a new controller code is created and inserted into the specific field in the robot hierarchical structure. In this respect, it should be pointed out that the Webots API (Application Programming Interface) has an extensive library, whose classes must be included in the code in order to call up the functions available for the various simulated devices. By calling up the Robot node library, its relative functions are enabled in order to interact with components such as sensors and actuators. Similarly, the Keyboard library must be invoked to trigger a series of functions that allow the controller program to read keys pressed on the computer keyboard when the Webots 3D window is selected and the simulation is running [51]. During the simulation, therefore, the user is asked to press the arrow keys to indicate the direction in which the robot should move. The controller code then receives the keyboard input and, based on the designated direction, sets the motors' speed to perform the desired manoeuvre.

When setting the speed given to the motors, it is necessary to take into account the kinematic model of the Bilby Rover, briefly outlined in Section 2.1.1. Actually, as the structure features four conventional non-steering wheels, the Bilby Rover is a non-holonomic system, having only one controllable degree of freedom, namely the longitudinal x-coordinate. Consequently, the robot will not pose any criticality in moving back and forth, since the movement is directly controllable through the actuation of the wheels. On the contrary, the steering manoeuvres cannot be performed by direct control of the θ-coordinate, but they are carried out through a combination of speeds given to the four wheels, according to the equations given in Equation (2.10). For this reason, when the up and down arrows are pressed, the controller assigns the same speed to all four wheels. On the contrary, when the user presses the side arrows, the controller calculates the speeds

according to the Equation (2.10): the differential speed between the left and right wheels will allow the robot to turn.

However, it should be stressed that, given the configuration of the system, the instantaneous centre of rotation will not fall within the mobile platform, since the robot cannot perform complete turns on itself, as it happens with differential drive robots or system equipped with mechanum wheels. Therefore, it must be kept in mind that, given the inherent complexity of trajectory control, the robot must be used with caution in restricted spaces or in abrupt manoeuvres.

Furthermore, in order to enhance the accuracy of trajectory control, it is necessary to find a workaround to avoid wheel slip in the longitudinal direction. In fact, in case of slippage, the robot could respond with delay to the user's commands, as the wheels would turn freely due to the lack of the proper friction to move the robot in the desired direction. To compensate for this problem, a higher friction between the rear wheels and the ground is set in the Webots simulation. This expedient, which in practice corresponds to using rear wheels made of a different material compared to the front ones, allows the robot to turn more smoothly and to increase responsiveness to user commands by reducing the possibility of longitudinal slippage.

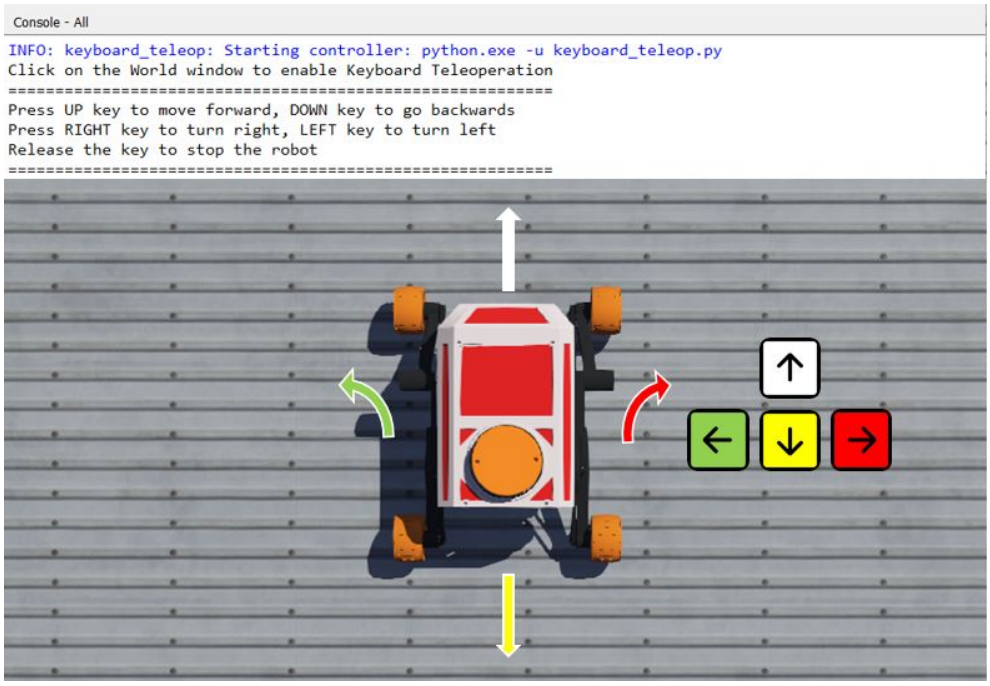

Figure 3.9: Teleoperated robot simulation on Webots

3.3.2 Line Following

The aim of this section is to describe the instructions and steps to implement a basic line following behaviour for the Bilby Rover. Line-following robots have been deployed in many industrial and commercial applications for quite a long time. The drive to use these systems for motion dynamics is motivated by their inherent simplicity of construction and programming. These robots can be flexibly configured for any track layout within a factory, commercial space and even hospitals. The use of line-following robots in factories is an integral part of automating the handling of materials and semi-finished products [55]. Traditionally, jib cranes, conveyor belts, overhead cranes and forklifts have been used to transfer material or products between different workstations or warehouses. Over the years, however, such material handling equipment has showed some significant limitations: actually, forklifts require fuel and tank replacements, with increasing maintenance costs, jib-cranes and conveyor belts take up space in the plant shop floor, and overhead cranes can be very hazardous for the employees sharing the same working area. Currently, with Industry 4.0 at its peak, autonomous technologies are used in every aspect of product development, logistics and management. The use of line-following robots represents a promising solution to overcome these issues and delivers valuable results for the efficient transportation of work-in-process good on factory floors [55]. In general, the hardware of line following robots consists of the robot body, motors, sensors and control boards, while the software handles the programming code, sensors parameters, and the controller design to communicate with the sensors and actuators installed on the robotic system.

The sensors play a fundamental role in the implementation of the line following behaviour. For this application, vision sensors must be mounted on the robot platform to detect and differentiate the colour of the path from the one of the surroundings, and, in turn, they allow the robot to adjust its trajectory to follow the track. In the present simulation, the line to be followed has to be segmented from the factory floor using a set of two infrared sensors, working simultaneously. In general, infrared sensors are equipped with two distinct diodes: one is in charge of sending IR rays (transmitter), while the other receives back the reflected rays (receiver). The specificity of infrared sensors is closely linked to the absorption or reflection properties of the different materials that are affected by IR radiation. Indeed, if the IR rays impact on a white surface, they are completely reflected, whereas if the rays are directed at a black surface, they are absorbed or slightly reflected. At the level of the sensor, which is equipped with specific circuitry,

the different amount of light that is collected by the receiver is translated into a different signal, namely into a higher (white surface) or lower (black surface) output voltage. The difference in voltage, therefore, is the discriminating element that permits to distinguish the colours that are captured by the sensors and, consequently, provides the necessary inputs for the line-following action to take place [56].

On Webots simulator platform, a factory environment is created, featuring a black closed path traced on the white floor.

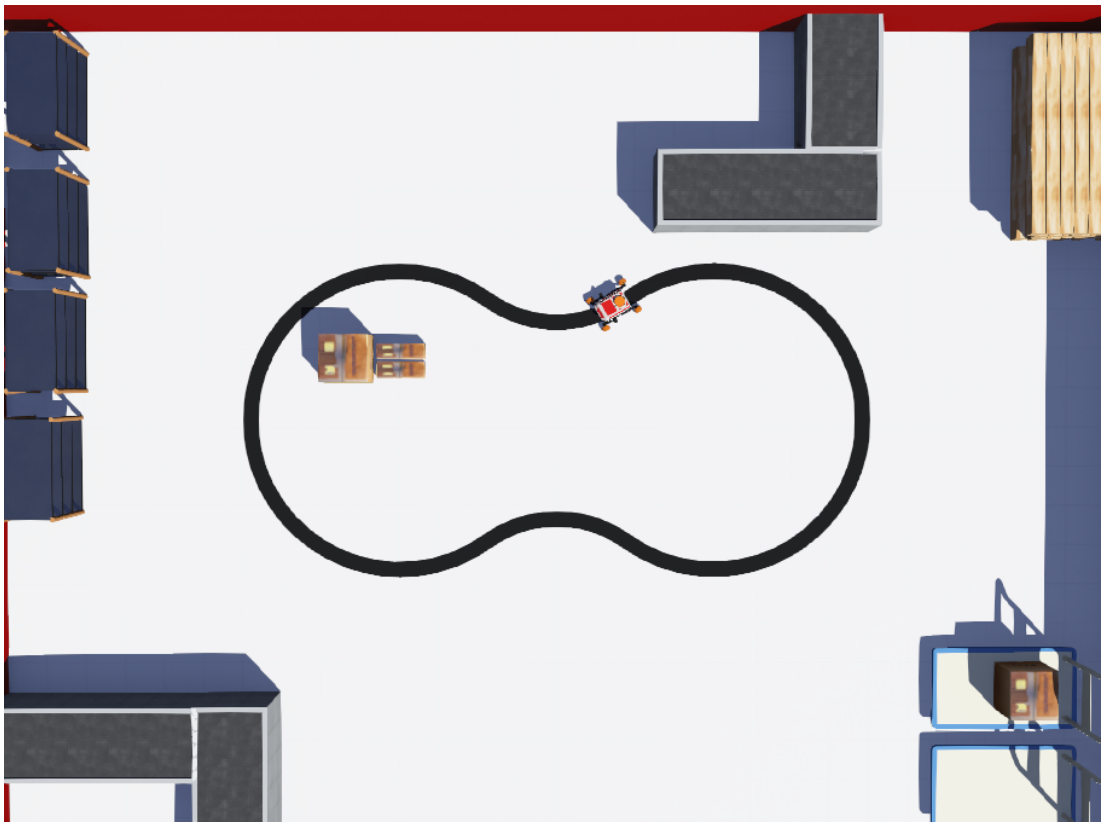

Figure 3.10: Webots scene for line-following simulation

Considering the line-following task to be accomplished, the two infrared sensors are installed in the front part of the Bilby Rover platform, pointing downwards on the ground. The sensors are placed 70 mm apart from each other and symmetrically with respect to the longitudinal plane: given that the width of the designed path is equal to 100 mm, the configuration of the sensors ensures that their rays will stay always within the black line.

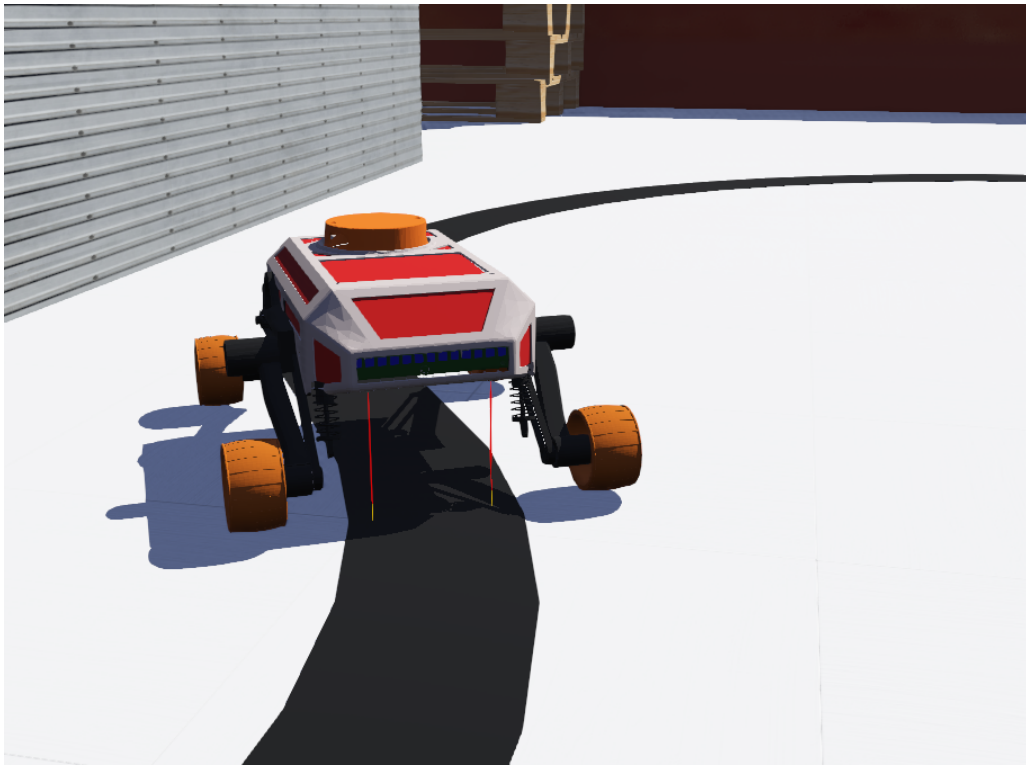

Figure 3.11: Infrared sensors configuration (IR rays are the red lines)

At this point, once the scene and the robot have been correctly configured, it is necessary to create the controller code to enable line-following behaviour. The code can be divided into two main parts:

- *Initialization*: in this section, the different API functions are called from the library and instantiated. In this code segment, the sensors and actuators of the Bilby Rover are initialised and triggered, thus allowing them to be controlled in the main cycle of the program.

- *Actuation*: this part of the code is inherent to the execution and implementation of the previously initialised sensors and actuators. This code segment is placed in a loop that is repeated indefinitely until the simulation is stopped. In this loop, the controller recursively receives output data from the infrared sensors and, based on the returned values, issues commands to the motors to adjust the trajectory, allowing the line to be followed.

The basic algorithm is fairly straightforward: the controller continuously reads the output values from infrared sensors, through which it can figure out whether the device beams are bumping into the black line or the white floor; as a matter of fact, the output values returned by the sensors will be different. Therefore,

as long as both sensors intercept the black line, the controller drives the wheels with the same speed in order to move the robot in the forward direction. On the contrary, as soon as one of the two sensors reports that it is intercepting the white floor, the controller issues commands to the motors setting a speed differential between left and right wheels, thus activating the steering mechanism to adjust the trajectory:

- If the left sensor detects the white colour, the controller sends the right motors the command to rotate in the opposite direction with respect to the left ones, thus steering the robot to the right.
- If the right sensor detects the white colour, the controller sends the left motors the command to rotate in the opposite direction with respect to the right ones, thus steering the robot to the left.

Figure 3.12 provides a basic schematic of the line following behaviour, showing the black path, the robot trajectory and the sensing module installed on the Bilby Rover.

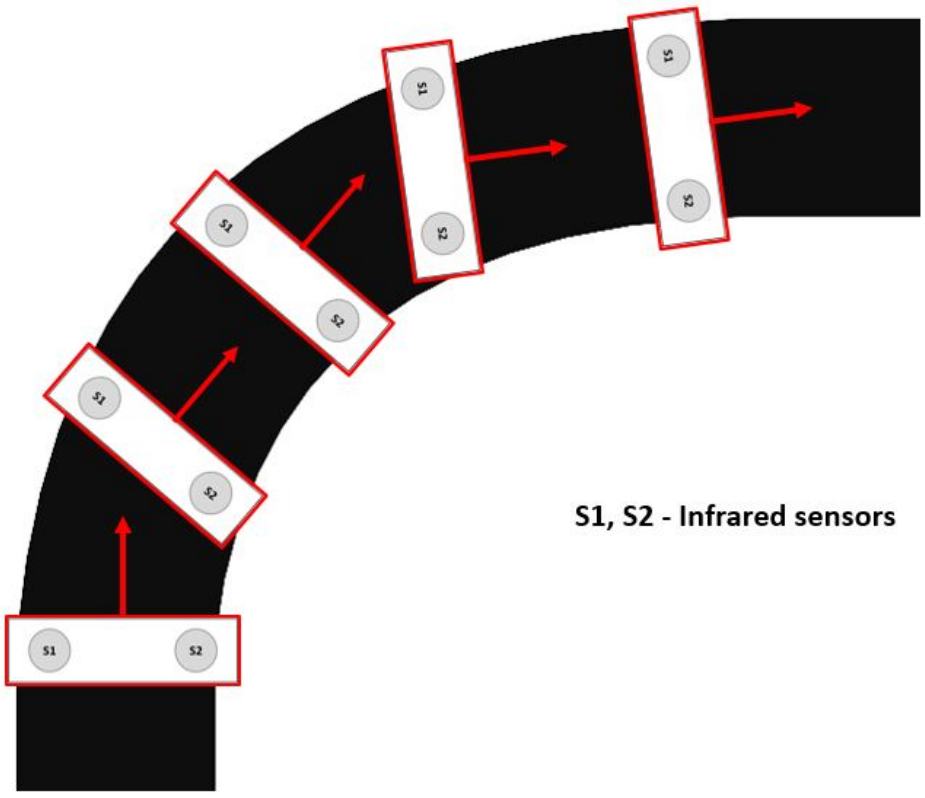

Figure 3.12: Line-following behaviour with infrared sensors

Therefore, this mechanism enables the robot to travel over the black path, turning left or right and even taking U-turns. The infrared sensors allow the

robot to perceive the world and to guide its motion by differentiating the line colour from its surroundings. This difference between black and non-black pixels is detected and processed to accede the motion of the robot [55].

Finally, referring to a possible future work aimed at improving the robot performance, it would be also necessary to address the safety issue. In this context, a suggestion to improve the performance of the robot would be to install a set of distance sensors (e.g. ultrasonic sensors) that allow the robot to perceive obstacles in its path. The integration of these devices, if appropriately programmed with a controller code, allows the robot to travel autonomously along the path and to stop to avoid collision with an obstacle. This measure would increase the safety of the robot operations, especially in environments shared with human operators.

3.3.3 Lane Keeping and Object Recognition

The aim of this section is to develop a simulation in which the Bilby Rover is made to move along a lane delimited by side lines and stop if an obstacle is detected along its path. Therefore, the simulation requires the implementation of two specific functions to be performed in parallel: on the one hand, the robot must be configured and programmed to move while following the lane; on the other hand, the robot must be equipped with specific visual sensors that, appropriately calibrated and coded, enable the scanning of the surrounding environment and the identification of the presence of any obstacles.

To address this twofold purpose, two types of visual sensors are used as enabling technologies: infrared sensors and a smart camera. Respectively, the infrared sensors are responsible for providing the necessary data for the lane-keeping task, while the smart camera is the device that makes obstacle identification possible.

The logic behind the application of ultra-red sensors for lane keeping is almost the same as the one illustrated in Section 3.3.2. Leveraging the light reflection and absorption properties of different colours, infrared sensors enable the robot to differentiate the track from the two side lines, thus allowing it to adjust its trajectory, by means of the feedback mechanism depicted in Figure 3.13. Actually, depending on the colour inspected by IR rays, infrared sensors output different values (proportionally to the amount of light reflected back to the receiving diode), which are used as discriminant elements to distinguish colours.

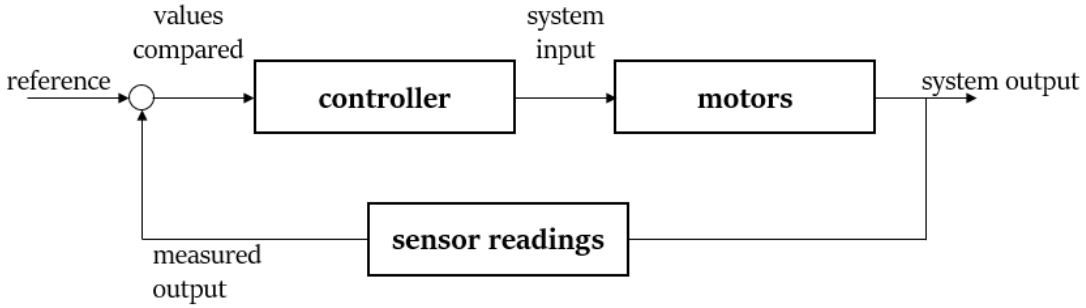

Figure 3.13: Control loop for the lane-keeping task

So, as soon as the sensors sense the presence of the white line, which is a warning of unintentional lane departure, the motors are actuated to activate a steering mechanism to direct the robot back to the center of the lane. The turning direction for the steering manoeuvre is established according to the sensor intercepting the white line: for instance, if the sensor installed on the left-hand side identifies the line, the motors will be actuated to steer to the right, and vice versa. For the sake of clarity, the algorithm underlying the lane-keeping capability using two infrared sensors can be illustrated by the flowchart in Figure 3.14

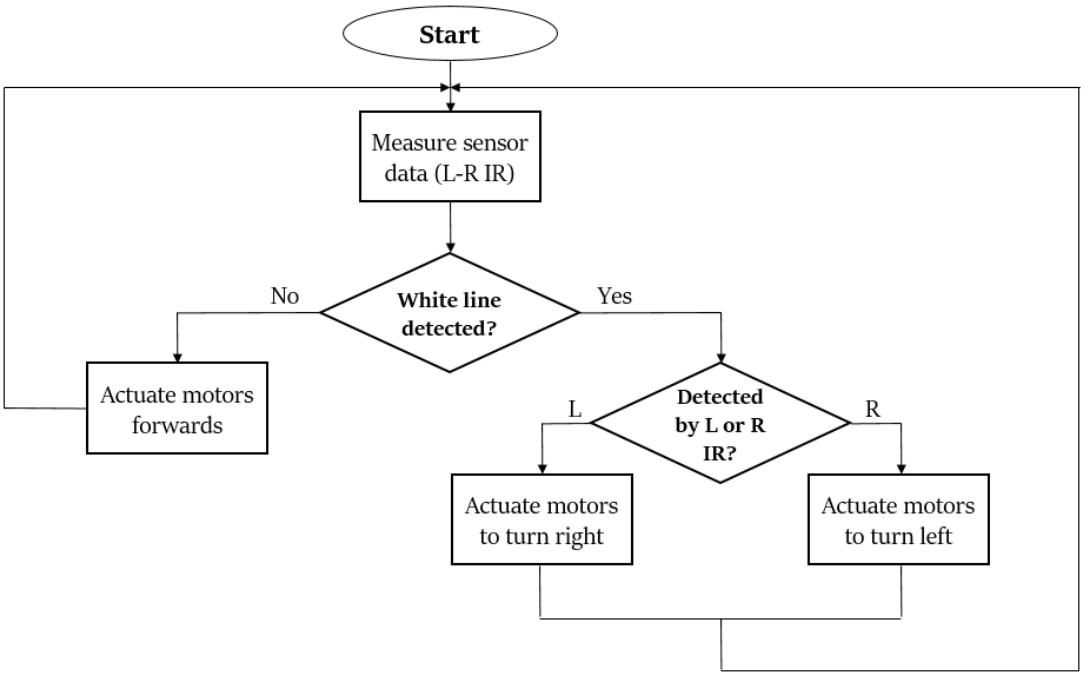

Figure 3.14: Lane-keeping algorithm flowchart

While lane keeping is based on the harnessing of infrared sensors, the object

recognition function is enabled by the installation of a smart camera on the robot body.

Since the 1990s, smart cameras have achieved remarkable popularity and market acceptance, notably in the surveillance and machine vision industries. Owing to intelligent image processing and purposely-engineered pattern recognition algorithms running on powerful microprocessors, a smart camera can perform tasks beyond simply taking photos and videos, such as image analysis and event/pattern recognition [57], thus representing a key technological enabler to build active and automated control systems for many applications.

What differentiates smart cameras from general purpose ones is the type of tasks performed by the embedded image processor and the output generated by the device. In this regard, a smart camera can be defined as a vision system that is capable of extracting application-specific information from captured images, generating event descriptions or decisions that are fed into an intelligent, automated system. The ultimate goal of smart cameras is to implement a mechanism that can mimic the functioning of the human eyes and brain, thereby making sense of the captured images through artificial intelligence [57]. It is worth stressing that a camera that comes with an integrated image processing capability does not necessarily qualify as an intelligent camera, as this attribute is strictly dependent on the purpose of the image processing. Actually, although many consumer digital cameras are equipped with built-in image processing capabilities (such as focus, exposure control, image compression, to name a few), most of them are mainly targeted just at producing better quality images. By contrast, the main purpose of image processing in smart cameras is to generate descriptions of events and guide decision-making processes in an automated control system.

The construction of intelligent cameras requires the integration of computer vision, machine vision and embedded systems technologies. Both machine and computer vision are concerned with building devices or systems that can retrieve valuable information from images and take decisions based on the acquired data. Embedded systems technology, instead, is dedicated to building systems that are low-power, low-cost and robust enough to operate reliably in real-world conditions. Figure 3.15 provides the simplified functional diagram of a smart camera.

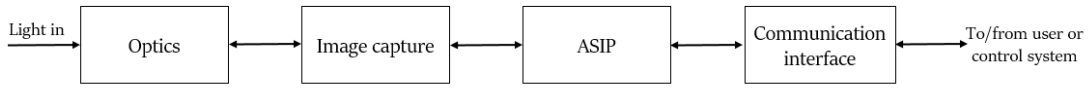

Figure 3.15: Simplified functional diagram of smart camera (Source: [57])

Intelligent cameras feature integrated optics that are designed to efficiently capture light, which is then forwarded to an image capture block, typically consisting of a solid-state image sensor and associated circuitry to implement the conversion from light to a digitized image array. The digital signal is then processed by the *Application-Specific Information Processing* (ASIP) block, which is responsible for understanding and describing what is happening in the captured images, generating information that aids decision-making in the intelligent control system. Finally, the communication interface is capable of receiving instructions from the host via the I/O ports and, in turn, transmitting data or issuing commands to the user or control system. Recalling the parallelism with humans, while the optics replicate the function of human eyes, the ASIP block can be suitably compared to the brain. On the hardware side, the ASIP block incorporates one or more microprocessors with associated memory, communication buses and other components. On the software side, the ASIP executes advanced image processing algorithms to gather relevant information, identify patterns, detect events and generate inputs for decision-making processes. In general, both hardware and software must be designed specifically for the selected application, taking into account technical requirements and performance specifications [57].

In this specific application, smart camera technology is leveraged to identify obstacles along the route of the robot: the vision sensor, suitably configured for this purpose, captures images of the surrounding environment and alerts the control system if an object is detected. This alert triggers the response of the control system, which stops the robot at a safety distance and lets it move again as soon as the track is free from any obstruction.

On the Webots simulation platform, a factory environment is created, featuring a closed lane delimited by two white stripes running around a sample production department.

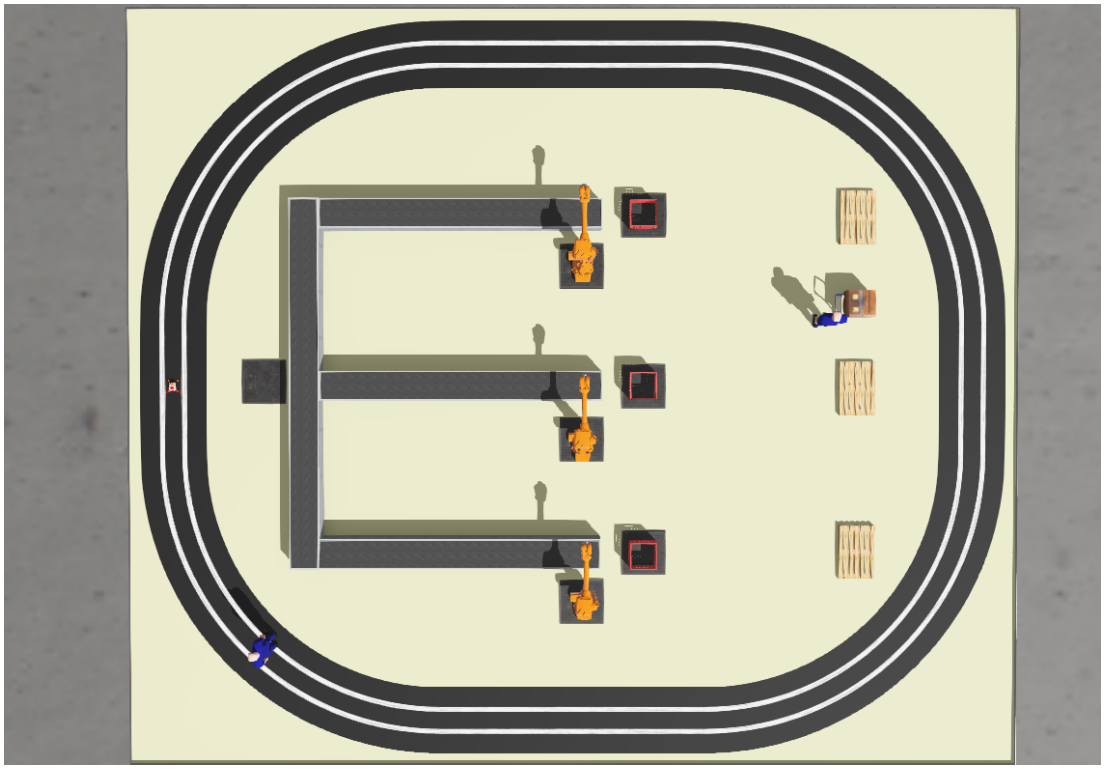

Figure 3.16: Webots scene for lane-keeping and obstacle detection simulation

Considering the lane-keeping task to be accomplished, the two infrared sensors are installed sideways in the front part of the Bilby Rover platform, pointing towards white lines. The sensors are placed 120 mm apart from each other and symmetrically with respect to the longitudinal plane: given that the width of the designed lane is equal to 1.5 m, the sensors are suitably oriented to ensure their rays to stay always within the track delimited by white lines. In addition to the infrared sensors, a *Camera* node is included in the Bilby Rover tree diagram. The device is installed in the front panel of the robot and oriented to capture the environment in front of it. Figure 3.17 illustrates the integrated installation of the aforementioned sensing devices. The window overlay that appears on the top right shows the image captured by the smart camera.

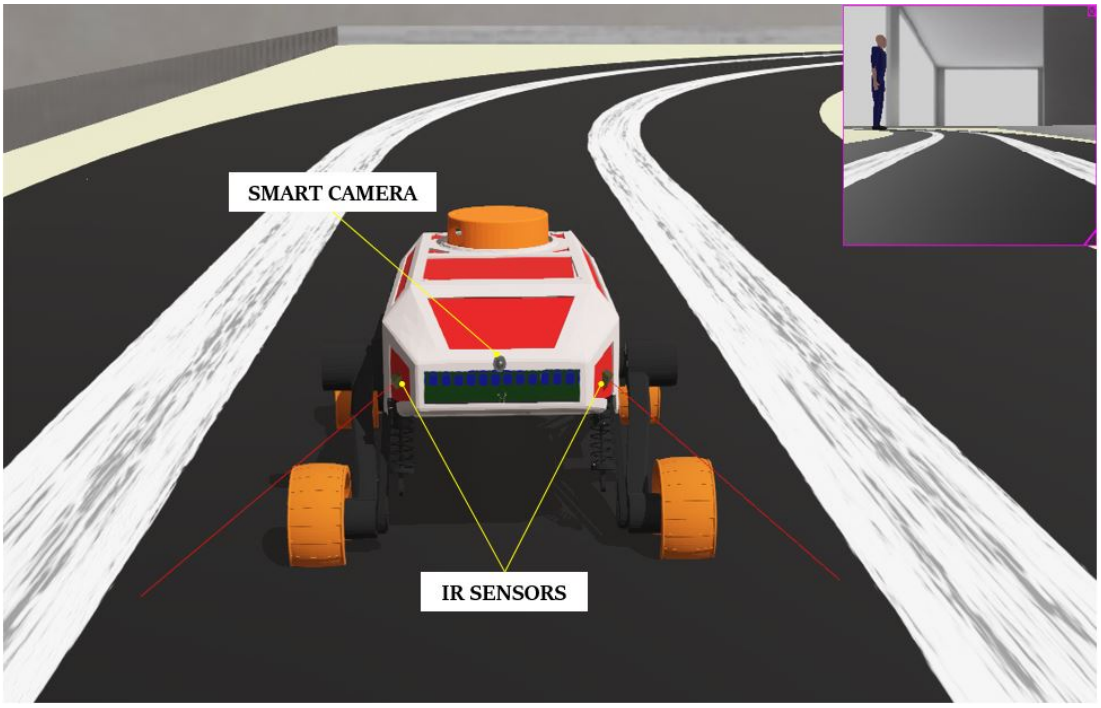

Figure 3.17: Vision sensors configuration

However, it must be emphasised that the Camera node itself provides the basic functionality of a general purpose camera, without any object recognition capability. Although real intelligent cameras have built-in components responsible for implementing image processing algorithms, Webots platform provides the user with a number of tools to simulate the intelligent functionalities of the visual sensor. In this regard, by adding a *Recognition* node in the respective field within the tree graph of the Camera node, the sensor becomes capable of recognizing objects that are captured in the image. In real practice, object identification is the outcome of the implementation of targeted algorithms based on artificial intelligence and machine and computer vision. However, for the sake of simplicity, in the Webots simulator, object recognition is made possible through a workaround: by initializing the *recognitionColor* field of the objects in the scene that have to be identified as obstacles, the smart camera will be able to detect them. In other words, this mechanism corresponds to marking the objects to be detected with an identifier recognizable to the eyes of the camera.

At this point, once the scene and the robot have been correctly setup, it is necessary to generate the controller code to activate the desired behaviour of the Bilby Rover. The program is composed of two macro parts. The first segment is responsible for initializing the sensors and actuators, which are instantiated

and enabled, allowing them to be controlled in the main code loop. The second part contains the implementation functions that are repeated recursively until the simulation is stopped. In this cycle, the controller receives output data from both the infrared sensors and the smart camera. On the one hand, the infrared sensor values are leveraged for controlling the motors to execute the lane keeping task; on the other hand, the controller continuously monitors the smart camera data readings and, as soon as an obstacle is detected, it issues a command to stop the robot at a safety distance until the obstacle is removed from the camera's field of view. Based on these considerations, the flowchart of Figure 3.14 can be integrated with the obstacle detection function, thus providing a complete representation of the behaviour triggered by the robot controller. The resulting modified flowchart is shown in Figure 3.18.

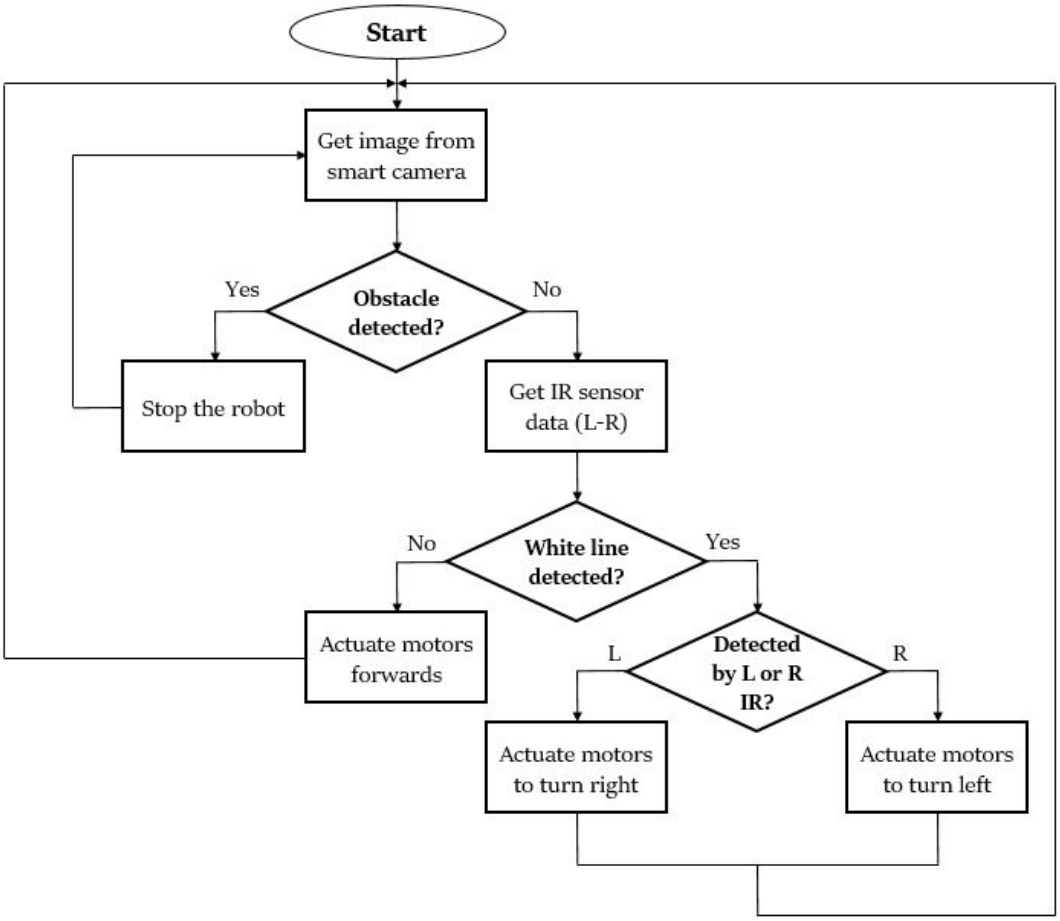

Figure 3.18: Flowchart of lane-keeping and obstacle detection algorithm

Through the implementation of the algorithm described in Figure 3.18, the Bilby Rover will move following the trajectory defined by the lane and stop at

a distance of 0.5 m from the identified obstacle, which in this case is a worker crossing the path. When the obstacle is removed from the smart camera field of view, the robot will resume moving along its intended path. The integrated approach of these vision sensors - infrared sensors and smart camera - allows the robot to move autonomously while ensuring the safety of its operations through the obstacle avoidance functionality.

3.3.4 Platooning control of a robot cluster

The present section describes the design and implementation of a simulation in which a cluster of two autonomous mobile robots are coordinated to move as a platoon.

Platooning is a coordination technique for groups of mobile modules, which seeks to ensure that each unit moves close to its preceding neighbour, thus forming a so-called platoon [58]. A platoon consists of a specific cluster in which the first robot is the leader, while the other robots are the followers: in such a configuration, each follower must follow the previous one to keep the formation [59]. Platooning has some interesting applications to autonomous vehicles in intelligent transportation systems: concurrent studies show that the deployment of platooning technology in vehicles and into the transportation infrastructure in general holds the promise of significant improvements to traffic safety and efficiency, allowing to lower traffic jams and air pollution in urban areas [60]. In this respect, truck platooning is the most prominent example of this technological application: it entails the connection of two or more trucks in convoy, using networking technology and automated driving support systems. These vehicles automatically maintain a close distance between each other when they are connected for certain parts of a route, for example on motorways [61]. Truck platooning has great potential in terms of creating safer, cleaner and more efficient road transport in the future, which is why the major truck manufacturers are already carrying out the first real-world tests to bring these platoons to European roads.

Platooning coordination of a group of two Bilby Rover robots is addressed in this section. In the considered system, the first robot is the *leader*, which moves at a predefined speed along an arbitrary path delimited by side lines, exploiting the algorithm for lane-keeping based on infrared sensor technology illustrated in section 3.3.3. The second robot, instead, is the *follower*, which, as soon as detects the presence of the leader, must precisely follow the route of the leader in front

of it and maintain the desired safety distance from that same vehicle. For this purpose, on Webots platform, the same factory environment illustrated in Section 3.3.3 has been replicated (see Figure 3.16): two Bilby Rover robots are therefore imported in the simulator world scene.

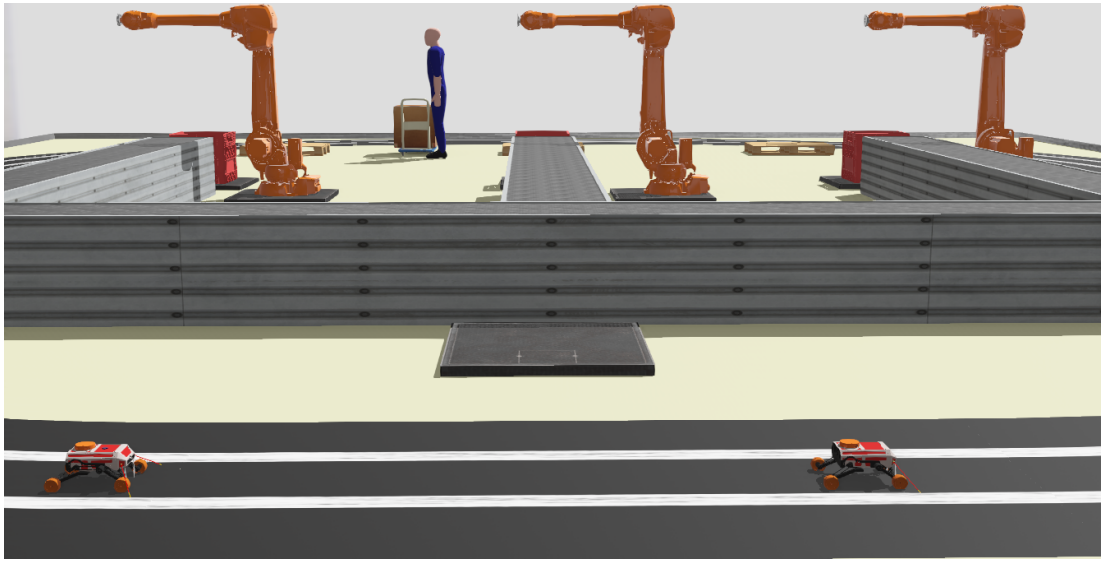

Figure 3.19: Webots scene with two Bilby Rover robots (leader and follower)

To make platooning technology effective, a set of devices must be installed on the robotic platforms, with the aim of enhancing perception capabilities of the robots. To clarify the necessary devices, it is essential to describe the operations of the robots and to understand which functionalities must be enabled to obtain the desired behaviour. Actually, among the requirements associated with the control of a platooning system, longitudinal and lateral control are the most relevant ones. Longitudinal control is aimed at stabilizing the distance between the leading vehicle and the following vehicle, while lateral control is concerned with aligning the follower's direction relative to the leader robot in front of it. This control consists in keeping the angle between the leader and the follower close to zero [62]. In the present simulation, the purpose is to enable the formation of the platooning cluster where the relative position information between the two modules is retrieved from a local sensing device. For this purpose, a radar sensor can be employed to fulfil this task.

Radar (short for *Radio Detection and Ranging*) is an active detection device that leverages electromagnetic waves in the radio or microwaves domain to detect, locating, tracking and recognizing objects of various kinds at even considerable distances [63]. Radar sensors typically involve radiating, in a pulsed or continuous

way, beams of electromagnetic energy towards a defined target in a region of interest via a transmitting antenna. When electromagnetic waves bump into the target, a portion of the radiated energy is reflected back towards the radar system. A receiver embedded in the output side of the antenna extracts the desired reflected signals, allowing measuring the location of the target in terms of distance and angular direction. Figure 3.20 shows a simplified functional diagram of the radar sensor, featuring the major elements involved in the process of transmitting and propagating the radiated beam and receiving the reflected signal back to the source. Although technical features may vary for a given radar, which can be simpler or more complex, the major subsystems include a transmitter, antenna, receiver, and signal processor.

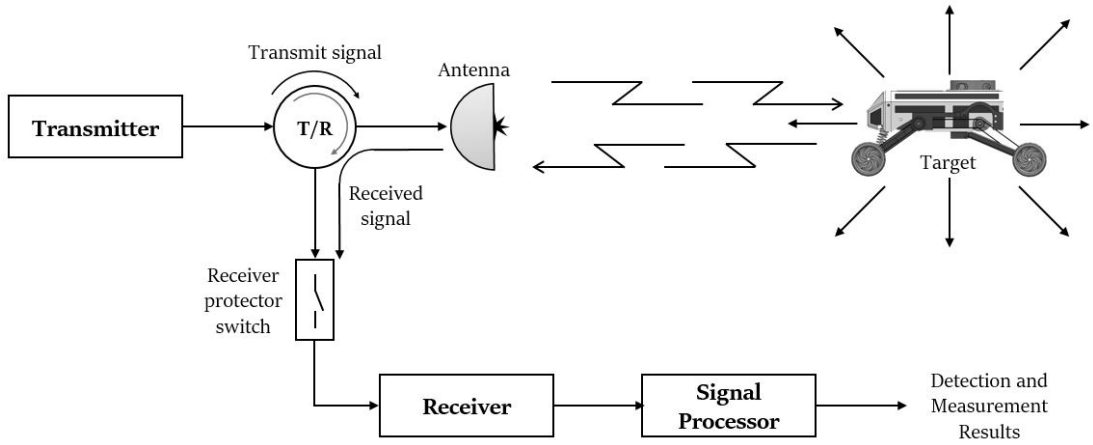

Figure 3.20: Simplified functional diagram of a radar (Source: adapted from [64])

EM waves, generated by the transmitter, are taken as input by the antenna and introduced into the propagation medium, typically the atmosphere. A transmit/receive (T/R) device – which is usually a circulator or a switch - is the responsible for providing a connection point so that the transmitter and the receiver can both be attached to the same antenna simultaneously, while ensuring a constant isolation between the transmitter and receiver side to protect sensitive components from the high-powered transmitted signal [64]. The radiated EM waves propagate through the environment to the target. After bumping off the target, a fraction of the radiated signal is reflected back to the source, captured by the antenna and applied to the receiver circuits. The signal is then processed to be finally sorted and analysed by the signal processor, which is capable of extracting valuable information about the detected object, such as the distance, or range, and the angular direction. Distance is retrieved by computing the total time it takes for the EM waves to make the round trip to target and

back at the speed of light (Time of Flight computation); instead, the angular direction of a target is found from the direction in which the antenna is pointing at the moment in which the reflected signal is received. Moreover, through consecutive measurements of the target location at successive instants of time, radar technology makes it possible to estimate the trajectory of the detected object.

According to the above considerations, a *Radar* node is introduced in the tree diagram of the Bilby Rover follower on Webots simulator. The detection sensor is thus configured by modifying fields of interest to replicate the desired technical specifications of a real device, such as the minimum and maximum range, vertical and horizontal field of view, as well as the resolution. The radar sensor thereby installed on the Bilby Rover follower (see Figure 3.21), after having identified the platooning leader during the simulation, continuously feeds the control system with relevant information, such as the distance D and the azimuth θ_a (orientation relative to the follower) of the robot in front of it [62]. This information is indeed used as input data for the control algorithm that activates the trajectory tracking behaviour, forcing the second robot in the cluster to follow the path of the robot in front of it.

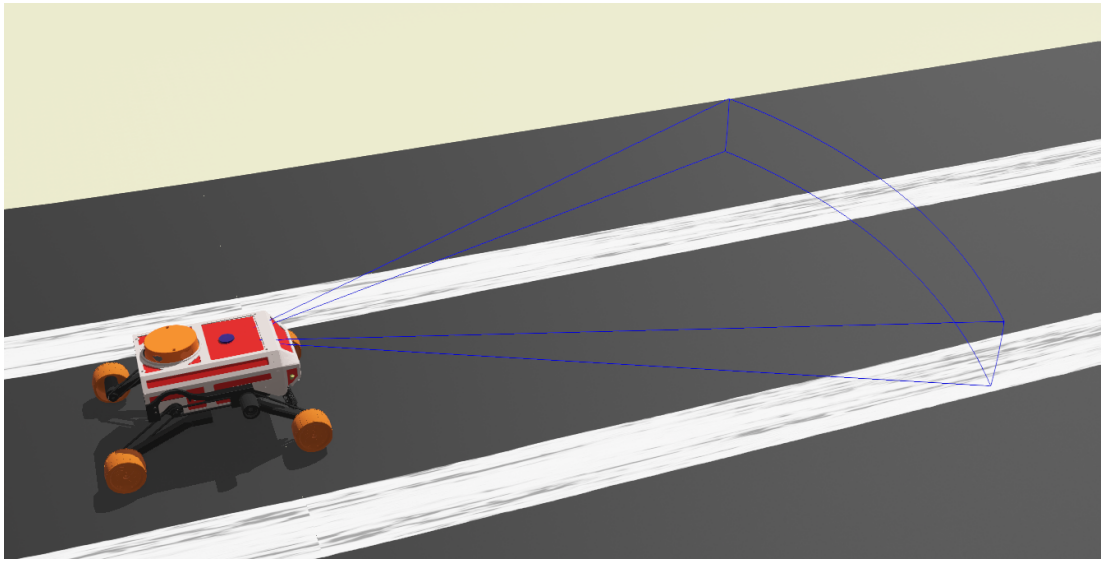

Figure 3.21: Radar installation on the follower (frustum defined by blue lines)

In addition to the radar sensor, the two Bilby Rovers forming the platooning cluster require the integration of appropriate devices to establish a communication link between the two modules. Actually, while on one hand the radar supplies information regarding the position, on the other hand it is also required to continuously track the speed of the leading robot, so as to adjust the speed profile of

the entire formation accordingly, thus avoiding possible collision situations and ensuring a constant distance between the modules. The general communication scheme requires the leader robot to broadcast at any discrete time slot speed information, which is then collected by the follower robot and used as input data in the control algorithm, allowing the speed to be tuned to match the one set by the leader.

In this simulation, communication between the two modules is accomplished through the deployment of emitters and receivers on the robot platforms: in this case, the leader robot continuously broadcasts information through the emitter, while the follower harnesses the receiver to capture the transmitted speed data, thus establishing a unidirectional communication scheme. Therefore, on Webots simulator, emitter and receiver nodes are introduced in the tree graph of the leader and follower Bilby Rover, respectively. Figure 3.22 provides an illustration of the arrangement of the platooning system, highlighting the formation of the two robots and the technologies involved.

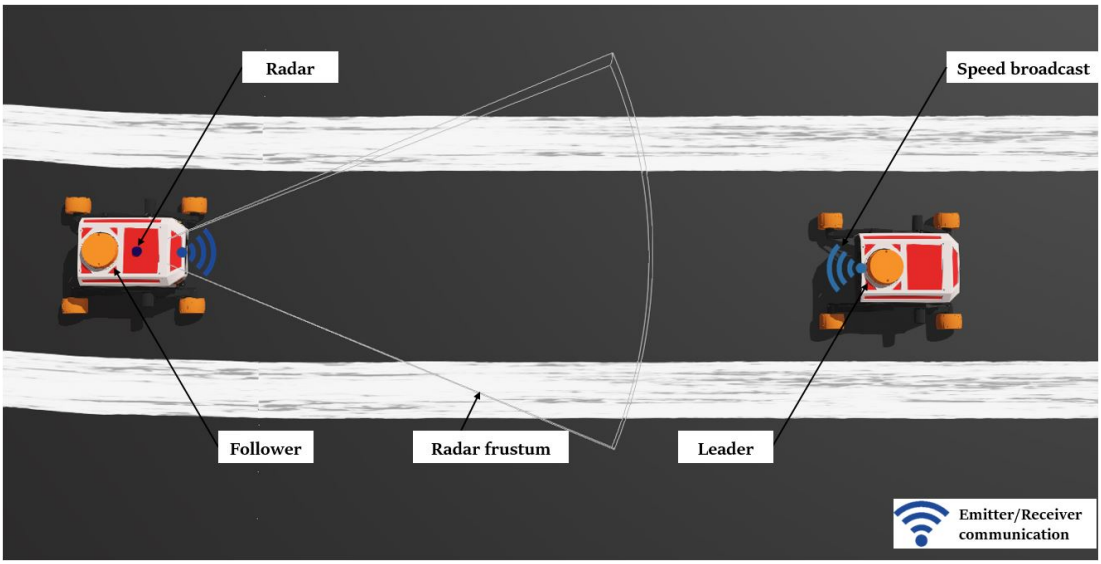

Figure 3.22: Platooning formation

After the scene is suitably setup, it is essential to generate the controller program to enable the desired performance.

As far as the leader robot is concerned, its controller code is aimed at implementing the lane-following behaviour that has already been described in Section 3.3.3. In addition, while moving independently along the path, the leader is asked to broadcast its speed at every time slot via an emitter, which has to be properly

initialised together with the other devices installed on the robot.

With regards to the robot follower, the source code, as usual, includes a first part devoted to the initialization of the sensors and actuators installed on the Bilby Rover platform. In this section of the code, the different devices are instantiated and labelled using a specific tag that allows them to be called up in the algorithm, thereby providing access to their specific functions. Following the preliminary initialization part, it is necessary to actuate the devices by routing application-specific commands generated through the processing of data from the sensors. The developed algorithm aims at the generation of the following mechanism. The robot repeatedly monitors data provided by the radar to investigate the presence of a robot in front of it: this is done by continuously polling the sensor with a dedicated function that returns the number of identified targets. If no target is revealed, the robot is controlled to move following the path by exploiting the lane-keeping algorithm discussed in Section 3.3.3.

As soon as the target is identified by the radar - in this case the leader of the platooning group - the distance between the two robots starts to be measured at each time slot: it must be stressed that one of the goals of platooning is actually to ensure a distance that is as stable as possible. As the follower robot initially moves at a faster speed than the leader, the distance from the front robot is progressively reduced. Hence, when the range between the two modules is recorded below a pre-determined threshold, the follower robot retrieves the speed information of the leader via the receiver and, consequently, it issues commands to the motors to adjust the speed profile accordingly.

Upon the establishment of this initial connection, the data provided by the radar are again exploited to trigger trajectory tracking behaviour. Actually, the radar constantly provides the control system with information pertaining to the azimuth of the leader, namely its angular direction measured relative to the follower. The azimuth value is therefore kept under close observation to verify the alignment of the two platforms and it is used as input to the control algorithm according to the following logic. If the azimuth is lower than a certain threshold ($0.1\ rad$), it is assumed that the two robots are reasonably aligned; therefore, the motors are actuated to move the follower robot forwards without applying any correction to its trajectory. By contrast, if the azimuth exceeds the said threshold, it is considered that the leader and follower robots are no longer aligned as they should be. This consequently triggers an immediate response from the

control system, which transmits commands to the motor to implement a steering mechanism capable of restoring the proper alignment between the two robots. Owing to this reiterative cycle of sensors interrogation and devices actuation, the robot in the platoon behaves as a reactive system with trajectory tracking capabilities without relying on the use of infrared sensors for path recognition. The flowchart in Figure 3.23 provides a schematic block illustration of the main control loop of the robot.

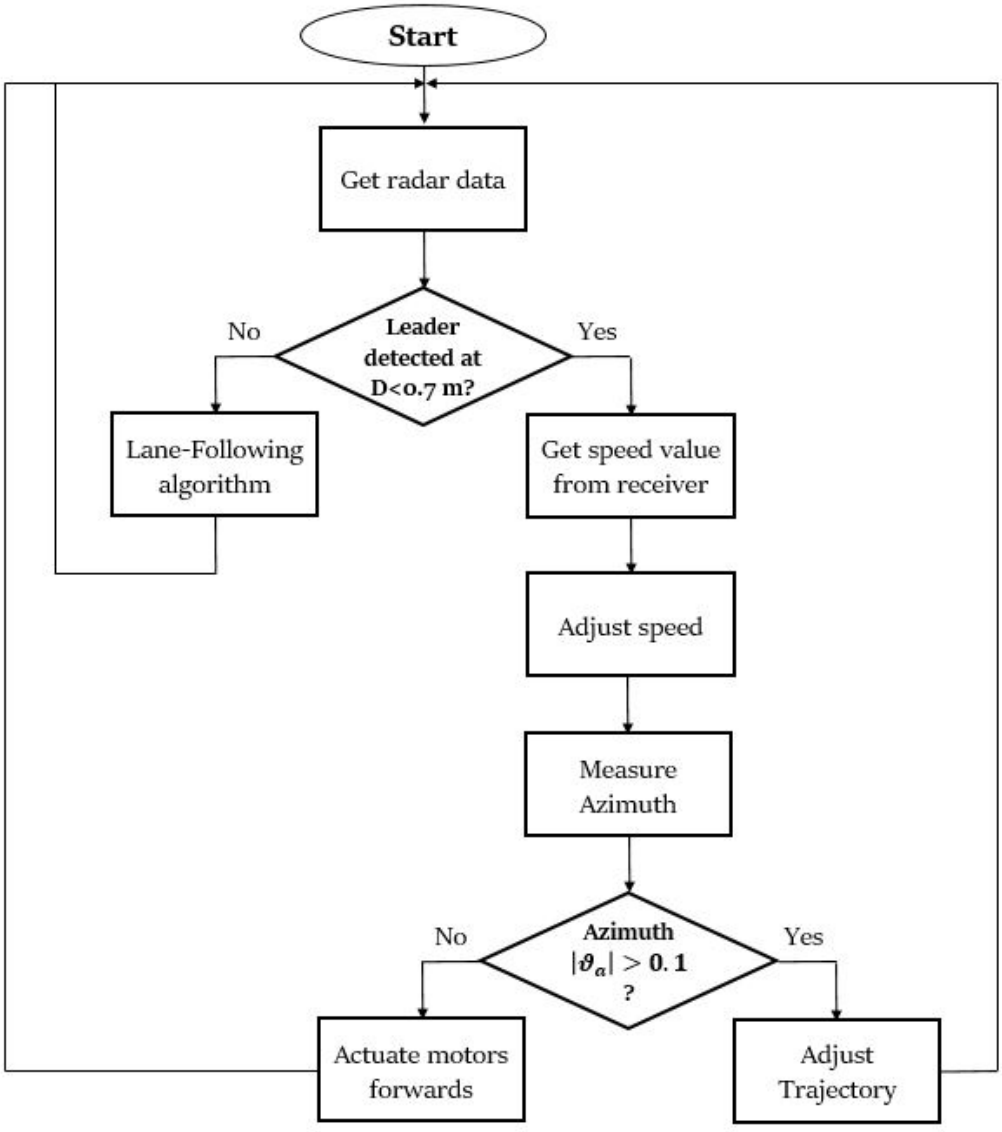

Figure 3.23: Block diagram of the platooning control algorithm

As a final consideration, it is worth pointing out that the proposed control program is applicable to both the follower and the leader, provided that both platforms are equipped with the same devices. Actually, as it is possible to see also

in Figure 3.23, the controller code, depending on the data received as output from the radar sensor, can enable two different behaviours: if the target is identified, the controller sends the necessary commands to establish the platooning cluster as described above; instead, if no target is identified, the controller issues commands to execute the lane-following mechanism by using the infrared sensors. For this reason, the same controller can be used for the robot leader; furthermore, this makes it possible for the leader to become on its turn a follower if a new robot is identified in front of it.

This solution thus ensures that the same control code can be applied to all robots, enhancing modularity and flexibility of the platooning formation. However, this also requires each robot (not only the leader module) to broadcast its speed via an emitter, so that every system can be the potential leader for the next robot.

3.3.5 Obstacle avoidance control

The purpose of this simulation is to develop an algorithm for controlling the Bilby Rover to achieve efficient and smooth navigation in an unknown environment, based on the idea of Braitenberg vehicles.

In conditions that are dangerous for humans and their environment, the use of robots can be a solution to overcome these problems and avoid potential hazards for the workers. In this framework, mobile robots, equipped with different types of sensors, can move around and investigate in open areas, rooms, and even industrial fields. Various devices can be employed to determine the obstacle free path and the exact position of the robot [65]. In order to navigate safely in an unknown environment, a mobile robot needs to deal with the uncertainty and imprecise, or incomplete, information about the environment in a timely manner. This results in real-time demands on the navigation system [66]. When it comes to the definition of the robot navigation problem, it can be generally decomposed into two sub-tasks: *goal-seeking* and *collision-avoidance*. For this purpose, Braitenberg vehicle strategy can be used to navigate the movements of the robot, thus enabling the desired behaviour to carry out the intended task.

Braitenberg vehicles have been used in robotics for decades on an empirical basis and were conceived in the 1980s by the Italian-Austrian cyberneticist Valentino Braitenberg, who conducted a series of thought experiments concerning the design of a set of vehicles. By means of elementary couplings, the sensors of these

systems directly influence the actuators, through inhibition or excitation, giving rise to even very complex global behaviours. For instance, by coupling sensors in various ways to the motors of a differential drive vehicle, a wide range of behaviours can be replicated, which, from the observer's point of view, can be interpreted as fear, aggression, love, and other behaviours in relation to the sources.

In practice, Braitenberg vehicles attempt to qualitatively replicate sensor-based animal movement and have been widely used in fields other than robotics, such as Artificial Intelligence, Neural Networks and Swarm Optimization. In the most basic applications, what is modelled is the movement of animals to or from a defined stimulus. The goal is to implement a robotic movement to replicate this behaviour, known in biology as positive or negative taxis: while positive taxis is a goal-seeking technique, negative taxis concerns avoidance behaviour [67]. Several Braitenberg vehicles have been deployed to provide robots with these capabilities on an experimental basis. Figure 3.24 provides a schematic illustration of these systems, named respectively Vehicle *2a, 2b, 3a*, and *3b*.

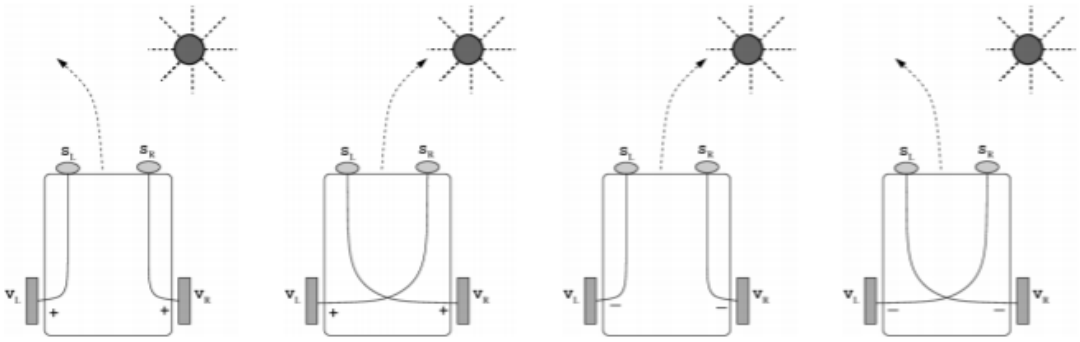

Figure 3.24: Braitenberg Vehicles 2a, 2b, 3a, 3b (Source: [67])

Both Vehicles 2 and 3 are equipped with two sensors and two motors, but with different configuration schemes. For Vehicles 2a and 3a, each sensor is connected to the motor on the same side, which is referred to as direct-connection; instead, for Vehicles 2b and 3b, each sensor is connected to the motor on the opposite side, which is defined cross-connection. Moreover, plus (+) and minus (-) signs can be observed in Figure 3.24: + means that the higher sensor readings, the faster the motors rotate; by contrast, - means that the higher the readings, the slower the motors rotate [66].

Considering the configurations of Vehicles 2a and 2b, if the system approaches a source from one side, the sensor closest to the source will be excited more than

the other; as a result, the motor connected to this sensor will be activated to rotate faster. Owing to the different connection patterns, this will trigger two different behaviours: actually, Vehicle 2a runs away, departing from the source, while Vehicle 2b moves towards it, heading to the source. Considering the configuration of Vehicles 3, while approaching the source, Vehicle 3a moves towards it, while Vehicle 3b drifts away from it [66]. The difference in behaviour between vehicles 2 and 3 is related to the negative connections between the sensors and the corresponding motors in 3a and 3b: indeed, in this case, the higher the sensor readings, the lower the speed of the corresponding motors. This justifies the different response to the presence of the source.

According to the described configurations and the derived behaviours, it is reasonable to assume that Vehicles 2b and 3a are suitable for the purpose of *goal-seeking*. In fact, Vehicle 2b drives faster towards the source once it is detected, as if attracted by it; Vehicle 3a, instead, approaches the source at a slower speed. In contrast, Vehicles 2a and 3b can be identified as suitable for *obstacle avoidance*. However, Vehicle 3b is preferable for the purpose of collision avoidance, as it represents a safer solution: as a matter of fact, it is safer for a mobile robot to slow down and avoid obstacles rather than accelerating towards them and then performing the avoidance [66].

Based on the above analysis, the aim is to develop a navigation scheme for the Bilby Rover, allowing it to move in an unknown environment and avoid obstacles by replicating the behaviour of the Braitenberg 3b Vehicle. For this purpose, the tuning scheme is based on sensor readings from the Lidar sensor installed on the robot. The values that will be used as references in this system are the angle coordinates and the object distance obtained from the sensing device. However, due to the nature of the stimulus and sensing hardware, there must be a necessary sensor data processing between sensing and motor actuation: for this reason, Bilby Rover will not be a Braitenberg 3b vehicle in the strict sense. Therefore, the Braitenberg Vehicle model can be exploited to design the robotic controller at the steering level [67].

On the Webots simulation platform, a factory environment is created, where several objects are located, including the Bilby Rover, as it is possible to see in Figure 3.25.

Figure 3.25: Webots scene for collision avoidance simulation

At this point, it is required to configure the Lidar node, which has been already introduced in the Robot tree graph upon the creation of the Bilby Rover model in Section 3.2. For this purpose, the ranging sensor is required to scan the area in front of the robot, covering a 180° horizontal field of view. Other technical parameters are initialized to replicate the characteristic of a real device, namely the minimum and maximum range, the number of emitted beams, and the sensor resolution. Chapter 2 already illustrates the basic principles underlying LiDAR technology; therefore, they will not be proposed again in this section.

Figure 3.26 depicts the Bilby Rover robot equipped with the Lidar, where blue rays represent the beams – forming the so-called Point Cloud - emitted by the ranging sensor to retrieve distance information during the simulation.

After the complete set-up of the world scene and the Bilby Rover, it is necessary to program the controller code, thus triggering the behaviour to replicate a Braitenberg 3b Vehicle. After the initialization of actuators and sensors in the first part, the main loop develops the control algorithm that receives range measurements as input and, in turn, after processing range information, outputs the necessary commands to perform obstacle avoidance. Figure 3.27 depicts the overall block diagram of the obstacle avoidance system.

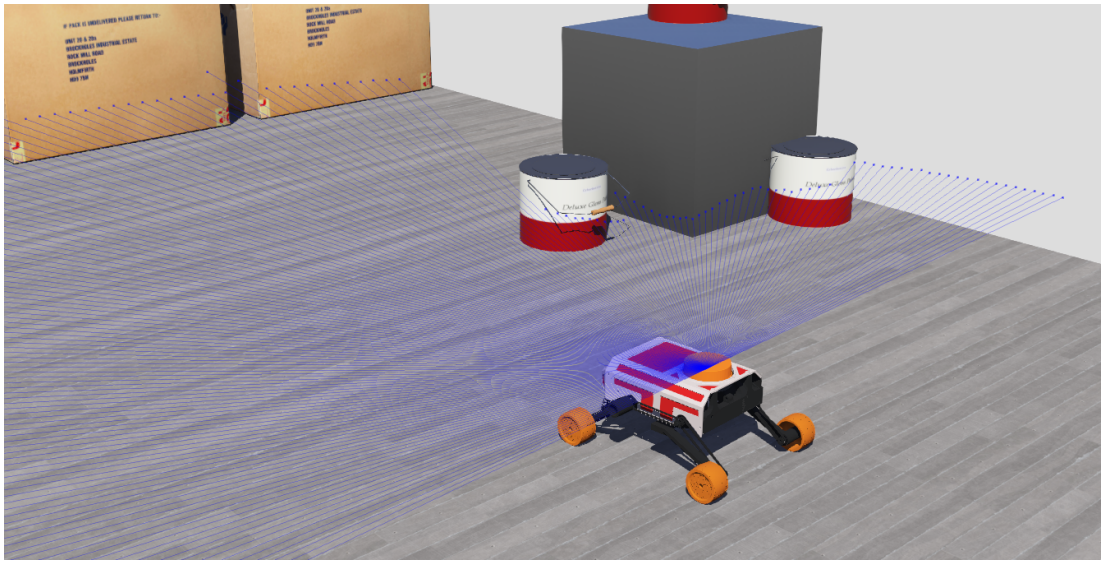

Figure 3.26: Environment scanning with Lidar sensor

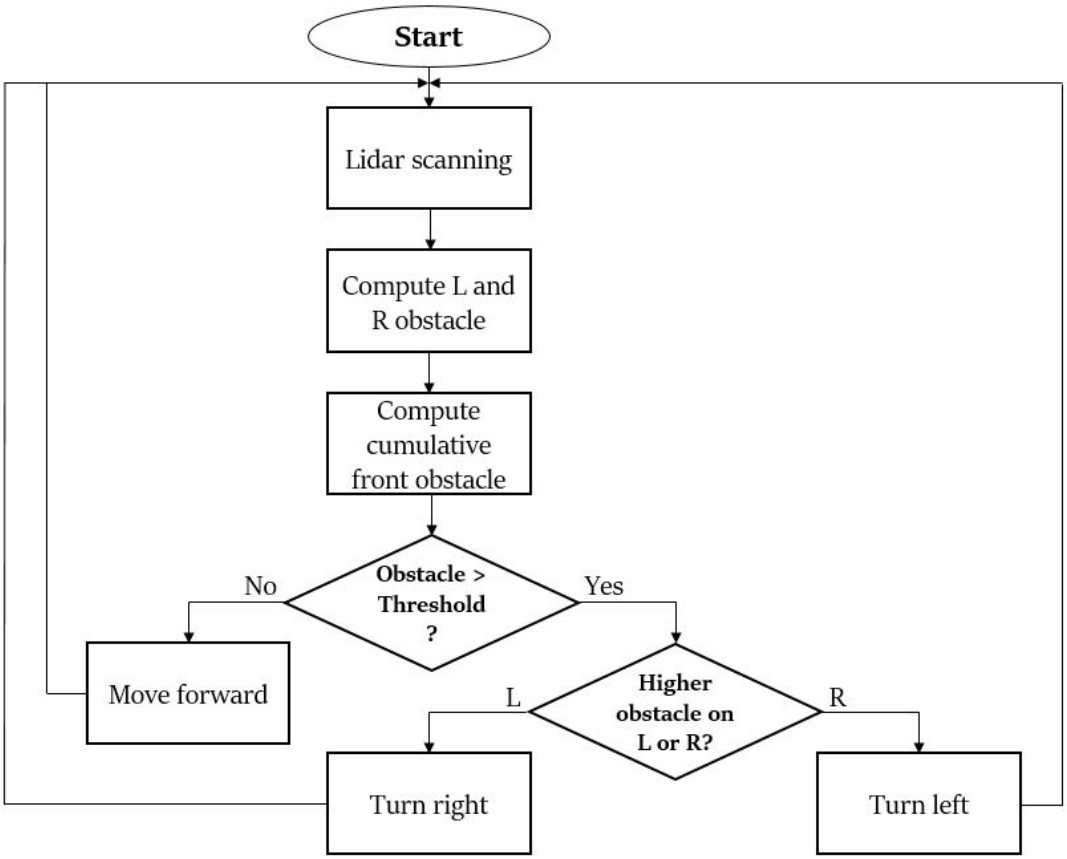

Figure 3.27: Flowchart describing collision-avoidance algorithm

The control system repeatedly polls the Lidar to retrieve the array containing distance readings from the sensor which scans the environment in front of the Bilby

Rover. At each time step, the controller monitors the distance to the obstacles and, by assigning specific weights to measured range values, calculates the cumulative obstacle for both the left and the right side of the robot: in these calculations, obstacles farther than a certain threshold ($d > 1m$) are neglected. Based on the cumulative obstacles calculated on the left and right side, the controller code calculates the speed values to be assigned to the motors, in order to move the Bilby Rover to avoid any collision. So, for instance, in the presence of an obstacle detected on the right side of the robot, the controller reads the distance values from the lidar and, after processing them suitably, generates a series of commands to trigger the steering mechanism aimed at avoiding the obstacle. Therefore, right-hand motors are made to rotate at a certain speed, while the left-hand motor is forced to rotate more slowly or even in the opposite direction. This response, whereby a higher excitation of the right-hand sensor corresponds to a lower speed of the left-hand motor, is consistent with Braitenberg 3b Vehicle model. As a result, the robotic platform will be forced to turn to the left, thus deviating from its trajectory and avoiding collision with the detected obstacle. The opposite mechanism applies if the obstacle is identified on the left side. It is also worth noting that the speed value to be assigned to the motors is calculated as a function of the sensor readings: different values of the cumulative obstacle will produce different system responses, hence different speeds to be assigned to the motors.

The results of the present simulation show that the Bilby Rover manages to navigate the room without hitting or crashing into the walls or obstacles, successfully implementing a Braitenberg-like behaviour.

3.3.6 Wall following

The purpose of this section is to present the steps to develop and simulate a basic wall-following control of the mobile robot Bilby Rover, which is intended to traverse a maze-like environment and stop once it reaches its destination. Therefore, the robot's objective is twofold: on the one hand, it must be able to navigate the environment by following the walls, and on the other hand, it must be equipped with perception capabilities to sense the arrival at the designated destination. These behaviours are empowered by the deployment of specific sensors on the robot's body, namely ultrasonic and radar sensors. While the ultrasonic sensors are responsible for supplying the necessary tools to ensure the stability of the wall-following behaviour, the radar sensor - whose basic principles are outlined in Section 3.3.4 - will have to inspect the surrounding environment during navigation,

thus recognizing the presence of the target destination.

Wall, corridor and path following are essential behaviours for robot in many workspaces, requiring viable obstacle avoidance techniques [68]. A wall following robot has the primary task to follow the wall by maintaining its movement. In many robots, ultrasonic sensors are used for wall and corridor following.

A sonar or ultrasonic sensor uses the propagation of acoustic energy at higher frequencies than normal hearing to extract information from the environment [69]. This device typically transmits a short pulse of ultrasonic waves towards a target, which reflects the sound back to the sensor. Similarly to other ranging devices, the system then measures the time for the echo to return to the sensor and computes the distance to the target using the speed of sound in the medium, as follows:

$$d = \frac{ToF \cdot v_c}{2} \tag{3.1}$$

Where d is the distance to the target, ToF (Time of Flight) defines the time elapsed between the emission of the sound waves and the reception of the echo at the source, and v_c is the speed of sound in the medium. Figure 3.28 provides a simplified scheme of the ultrasonic sensor functional diagram, showing its basic features.

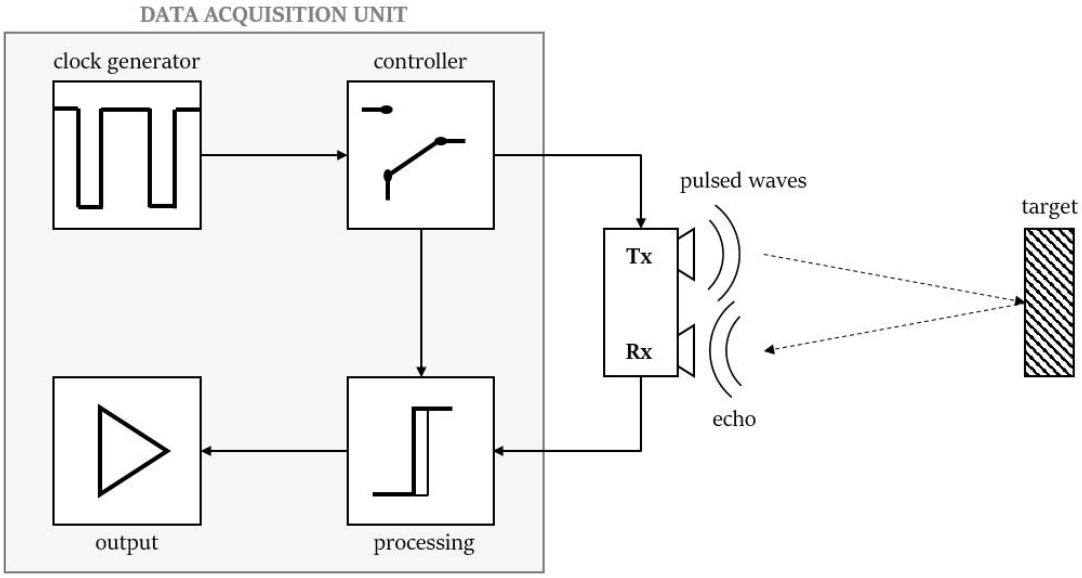

Figure 3.28: Ultrasonic sensor functional diagram (Source: adapted from [71])

A typical ultrasonic sensor comprises a clock (signal) generator, a controller

to excite a transducer to transmit sound waves and receive the echo, a processor and an output amplifier to handle the return signal [70], which make up the Data Acquisition Unit. During the time period from transmitting sound waves to receiving the echo, the ultrasonic sensor gives a high-level signal to the DAU, which leverages a timer to compute the interval in which the signal remains high [71]. This data is used as input for the ToF to compute the distance as showed in Equation (3.1).

Thus, leveraging the mounted sensors, the robot is capable of sensing the distance between the wall and itself along the operation. Besides, these distances are processed to generate the proper movement with the involvement of a specific controller [73].

To test the wall following ability of the robot, a maze-like environment is designed on the Webots simulator through a suitable arrangement of walls. In addition, a cardboard box is included in the world scene to pinpoint the destination position of the robot. This choice is realistic, as in industrial settings robots are often instructed to navigate autonomously along a defined route to reach a material loading or unloading station, thereby using the robots themselves as a material handling equipment.

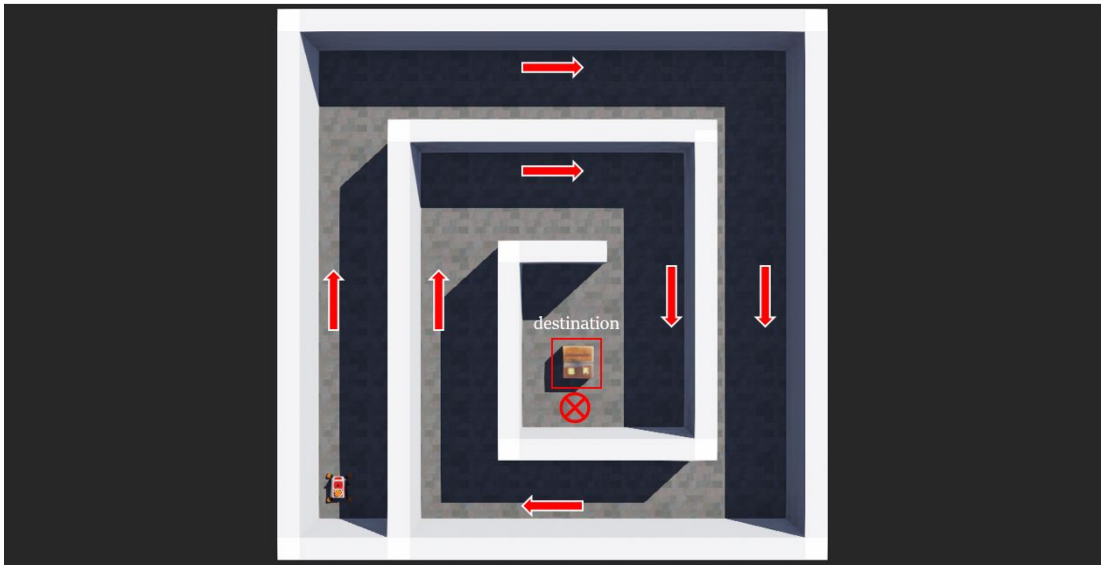

Figure 3.29: Webots scene for wall-following algorithm

Once imported in the scene, the Bilby Rover model is completed with the installation of the radar, in charge of detecting the target destination, and two

ultrasonic sensors: the latter are placed on the precise direction that allows the robot to sense the presence of left-side and front-side walls accurately. Figure 3.30 illustrates the configuration of the robot, where red rays represent the direction of sound waves emitted by the front and side sonar sensors, while the blue lines define the frustum of the radar installed on the top panel. It is worth noting that the radar is purposely oriented to scan the environment on the right side of the robot, because, considering the navigation scheme, the Bilby Rover approaches the destination leaving the carton box on its right.

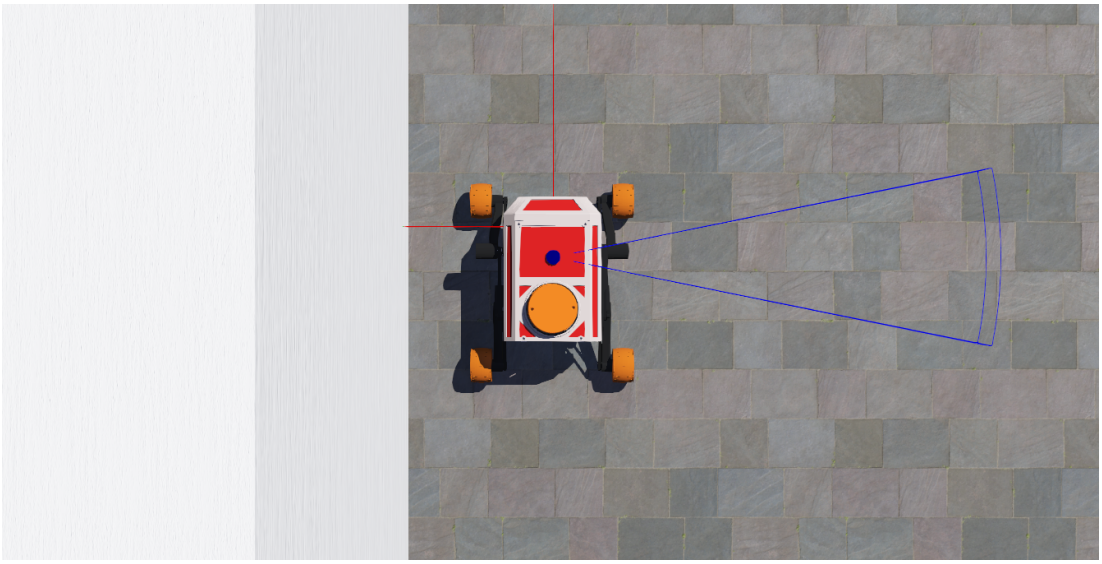

Figure 3.30: Sensors configuration for wall-following and target identification

At this point, it is necessary to generate the controller code to enable the desired wall following behaviour, getting finally to the destination. For this purpose, all sensors and actuators are initialized and instantiated in the first part of the controller code, labelling them with identification tags to call them in the main loop. Once initialization is completed, it is necessary to set up the control algorithm, leveraging sensor data.

There are two main abstractions in the wall following algorithm: the sensors and the controller. Sensors are the means that allow pulling data from the environment, interpreting the wall model, and estimating the robot position relative to the walls. The controller, based on the wall model, issues command to ensure a smooth navigation of the system through the maze. Supposed that the robot moves by following a wall detected on the left, based on ultrasonic sensor readings, it is possible to highlight four different situations, which trigger in turn four

alternative responses of the system. Figure 3.31 provides a schematic summary of the four cases, with the relative behaviours triggered by the control system.

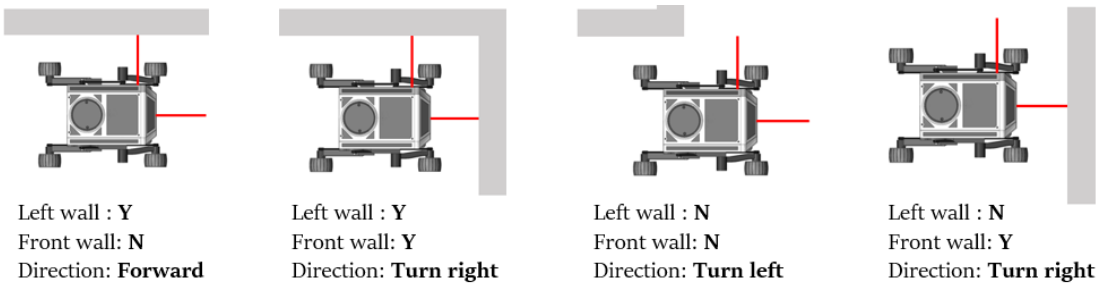

Figure 3.31: Schematic summary of wall configurations and relative responses

According to these considerations, the implementation of the wall-following algorithm is quite straightforward: at each time step, the controller processes the output data from the ultrasonic sensors and, based on their readings, matches the robot's position to one of the four schematic situations outlined in Figure 3.31. Then, based on the identified wall configuration, it sends commands to the motors to activate the desired response accordingly.

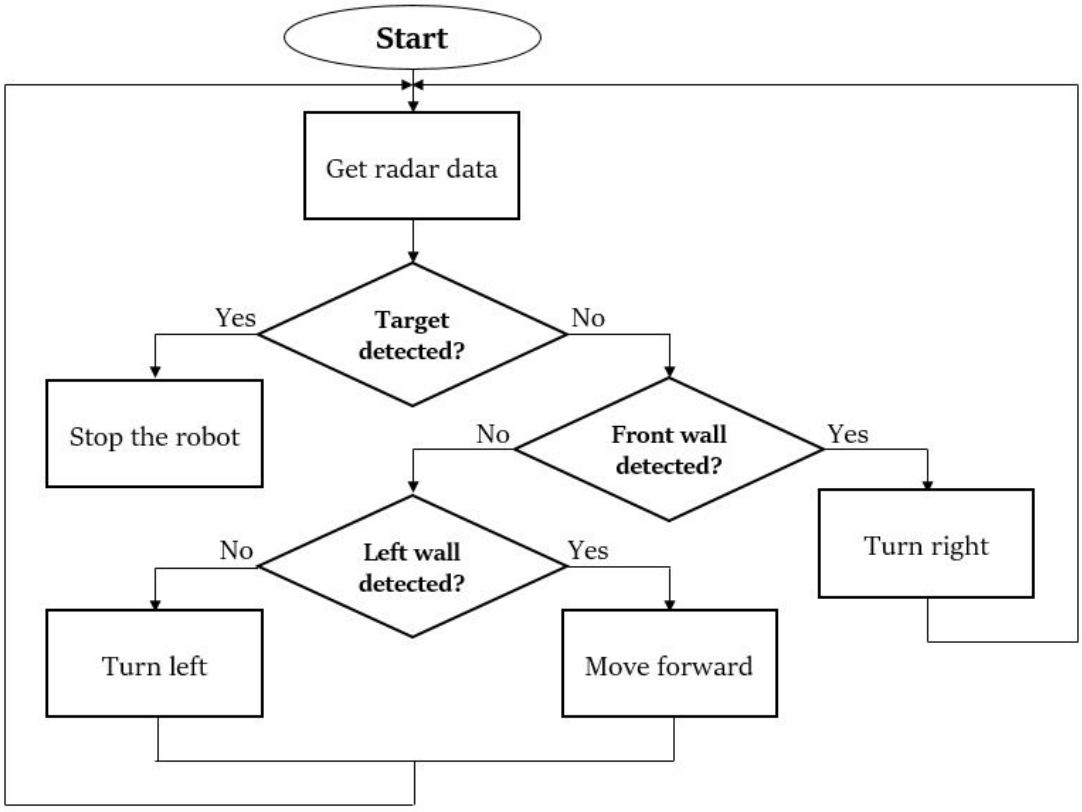

Figure 3.32: Flowchart describing the robot control loop

To integrate the robot performance with the capability of recognizing the established destination, at each cycle the controller monitors the output data coming from the radar. If the radar does not identify any target, the control system drives the motors to implement the wall following function; conversely, if the target is detected, the controller issues the orders to prevent the motors from rotating, thus stopping the robot in front of the designated destination.

For the sake of clarity, the flowchart depicted in Figure 3.32 provides a schematic of the main control loop underlying robot behaviour in the present simulation.

Chapter 4

Hector-SLAM implementation

The capability to acquire a general model of the surrounding environment and to position itself in it is a key ability of truly autonomous robots, which must be able to operate in real world settings [72]. The ability to perceive the environment and to locate in it are crucial for robots to carry out their assigned tasks. Environmental maps are required in a wide range of applications including autonomous navigation, obstacle avoidance, urban search and rescue (USAR), and surveillance tasks, both in outdoor and indoor areas. Mapping knowledge of a room or of a building is extremely important for robots in the execution of their duties as well as for humans who might need to access a certain area that would be otherwise difficult to reach. By mapping a room, information about the environment is thereby generated [73].

To tackle the requirements of accurate maps for robot navigation, Simultaneous Localization and Mapping (SLAM) has emerged as the priority mapping method, becoming a key technology in the field of robotics, automation and computer vision [73]. Research in SLAM and in the navigation of unknown areas has been attracting considerable interest in recent years, proving impressive results and several suggested implementations harnessing the potential of a variety of sensory devices. Many algorithms have been developed and tested to address the need to find a solution to the SLAM problem. In this respect, Chapter 2 proposes a short summary of the state of the art of SLAM technology, highlighting the most widely used techniques in the industrial field.

The objective of this chapter is to test the suitability of the Hector-SLAM algorithm in view of its application on the Bilby Rover robot, which, by exploiting the LiDAR sensor, would be expected to provide a 2D map while moving within an

indoor environment. In this framework, the YDLIDAR X4 sensor is used as an indoor scanner. Raspberry Pi 4 B single board computer (SBC) is used to access the LiDAR data and then send it to a computer wirelessly to process it into a map. This computer and the SBC are integrated into a robot operating system (ROS). The Hector-SLAM algorithm determines the position of the robot based on the scan match of the LiDAR data. Then, the LiDAR data will be exploited to determine the obstacles encountered by the robot, which will then be represented in occupancy grid maps.

The chapter is structured as follows. The first section is devoted to the mathematical discussion of the Hector algorithm developed by the Hector Team of the Technical University of Darmastadt for the solution of the SLAM problem. The second section is dedicated to the description of Robot Operating System (ROS), an open-source middleware that is become a *de-facto* standard for the development of robotic software modules: its features and functionalities will be briefly presented. In the third section, the methodology followed in the implementation of the Hector algorithm will be illustrated: the configuration of the architecture employed together with the ROS-based libraries used, will be covered in detail, providing an overview of the systems and tools needed for running the simulations. In the last section, finally, the results of the tests performed in the chapter will be proposed, with considerations on the suitability of the algorithm in the perspective of its integration on the Bilby Rover robot.

4.1 Review of the Hector algorithm

Hector algorithm is an open source implementation of the 2D SLAM technique proposed by Kohlbrecher, von Stryk, Meyer and Klingauf at the Technical University of Darmstadt [72]. The Hector SLAM algorithm is designed to enable accurate environmental perception and self-localization while preserving low computational burdens. It can be applied for SLAM in small-scale settings where large loops do not have to be necessarily closed and where leveraging the high sampling frequency of modern 2D LiDAR systems is advantageous [72]. Compared to the majority of SLAM grid-map techniques, the Hector algorithm does not rely on accurate odometry information, and hence it is particularly suited for robotic platforms with restricted processing capabilities [74]. The Hector algorithm serves as a SLAM front-end, aiming to provide online robot motion estimation in real time, without performing any back-end pose graph optimization, as described in Chapter

2. Although the system does not provide explicit loop closure detection, it is reasonably accurate for many real-world scenarios. Actually, by taking advantage of the low distance measurement noise and high scanning rate of modern LiDAR sensors, the reliability and precision can be much higher than that offered by odometry data, if available at all.

The Hector SLAM technique relies on the use of laser scanning to construct a grid map of the surroundings, exploiting the process of scan matching, which seeks to align laser scans with each other or with an already existing map. In this framework, the alignment problem is viewed as an optimization problem and it is thereby tackled by employing the Gauss-Newton approach: drawing inspiration from the relevant work in computer vision, the basic idea in the scan-to-map matching algorithm is to use the Gauss-Newton approach to find a local optimal solution to align the LiDAR scans with the map learned so far [75]. Hence, the approach is concerned with finding the optimal alignment of the laser beam endpoints with the generated map by iterating the Gauss-Newton algorithm to find the rigid transformation $\xi = (p_x, p_y, \psi)^T$ that minimizes

$$\xi^* = \arg\min_{\xi} \sum_{i=1}^{n} [1 - M(S_i(\xi))]^2 \tag{4.1}$$

That is, the algorithm has the objective to find the transformation that maximizes the probability at the scan points in the map. In Equation (4.1), $\xi = (p_x, p_y, \psi)^T$ indicates the position and orientation of the mobile carrier in the global coordinate system, n indicates the number of scan points, and $S_i(\xi)$ is a function of ξ that converts a LiDAR scan point $s_i(s_{i,x}, s_{i,y})^T$ from the local to the global coordinate system, according to Equation (4.2):

$$S_i(\xi) = \begin{pmatrix} cos\psi & -sin\psi \\ sin\psi & cos\psi \end{pmatrix} \begin{pmatrix} s_{i,x} \\ s_{i,y} \end{pmatrix} + \begin{pmatrix} p_x \\ p_y \end{pmatrix} \tag{4.2}$$

Finally the function $M(S_i(\xi)$ will return the occupancy probability at the coordinates given by $S_i(\xi)$.

The iterative Gauss-Newton method is a gradient descend algorithm. However, since the discrete nature of occupancy grid maps does not allow for the direct computation of interpolated values or derivatives, the method of bilinear interpolation is exploited to calculate the occupancy probability and gradient [75]. Intuitively, the grid map cell values can be viewed as samples of an underlying continuous

probability distribution. Given a continuous map coordinate P_m, the occupancy value $M(P_m)$ together with the gradient $\nabla M(P_m) = (\frac{\delta M}{\delta x}(P_m), \frac{\delta M}{\delta y}(P_m))$ can be approximated by using the four nearest integer coordinates around it, as showed in Figure 4.1. Linear interpolation along x and y axis yields to the formulation proposed in Equation (4.3):

$$\begin{aligned} M(P_m) \approx & \frac{y - y_0}{y_1 - y_0}\left(\frac{x - x_0}{x_1 - x_0}M(P_{11}) + \frac{x_1 - x}{x_1 - x_0}M(P_{01})\right) \\ & + \frac{y_1 - y}{y_1 - y_0}\left(\frac{x - x_0}{x_1 - x_0}M(P_{10}) + \frac{x_1 - x}{x_1 - x_0}M(P_{00})\right) \end{aligned} \tag{4.3}$$

From this, derivates can be approximated as:

$$\frac{\delta M}{\delta x}(P_m) \approx \frac{y - y_0}{y_1 - y_0}(M(P_{11}) - M(P_{01})) + \frac{y_1 - y}{y_1 - y_0}(M(P_{10}) - M(P_{00})) \tag{4.4}$$

$$\frac{\delta M}{\delta y}(P_m) \approx \frac{x - x_0}{x_1 - x_0}(M(P_{11}) - M(P_{10})) + \frac{x_1 - x}{x_1 - x_0}(M(P_{01}) - M(P_{00})) \tag{4.5}$$

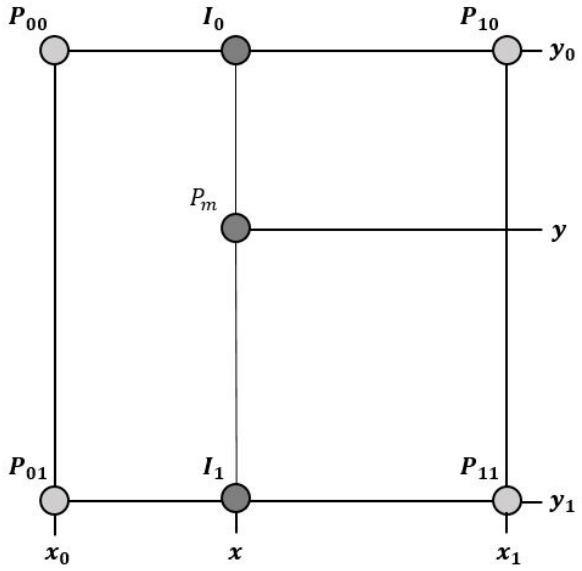

Figure 4.1: Bilinear filtering for P_m in the occupancy grid map (Source: [72])

The problem stated in Equation (4.1) is solved iteratively. Given an initial estimate of the pose ξ_0, the objective is to compute $\Delta\xi$ that minimizes the error measure according to

$$\sum_{i=1}^{n}[1 - M(S_i(\xi + \Delta\xi))]^2 \to 0 \tag{4.6}$$

By applying the first order Taylor expansion to $M(S_i(\xi + \Delta\xi)$, Equation (4.6) becomes:

$$\sum_{i=1}^{n}\left[1 - M(S_i(\xi)) - \nabla M(S_i(\xi))\frac{\delta S_i(\xi)}{\delta\xi}\Delta\xi\right]^2 \rightarrow 0 \tag{4.7}$$

By setting the partial derivative with respect to $\Delta\xi$ to zero, Equation (4.7) can be minimized as follows:

$$2\sum_{i=1}^{n}\left[\nabla M(S_i(\xi))\frac{\delta S_i(\xi)}{\delta\xi}\right]^T\left[1 - M(S_i(\xi)) - \nabla M(S_i(\xi))\frac{\delta S_i(\xi}{\delta\xi}\Delta\xi\right] = 0 \tag{4.8}$$

where from Equation (4.2)

$$\frac{\delta S_i(\xi)}{\delta\xi} = \begin{pmatrix}(1 & 0 & -sin(\psi)s_{i,x} & -cos(\psi)s_{i,y}\\ 0 & 1 & cos(\psi)s_{i,x} & -sin(\psi)s_{i,y}\end{pmatrix} \tag{4.9}$$

Solving for $\Delta\xi$ yields the Gauss-Newton equation for the minimization problem:

$$\Delta\xi = H^{-1}\sum_{i=1}^{n}\left[\nabla M(S_i(\xi))\frac{\delta S_i}{\delta\xi}\right]^T[1 - M(S_i(\xi))] \tag{4.10}$$

where

$$H = \left[\nabla M(S_i(\xi))\frac{\delta S_i}{\delta\xi}\right]^T\left[\nabla M(S_i(\xi))\frac{\delta S_i}{\delta\xi}\right] \tag{4.11}$$

4.2 SLAM implementation using ROS

After the review of the algebraic calculations underlying Hector algorithm proposed in Section 4.1, the aim of this section is to describe the testing methodology and results of Hector-SLAM in an indoor environment, where all the experiments are carried out based on the Robot Operating System (ROS), a robotic middleware for the large-scale integration of devices to build robotic systems. Actually, members of the team Hector from the Technical University of Darmstadt have developed a dedicated software to perform Simultaneous Localization and Mapping in unstructured environments which is available as an open source package for ROS, called *hector_slam*. This section thus provides the background knowledge required to understand the work presented in this chapter, including information about ROS framework and the modules provided by the *hector_slam* package to perform Simultaneous Localization and Mapping.

4.2.1 Robot Operating System (ROS)

The robotics community has achieved remarkable breakthroughs in recent years, offering cutting-edge robotic hardware and developing algorithms that help those robots operate with an increasing level of autonomy [77]. Despite these rapid advances, robots still pose significant challenges for developers and researchers owing to the increasing complexity of the systems: writing software for robots is difficult, especially as the scale and scope of robotics continues to grow [78]. Different types of robots can have extremely different hardware, making code reuse non-trivial; moreover, as more functionalities are introduced, the software gradually becomes more complex and, together with the breadth of skills required, makes the need to develop robotic architectures capable of supporting large-scale software integration compelling. To deal with these challenges, a wide variety of robotics frameworks have been developed to manage complexity and facilitate rapid prototyping of software for experiments. In this context, ROS was originally developed in 2007 by the Stanford Artificial Intelligence Laboratory and, since 2008, Willow Garage has started to make contributions to the native framework, with improvements to the core software and an increasing number of packages to form a broader software ecosystem.

The official description of ROS is:

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers [79].

According to the proposed definition, ROS can be defined as an open-source framework designed to build software for various types of robots, providing libraries and tools that facilitate the process of creating and connecting complex robotic systems [80], thus resulting in a middleware architecture that does not replace, but rather works alongside a traditional operating system [77].

The ROS philosophy can be summarised and framed within five main design criteria [78]:

- *Peer-to-peer (distributed)*: a ROS-based system consists of a number of

processes and programs - possibly running on a number of different hosts - networked together at runtime in a peer-to-peer topology over defined APIs. For instance, many service robots are typically equipped with several on-board computers that are connected via Ethernet. This network server, in turn, is connected to external machines that are responsible for performing computationally intensive tasks such as computer vision and object recognition. The peer-to-peer topology makes it possible to overcome the limitations of frameworks based on the central server, which proves to be inconvenient in case of computers connected in a heterogeneous network. However, the common thread in these frameworks is the need for communication among multiple processes: the peer-to-peer topology calls for a sort of search mechanism to allow processes to find each other at runtime. For this purpose, ROS provides the master mechanism, which will be described in the following.

- *Tools-based*: to manage the overall complexity of the framework, instead of building a monolithic development and execution environment, ROS relies on a set of tools that are deployed to build and run various components, including tools for compilation, data plotting, parameters configuration, visualization of peer-to-peer connection topology, documentation generation, and so on. Custom tools can also be written by the user in the form of new packages.

- *Multi-lingual*: designed to be language-neutral, ROS supports four programming languages, such C++, Python, Octave, and LISP. Actually, peer-to-peer configuration relies only on XML-RPC, for which a reasonable implementation exists in most major languages, either by directly writing the full library that interacts with ROS core, or by building a wrapper of the ROS C++ library.

- *Thin (light-weight)*: ROS tries to overcome the limitations of many robotics software projects, which, although containing drivers and algorithms that could be conceptually transferred to other external projects, in practice are based on a code that is so entangled with the middleware that it is difficult to extract and reuse outside its original context. To cope with this issue, ROS is built by developing algorithms and drivers in independent libraries, following a modular logic. By putting virtually all the complexity in libraries and creating small executables that expose the desired functionality to ROS, it is then possible to facilitate the extraction and reuse of code beyond its

original scope.

- *Free and Open-Source*: ROS source code is publicly available and distributed under the terms of the BSD License, which allows the development of both non-commercial and commercial projects.

The implementation of the ROS software is based on a set of fundamental components, namely nodes, messages, topics, and services.

The execution of ROS is partitioned into smaller pieces of code, called *nodes*, which are responsible for performing dedicated tasks. In a broader sense, nodes can be considered as mostly stand-alone programs that perform computations. Based on the modular logic underlying the ROS environment, a robotic application is usually a collection of nodes that run in parallel and provide a specific task. For this to work, these nodes must be able to communicate with each other. The part of ROS that aids this communication is called *ROS master*, which provides a lookup mechanism that allows ROS to be initialized and nodes to find each other through registration.

The main mechanism that ROS nodes use to communicate is the transmission of *messages*, which consist of strictly typed data structures, including standard primitive types (integer, floating point, Boolean, etc.) as well as arrays of primitive types and constants with a variable level of complexity and nesting [78].

For nodes to communicate, they interact by publishing messages in *topics*, which can be broadly defined as channels used by nodes to exchange asynchronous information. In other terms, topics are buses where each node can publish messages following the message type standards. In turn, the other active nodes can then subscribe to topics that are relevant to them and act upon data captured on the topic [80]. The idea is that a node that wants to share information will *publish* messages on the appropriate topic or topics; a node that wants to receive information will *subscribe* to the topic or topics that it is interested in [77]. The ROS master ensures that the publisher and subscribers can find each other, while the messages themselves are sent directly from publisher to subscriber: nodes can ask the master for a publisher or subscriber of a certain topic and nodes can then communicate directly with each other [80], as depicted in Figure 4.2

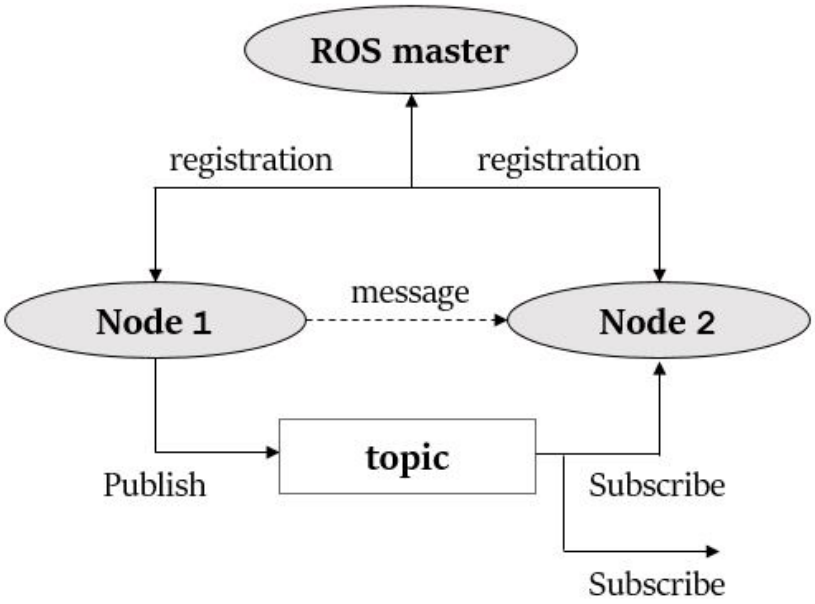

Figure 4.2: Asynchronous communication via topics

Although the topic-based publish-subscribe approach is the principal communication paradigm in ROS, it shows relevant limitations in case of synchronous transactions, for instance when sending a message that requires a reply. Actually, in certain tasks it is more efficient to let a node send a request for a message rather than subscribing to a topic. This is particularly true for tasks that need a lot of computing power but only need to be executed once in a while, so instead of subscribing to a node that computes it in every iteration, it is preferable to send a request only when it is needed to retrieve data from the intended node. In this case, it is better to use *services* instead of topics. ROS services are defined by a string name and a pair of typed messages, one for the *request* and one for the *response*. A node provides a service by listening to an expected request and then responding when the request is issued by another node. Simply put, the idea is that a *client node* sends some data called a request to a *server node* and waits for a reply. The server, upon receiving this request, performs some actions and then sends some data as a response back to the client. The specific content of the request and response data is determined by the service data type, which is analogous to the message types transmitted through the topics. However, services differ from messages in two main respects: services are *bidirectional*, as information flows in both directions, unlike messages, for which, once published, there is no notion of response; services are suitable for *one-to-one communication*, allowing synchronous transaction between nodes. Therefore, each service is initiated by a

node and the response returns to the same node; on the contrary, each message is transmitted through a topic that may have several publishers and subscribers. Figure 4.3 depicts the general scheme of service call communication between two nodes.

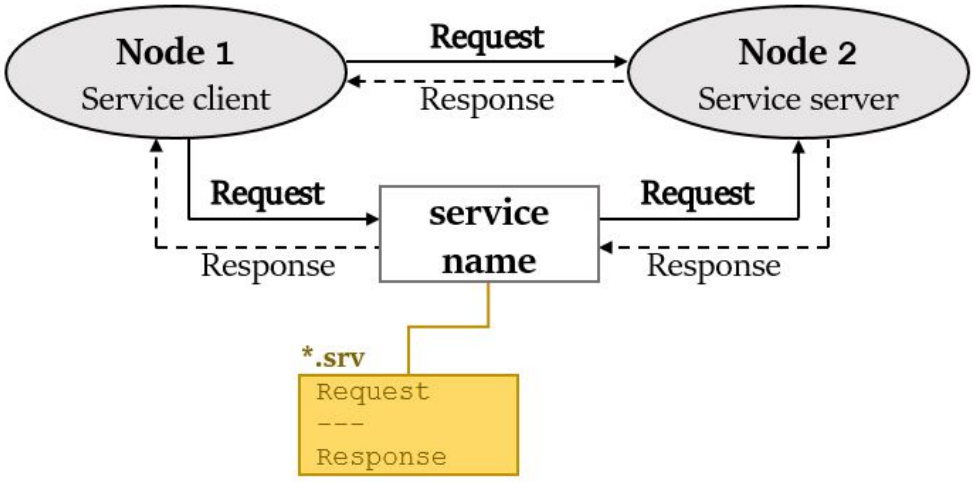

Figure 4.3: Synchronous communication via services

To build a robust and modular robotic system and to support collaborative development, ROS software is organised into *packages*, which are consistent collections of files, typically comprising both executables and support files, that are intended for a specific purpose [77]. Each package is identified by a manifest, which is a file - called *package.xml* - that provides some details about the package, including its name, maintainer and dependencies. The main advantage of the package architecture is to make the code organized in its own subsystems, which become specialized in performing a precise functionality so that other packages and processes can exploit them in carrying out their tasks. The open-ended nature of ROS packages allows for great variation in their structure and purpose: some packages wrap existing software enabling the transfer of their functionalities, others provide standalone libraries and executables, and still others provide scripts to automate demonstrations and tests. The package-based structure aims to divide ROS software into small and manageable segments, each of which can be maintained and extended by its own team of developers [78].

4.2.2 `hector_slam` package

As mentioned in the introduction to Chapter 4, the Hector Team at the Technical University of Darmstadt has provided an open source meta-package in ROS,

called `hector_slam`, aimed at the implementation of the complex capabilities underlying the SLAM algorithm for self-localization and mapping in unknown environments, reported in Section 4.1. In line with the packaged architecture of the ROS framework, `hector_slam` comprises a set of modules that form the building blocks for a system capable of autonomous exploration in unstructured environments such as those encountered in Urban Search and Rescue (USAR) scenarios [81]. All modules were successfully applied and originally tested in the RoboCup Rescue competition, an annual robotics tournament to promote research in robotics and AI. Since then, the `hector_slam` package has been reused and adopted by numerous research groups for a wide variety of tasks.

By exploiting the `hector_slam` package in the Robot Operating System, it is possible to process LiDAR data in order to generate better and more accurate mapping views, thus allowing the system to supply information about the room map in real time. This is accomplished by executing a set of specifically designed nodes - embedded in libraries within the `hector_slam` metapackage - that are responsible for performing dedicated tasks. By definition of a metapackage, referred to a bundle of applications, libraries and documentation, `hector_slam` serves multiple scopes through the integration of the following modules: `hector_compressed_map_transport`, `hector_geotiff`, `hector_geotiff_plugins`, `hector_imu_attitude_to_tf`, `hector_map_server`, `hector_map_ tools`, `hector_mapping`, `hector_marker_drawing`, `hector_nav_msgs`, `hector_slam_launch`, `hector_trajectory_server`. Each of them is purposely intended for a specific application.

Considering the objectives of the present project, the relevant packages that will be used in practice are `hector_mapping`, `hector_geotiff`, `hector_slam_launch` and `hector_trajectory_server`.

`hector_mapping` is the node in charge of performing the SLAM, learning the environment map and estimating the 2D platform pose at the frame rate of the laser scanner, harnessing the high update frequency of modern LiDAR systems (such as YDLIDAR X4). To work properly, whilst odometry data is not needed, the `hector_mapping` node entails two main hardware requirements, namely a source of laser scan data (transmitted as a `sensor_msgs/LaserScan` message) and the `tf` package for the transformation of the scan data.

Transformations, or `tf` in short, are the way ROS deals with coordinate frames in

space. Since the poses of each link in a robot can vary over time, it keeps track of these changes and supplies tools to assist the user in making transformations with the data. For instance, each robotic system moving in a space is defined by at least two frames: `map` and `base_link`. The `map` coordinate frame is a world-fixed reference system, with its z-axis pointing upwards. In a typical setup, a localization component constantly recomputes the robot pose in the `map` frame based on sensor observations with reduced drift over time [82]. The `base_link` coordinate frame, on the other hand, is rigidly attached to the base of the mobile robot in an arbitrary position or orientation. Hypothesizing now the presence of a LiDAR installed on the robot, this will be defined by an additional frame, which will be called `laser_link`. Since the range measurements captured by the sensor are referenced to the `laser_link` itself, the advantage of the `tf` package is that it automatically computes transformations between coordinate frames, thus calculating the laser readings relative to any link of interest, allowing, for example, to recompute the distance to obstacles with respect to the robot body, rather than in relation to the sensor frame.

To provide a broader overview of the system, Figure 4.4 shows all potential frames of interest in a simplified 2D view of a robot moving in a defined environment.

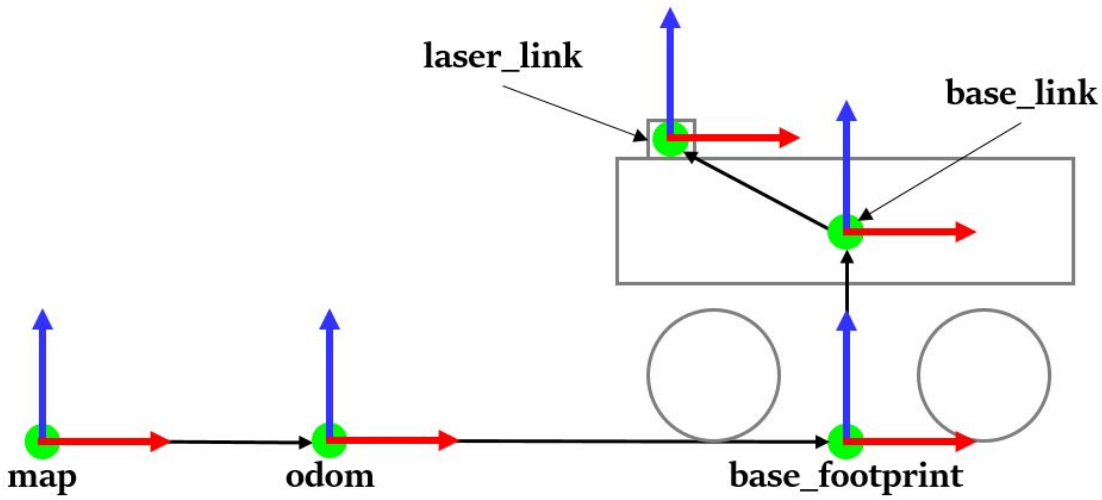

Figure 4.4: Coordinate frames for mobile platforms

Two frames are usually used between the `map` and the `base_link`. The `odom` frame is a world-fixed reference system with respect to which the robot pose can drift in time, serving as an accurate local short-term reference. In a conventional setup, the `odom` frame is estimated based on a source of odometry, such as

wheel/visual odometry, or on an inertial measurement unit. The `base_footprint` frame does not provide height information and accounts for the 2D position and orientation of the robot platform. Starting from this general configuration, the different frames can be used or omitted depending on the specific application by modifying the associated parameters in the package. For instance, in the case of Hector-mapping, the `odom` frame will not be considered since odometry data is not strictly necessary in the execution of the SLAM algorithm.

The basic working principle of the `hector_mapping` node is almost straightforward. Under the assumption that laser data is available, once initialised, the node subscribes to the `scan` topic through which `LaserScan` messages containing LiDAR readings are transmitted. Range measurements are automatically converted with respect to the `base_link` via a `tf` node (`base_link_to_laser4`) that periodically broadcasts the relative position between the `laser_link` and the `base_link`. Then, the transformed range measurements are processed to perform the scan matching and abstract the occupancy grid maps of the environment: then, at each iteration, the `hector_mapping` node publishes the map data on the `map` topic, which will be subscribed by other nodes, for example to display the obtained results. Together with the map abstraction, the laser data is also leveraged to perform the estimation of the current robot's pose within the map frame: the pose evaluation is published on a `slam_out_pose` topic. At the same time, during program execution, a `tf` node (`map_nav_broadcaster`) is in charge of broadcasting a transform between the map frame and the base frame, which will be used by the `hector_trajectory_server`, as will be illustrated shortly.

`hector_trajectory_server` keeps track of the trajectories extracted from the transmitted `tf` data and makes this information accessible via a service and a topic. In other words, this package provides a node that is responsible for saving robot trajectories that are abstracted from the `tf` data which is periodically broadcasted between a given target frame (namely, `base_link` in this case) and a source frame (`map` in this case). Internally, the program listens to `tf`, performs the necessary transformations, and finally pushes the resulting poses into the saved trajectory, which is updated on a regular basis. Unlike the `hector_mapping` node, which continuously streams map data on the related topic, the `hector_trajectory_server` node internally saves the trajectory information as a `nav_msgs/Path` message, which can be accessed using a service. Therefore, the node, upon request of any service client within the network, acts as a service server, providing the necessary information which is then published on a trajectory topic only when needed.

Figure 4.5 provides a basic scheme that illustrates the relationships between each node described so far and the topics that they communicate with, outlining a synthetic picture of the process.

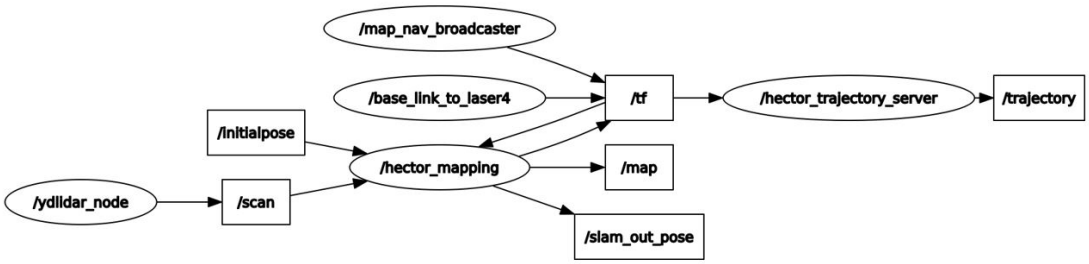

Figure 4.5: Diagram of implemented ROS nodes and topics

In the presented graph, circles and squares represent nodes and topics respectively, while arrows indicate the direction in which communication takes place.

Assuming that the necessary sensor data is supplied to the system according to ROS standards, once the packages are properly configured and executed, the described nodes enable the construction of an accurate map of the environment, as well as the extraction of the trajectory into a moving path, both of which can be displayed in a ROS-based GUI interface. At this point, to make the map and trajectory information available for further uses and purposes, such as path planning and autonomous navigation, it is necessary to store the retrieved information in dedicated files. For this purpose, a specific module provided by the `hector_slam` metapackage comes in handy: `hector_geotiff`. This package supports a node that can be used to save the occupancy grid map, robot trajectory and objects of interest in GeoTIFF compliant images, which is a public domain metadata standard that has georeferencing information embedded in the image. The georeferencing information is included by means of tags that contain spatial information about the image file, such as map projection, coordinate systems and resolution.

In its basic mechanism, the node employs services to retrieve the requested data:

- Map data is retrieved using a specific map service by forwarding the request with a `nav_msgs/GetMap` message.
- The robot trajectory is retrieved by sending a trajectory service request with the message `hector_nav_msgs/GetRobotTrajectory`

By executing the `hector_geotiff` node, the program will generate a GeoTIFF file that will be stored in the maps directory included in the package. This file will then be available to other applications.

4.3 Hector-SLAM testing

To verify the performance of the Hector-SLAM algorithm described in the Sections 4.1 and 4.2.2, a LiDAR system has been designed and implemented in the view of its potential integration into the Bilby Rover mobile robot. The ultimate goal is to test the functionality and compatibility of the `hector_slam` package and of the entire LiDAR system in order to ascertain its transfer viability to the robotic platform to perform SLAM in indoor environments. The hardware description and the implementation of the system using the Robot Operating System as well as the experimental results are addressed in this section.

4.3.1 Hardware overview

Having to test the Hector-SLAM algorithm as a preliminary procedure to its integration on the physical robot, this allows keeping the system architecture rather simple. As a matter of fact, the system will comprise only the components strictly relevant for the execution of the algorithm. It goes without saying that the final architecture of the robot will inevitably become more complex after the integration of structural parts, such as the chassis and the body, and of the components involved in locomotion, including wheels, motors and motor drivers with the related wirings.

Figure 4.6 depicts the two main hardware components used in the presented set of experiments, namely the YDLIDAR X4 sensor and the Raspberry Pi Model B single board computer, whose technical specifications have been already outlined in Chapter 2. YDLiDAR X4 is used as an indoor 2D scanner, using the triangulation method in determining the distance to the objects. The sensor typically operates at a ranging frequency of 5 kHz and is equipped with a DC motor which enables a 360° scanning of the environment, with an angular resolution of 0.25° and a maximum effective range of 12.5 m. YDLIDAR X4 comes with USB adapter board, which enables the sensor to be quickly interconnected with any computer, and provides also a MicroUSB Power Interface (PWR) for auxiliary power supply, which may be required in case the drive current from the standard USB interface of the computer is too low to drive X4.

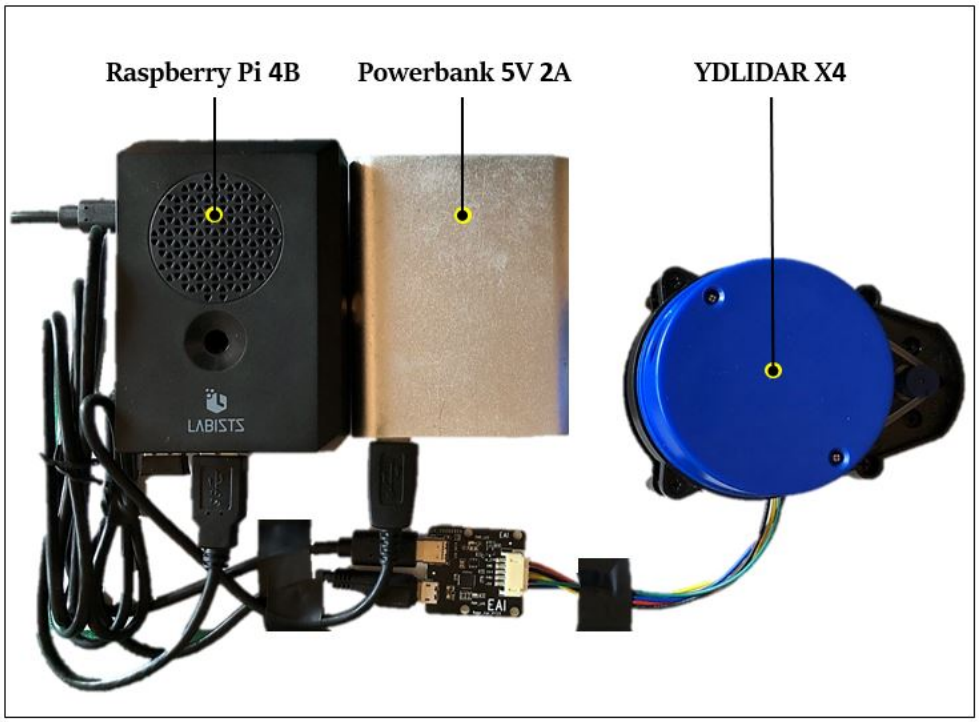

Figure 4.6: Hardware components of the LiDAR system

The LiDAR is in turn connected to the single board computer Raspberry Pi 4B, which is used to access the sensor data and, owing to its small dimensions and high computing capabilities, is widely suitable for robotics applications, serving as a robot controller. The LiDAR data is then sent to a remote computer wirelessly to externalize heavy calculations that would be computationally taxing on the SBC, such as to process range measurements on the scan matching algorithm to obtain map information. For this purpose, both remote computer and SBC Raspberry Pi 4 are integrated into Robot Operating System, which – as it will be further explained in Section 4.3.2 – offers a multi-machine functionality. For the sake of completeness, Figure 4.7 shows the scheme of the overall system used in the experiment.

Figure 4.7: System setup for the implementation of the Hector-SLAM algorithm

4.3.2 Software and network setup

Regardless of its compact dimensions, Raspberry Pi 4, like all its predecessors, is designed as an SBC module, so it can be effectively exploited as a minicomputer for a broad range of applications, being particularly suited to serve as a microcontroller for robotic platforms. Being a minicomputer, Raspberry Pi requires an operating system to work. There is a wide variety of third party operating systems compatible with Raspberry Pi, including Windows 10 IoT Core, MacOS and Ubuntu Mate. Raspberry Pi OS, formerly called Raspbian, is a free Debian-based computer operating system officially provided by the Raspberry Pi Foundation as the primary operating system for the family of single-board computers. It includes a set of basic programs and utilities which allow the development and execution of programs, acting as a bridge between the user and the Pi hardware while managing the CPU, memory, memory disk, GPIO pins and more [83]. As it is compatible with the ROS middleware, in the present project Raspberry Pi OS is chosen as the operating system for the SBC, and it is booted by installing it on an SD card which is flashed into the Raspberry Pi. Once the operating system is correctly configured, SSH (Secure Shell) is enabled: this is a network protocol that allows the Pi board to be operated securely over a network by accessing its command line from another computer. As a matter of fact, in view of the integration of the SBC in the robotic platform that is intended to move autonomously in an environment, it is necessary to access the Raspberry Pi without connecting it to a monitor and other peripherals. Through SSH configuration, therefore, the Raspberry Pi will act as a remote device, allowing the connection to it using a client on another machine.

On the laptop side, the operating system selected for this application is Ubuntu 18.04, which is a distribution based on the Linux kernel. In this regard, in order to run the operating system, VMware Workstation Player is exploited as a virtualization software to perform Virtual Machine operations, thus allowing Ubuntu 18.04 to be executed in an isolated and secure sandbox and in parallel with Windows on the user's laptop.

After the operating systems on both the Raspberry Pi and the remote computer are correctly configured, Robot Operating System is installed on both terminals. Among the different distributions available, ROS Melodic Morenia is installed, being the one primarily targeted at the Ubuntu 18.04 release and fully compatible with Raspberry Pi OS.

The solution of installing ROS on both terminals - instead of installing it only on the Raspberry Pi and controlling the operations remotely via SSH protocol - reflects the purpose of creating a distributed real-time control architecture based on ROS. Actually, although the Raspberry Pi is versatile and practical for many applications, the execution of elaborated programs may require an excessive amount of computing power compared to the capabilities of the single board computer. Considering the specific application of this project as a reference, there are two main tasks to be addressed: on the one hand it is necessary to retrieve data from the Lidar sensor, on the other hand it is necessary to execute the scan matching algorithm for the SLAM implementation. The real-time reading of the sensor data requires a lower computational power, yet a higher frequency of loop execution. Therefore, keeping this function on the Raspberry Pi is the best solution, as it is also evident given the configuration of the system, whereby the Lidar has to be installed on the robot platform. On the contrary, the scan matching algorithm entails the execution of much heavier calculations, which is something that the Raspberry Pi does not easily handle. In this scenario, to reduce power consumption and to minimize the number of lags, it is advisable to externalize the implementation of the Hector-SLAM algorithm to another computer, working remotely from the Raspberry Pi.

Robot Operating System comes in handy for this purpose, being designed with a distributed computing in mind [84]. A running ROS system may comprise a massive number of nodes, scattered across multiple machines that communicate with each other through a bidirectional connectivity between all pairs of terminals. As a matter of fact, a properly configured node makes no assumptions about where in the network it is running, thus allowing computation to be relocated at runtime to match available resources [85]. Deploying a ROS system across multiple machines is almost straightforward: one of the machines is designated to run the master, which must be unique; all the nodes in the network, therefore, must be configured to use the same master, regardless of the terminal on which each program is running. This connection is established via the IP address of every machine in the network: in such a way, each machine is able to advertise itself with a recognizable name so that all the other machines in the distributed setup can resolve. In this ROS distributed network, the master device is in charge of running the ROS core, which entails the implementation of the ROS master instance, providing topic name services and managing peers connection.

Under these considerations, the main purpose is to leverage the ROS distributed

computing environment to establish a master-slave relationship between the Raspberry Pi and the remote computer. In such a framework, the SBC will be the master, responsible for running the master node, while the remote laptop will serve as the slave in the network. The Figure 4.8 provides a basic representation of the distributed network configuration with the definition of the relative functions carried out by the individual instances.

ROS Master (Raspberry Pi)	**ROS Slave (Linux Server)**
Drive LiDAR	Execute Hector-SLAM algorithm
Receive LiDAR data and distribute it via ROS topic	Save and replay the indoor map
	Visualization of sensor data, map and localization

Figure 4.8: Master-Slave configuration of the network

Considering the configuration of the network and the related allocation of tasks, at this point specific ROS packages need to be installed on the two machines to perform their associated tasks. In this framework, two packages are necessary: on the slave machine, the `hector_slam` package is installed for the implementation of the SLAM algorithm as described in the 4.2.2 section; as far as the ROS master is concerned, the `ydlidar` package must be integrated into the ROS environment. The `ydlidar` package is provided directly by the YDLIDAR X4 sensor manufacturer, Shenzhen EAI Technology Co. Ltd, and it consists of all the software components, files and information that must be supplied for the device to be supported by the operating system and interfaced with ROS.

Among the modules included in the package, the file launcher `lidar.launch` allows the execution of a dedicated node that powers the LiDAR while retrieving sensor data, thereby processing the sensor measurements to extract range information and iteratively broadcasting it on the `/scan` topic. Through the exploitation of the distributed network that provides bidirectional connectivity between the master and the slave, thus allowing real-time peer-to-peer communication between

the two terminals, the ROS slave running on the remote computer subscribes to the `/scan` topic to fetch the stream of range measurements and exploits this information to perform SLAM, as described in the Section 4.1. The resulting map and trajectory can then be displayed in `rviz`, namely a dedicated graphical interface embedded into ROS.

Figure 4.9 depicts a high-level representation of the distributed network process, showing the ROS core and the two main nodes - `ydlidar` and `hector_slam` - that communicate over their relative topics, represented as squares in the scheme, to enable the visualization of the results on `rviz`.

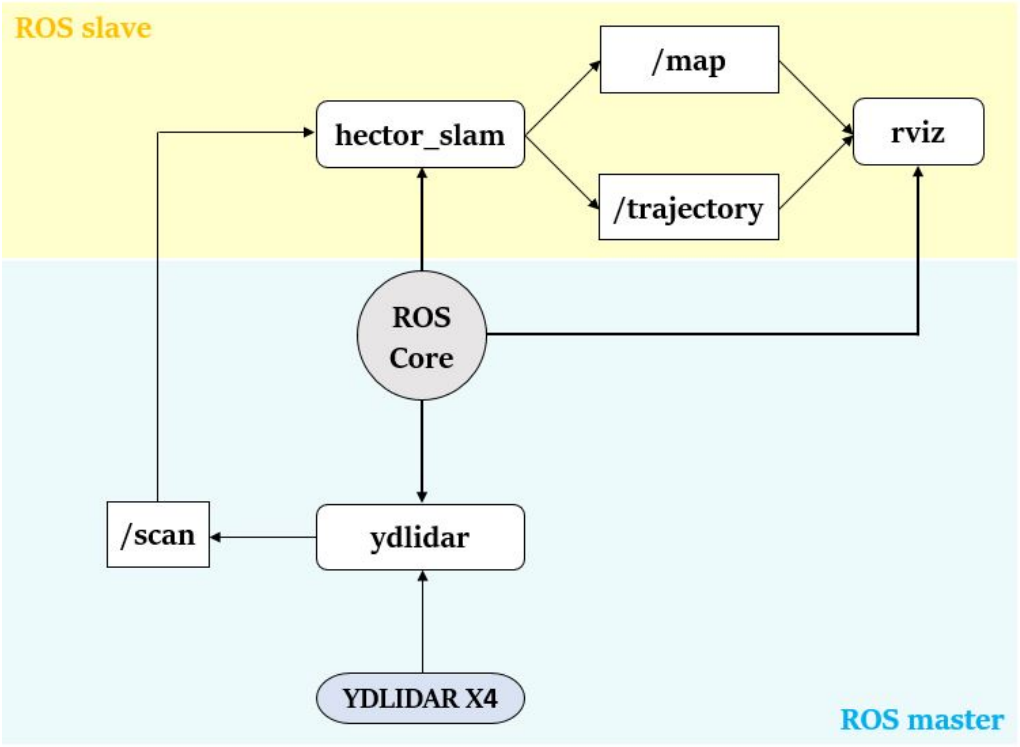

Figure 4.9: High-level representation of the distributed network process

4.3.3 Results and analysis

Distance measurements

Before implementing the Hector-SLAM algorithm, as a preliminary step, a set of LiDAR measurements is collected. In this respect, the YDLIDAR X4 is connected to the Raspberry Pi and is operated via a dedicated launch program; scan readings are acquired by subscribing to the `/scan` topic, where the `ydlidar` node publishes range information at each loop iteration. For the sake of the accuracy, the experimental measurements are repeated 10 times for each sample distance; the results obtained are then averaged. The Table 4.1 shows the results of the distance measurements in a range between 0.5 m and 5 m, providing the error expressed in percentage. Producing an average error of 0.253%, the results demonstrate the suitability of the sensor for indoor mapping.

Table 4.1: Distance measurements

Nominal value $[m]$	**Measured value** $[m]$	**Error** [%]
0.500	0.501	0.30
1.000	0.996	0.35
1.500	1.495	0.33
2.000	1.993	0.34
2.500	2.497	0.11
3.000	2.990	0.32
3.500	3.493	0.20
4.000	3.990	0.26
4.500	4.494	0.13
5.000	4.991	0.19
Average error		**0.253**

Hector-SLAM results

The configured hardware and software are harnessed to implement the Hector-SLAM algorithm to generate the map of unknown environments and simultaneously track the position within the map. To this end, a series of experiments are carried out in two different environments: a living room of reduced dimensions and a conference room of greater size. To simulate the movement of the robotic platform across the room, the LiDAR sensor and the Raspbbery Pi are installed on a wheeled cart and then moved manually to scan the surrounding area. The LiDAR data is then extracted from the Raspberry Pi and then sent to the remote computer to be processed and visualised using the `rviz` visualisation tool. Once the mapping

process is complete, the abstracted map is saved in a GeoTIFF file that is stored in a dedicated directory. In this respect, it is necessary to exploit ground truth to verify the accuracy of the estimation results. To provide a reference, Figures 4.10 and 4.11 provide drawings of the rooms to be mapped.

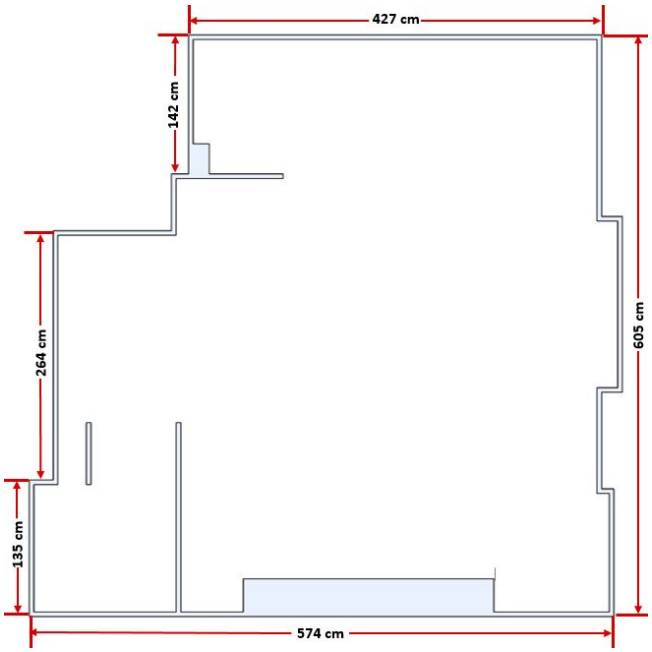

Figure 4.10: Drawing of the room 1 (living room)

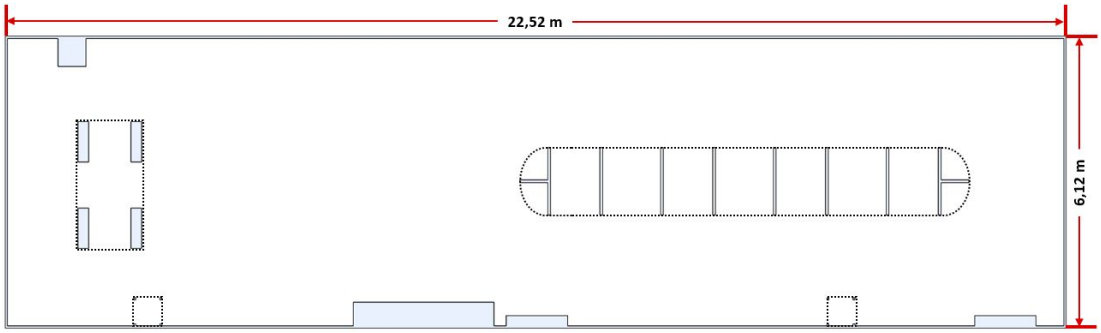

Figure 4.11: Drawing of the room 2 (conference room)

The results obtained from the implementation of the Hector-SLAM algorithm are displayed and compared with the reference design in Figure 4.12 and 4.13 to gauge the precision of the map constructed in both environments. Furthermore, Tables 4.2 and 4.3 provide a comparison between the nominal values of the most relevant features in the map and the measured values during the mapping process.

From the analysis of the obtained results, it is possible to highlight a remarkable accuracy of the constructed maps, showing an average error of 2.93% and 2.96% respectively for room 1 and room 2.

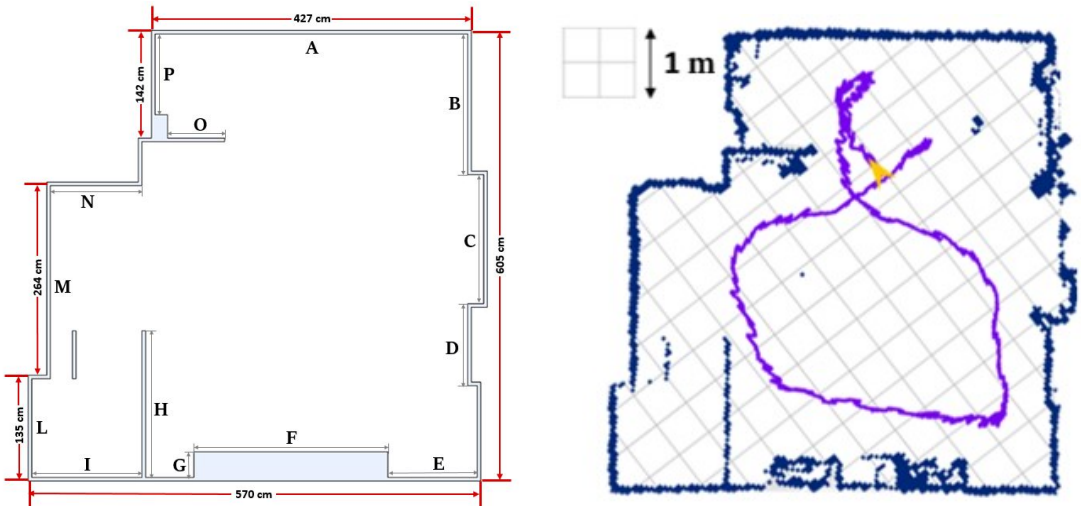

Figure 4.12: Comparison between reference design and constructed map (room 1)

Table 4.2: Comparison between nominal and measured values (room 1)

Room 1 - Living room		
Wall index	**Nominal value** $[m]$	**Measured value** $[m]$
A	4.270	4.269
B	1.920	1.923
C	1.760	1.769
D	1.120	1.192
E	1.220	1.231
F	2.650	2.692
G	0.350	0.385
H	2.000	2.038
I	1.370	1.462
L	1.350	1.346
M	2.640	2.692
N	1.250	1.285
O	0.960	1.008

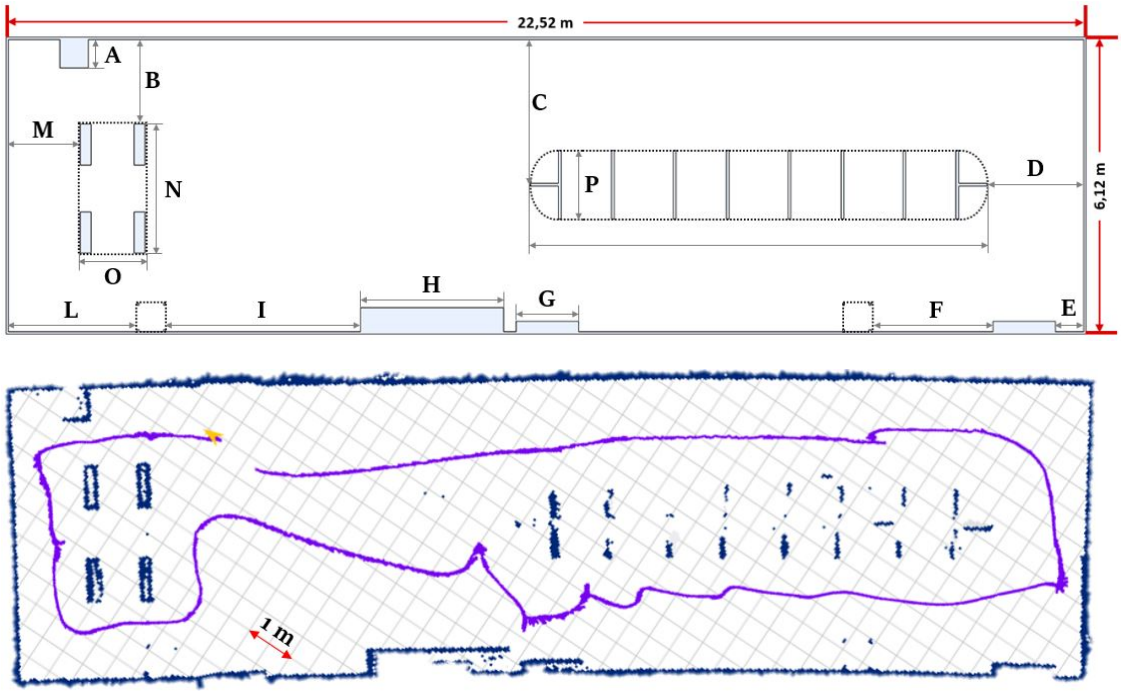

Figure 4.13: Comparison between reference design and constructed map (room 2)

Table 4.3: Comparison between nominal and measured values (room 2)

Room 2 - Conference room		
Wall index	**Nominal value** $[m]$	**Measured value** $[m]$
A	0.600	0.649
B	1.770	1.702
C	3.040	3.010
D	2.020	2.001
E	0.590	0.610
F	2.550	2.590
G	1.310	1.399
H	3.000	2.950
I	4.080	4.100
L	2.690	2.710
M	1.500	1.549
N	2.700	2.801
O	1.360	1.356
P	1.440	1.599

Based on the outcomes of these experiments, it can be reasonably assumed that the system per se and the algorithm can also be compatible and work efficiently when embedded on the Bilby Rover robot platform.

Conclusions and Future Work

In this thesis, stemming from the collaboration with McMaster University's W Booth School of Engineering Practice and Technology, the autonomous mobile robot Bilby Rover has been studied and simulated.

After introducing the landscape of Industry 4.0, exploring its characteristics and opportunities in the manufacturing industry, the study focused on the description of the Bilby Rover robot, which was designed by a team of students from McMaster University using a CAD software dedicated to mechanical design. Based on the 3D model, the overall objective of this thesis has been to investigate the performance of the Bilby Rover engaged in various tasks and to explore a series of technologies that provides the robot with the capability to perceive its surroundings, a fundamental condition for any mobile platform employed in the real world applications. To this end, the project has been divided into two macro-sections, each of them devoted to a specific objective.

In the first section, the objective has been to explore a series of applications of the Bilby Rover enabled by the integrated deployment of specific sensors, suitably calibrated for the intended functions. In this connection, the study was pursued by harnessing the potential of virtual simulation, which represents a fundamental tool for planning and visualization, opening a wide range of options for problem solving and allowing the examination of the robotic system even if it does not physically exist yet. Using Webots robotic simulator, different behaviours - typically helpful in an industrial setting - have been simulated, such as line-following, lane-tracking and object recognition, platooning control, obstacle avoidance, and wall-following. For each of these behaviours, dedicated sensors with optimal operational capabilities for the designated application have been selected.

In the second section, the goal has been to study the Simultaneous Localization and Mapping (SLAM) problem, in view of the implementation of these technologies on the Bilby Rover robot. In this context, the Hector-SLAM algorithm has been tested by building a distributed architecture on ROS. The YDLIDAR X4 sensor has been used as an indoor 2D scanner and connected to the SBC Raspberry Pi 4 B, which has been leveraged to access the distance measurements. The extracted LiDAR data have been then sent wirelessly to a remote computer, in charge of executing the Hector-SLAM algorithm to process the range information into a map of the environment. The results of the mapping experiments have been presented, demonstrating a remarkable accuracy that makes the constructed architecture suitable for real-world Bilby Rover operations.

Given the objectives achieved, possible future developments are various and diverse, as the study is still at an embryonic stage. In this regard, from the hardware point of view, the assembly of the physical robot is certainly left as a future development. In this respect, the platform will have to be integrated with specific components necessary for navigation, namely motors and motor drivers, which must be suitably selected for the system depending on technical specifications and connected with the relevant wirings and power supply. From the software point of view, the ROS framework will have to be configured with dedicated packages to allow the Raspberry Pi to be interfaced with motors through motor drivers. In such a way, the actuation of motors can be triggered through commands issued directly by the Raspberry Pi, which will act as a microcontroller board allowing the robot to move in the environment.
Once the Raspberry Pi and the motors are properly connected and after verifying the functionality of the Hector-SLAM algorithm, a further development could be to install modules that enable path planning and autonomous exploration without the need of any human intervention or command, thus relying only on the knowledge of the environmental map. Also for this purpose, the Hector team has developed packages dedicated to the development of these functionalities, which can be integrated on the ROS framework in order to increase the level of autonomy and the perception capabilities of the Bilby Rover.

Appendix A

Webots Simulations

A.1 Keyboard Teleoperation

```
from controller import Robot, Keyboard

#create the Robot instance
robot = Robot()

#get the time step of the current world.
TIME_STEP = 32

kb=Keyboard()
kb.enable(TIME_STEP)

wheels = []
wheelsNames = ['wheel1_motor', 'wheel2_motor',
               'wheel3_motor', 'wheel4_motor']
for i in range(4):
    wheels.append(robot.getDevice(wheelsNames[i]))
    wheels[i].setPosition(float('inf'))
    wheels[i].setVelocity(0.0)

leftSpeed=0.0
rightSpeed=0.0

print('Click on the World window to enable Keyboard Teleoperation')
print('==========================================================')
```

```
print('Press UP key to move forward, DOWN key to go backwards')
print('Press RIGHT key to turn right, LEFT key to turn left')
print('Release the key to stop the robot')
print('============================================================')

#main loop
while robot.step(TIME_STEP) != -1:
    key=kb.getKey()
    if (key==Keyboard.UP):
        #print('Moving forward')
        leftSpeed=4.0
        rightSpeed=4.0
    elif (key==Keyboard.DOWN):
        #print('Moving backwards')
        leftSpeed=-4.0
        rightSpeed=-4.0
    elif (key==Keyboard.LEFT):
        #print('Turning left')
        leftSpeed=-15.9562
        rightSpeed=17.9562
    elif (key==Keyboard.RIGHT):
        #print('Turning right')
        leftSpeed=17.9562
        rightSpeed=-15.9562
    else:
        leftSpeed=0.0
        rightSpeed=0.0

    wheels[2].setVelocity(leftSpeed)
    wheels[3].setVelocity(rightSpeed)
    wheels[0].setVelocity(leftSpeed)
    wheels[1].setVelocity(rightSpeed)
```

A.2 Line Following

```
from controller import Robot

def run_robot(robot):
    timestep=32
    max_speed=6.28
    #Motors
    wheels = []
    wheelsNames = ['wheel1_motor', 'wheel2_motor',
                   'wheel3_motor', 'wheel4_motor']
    for i in range(4):
        wheels.append(robot.getDevice(wheelsNames[i]))
        wheels[i].setPosition(float('inf'))
        wheels[i].setVelocity(0.0)

    #Enable IR sensors
    left_ir=robot.getDevice('ir0')
    left_ir.enable(timestep)
    right_ir=robot.getDevice('ir1')
    right_ir.enable(timestep)

    print('IR-enabled line following behaviour')
    #Step simulation
    while robot.step(timestep) != -1:

        #read IR sensors
        left_ir_value=left_ir.getValue()
        right_ir_value=right_ir.getValue()

        leftSpeed=max_speed
        rightSpeed=max_speed

        if (left_ir_value < right_ir_value):
            #print("Turn right")
            rightSpeed=-25.4343
            leftSpeed=25.4343
        elif (right_ir_value < left_ir_value):
```

```
            #print("Turn left")
            leftSpeed=-25.4343
            rightSpeed=25.4343

        wheels[2].setVelocity(leftSpeed)
        wheels[3].setVelocity(rightSpeed)
        wheels[0].setVelocity(leftSpeed)
        wheels[1].setVelocity(rightSpeed)

if __name__=="__main__":
    my_robot=Robot()
    run_robot(my_robot)
```

A.3 Lane Keeping and Object Recognition

```
from controller import Robot

def run_robot(robot):
    timestep=32
    max_speed=15
    #Motors
    wheels = []
    wheelsNames = ['wheel1_motor', 'wheel2_motor',
                   'wheel3_motor', 'wheel4_motor']
    for i in range(4):
        wheels.append(robot.getDevice(wheelsNames[i]))
        wheels[i].setPosition(float('inf'))
        wheels[i].setVelocity(0.0)

    #Enable IR sensors
    left_ir=robot.getDevice('ir2')
    left_ir.enable(timestep)
    right_ir=robot.getDevice('ir3')
    right_ir.enable(timestep)

    #Enable camera
    camera=robot.getDevice('camera')
    camera.enable(timestep)
    camera.recognitionEnable(timestep)

    detection=0
    print('Lane Following with smart camera for object recognition')
    print('=======================================================')
    #Step simulation
    while robot.step(timestep) != -1:
        object=camera.getRecognitionObjects()
        if len(object)>0:
            detection+=1
            leftSpeed=0
            rightSpeed=0
            if detection==1:
```

```
                print('WARNING: Obstacle Detected')
                print('Waiting for the pedestrian to cross the road')
                print('==========================================')
        else:
            detection=0
            #read IR sensors
            left_ir_value=left_ir.getValue()
            right_ir_value=right_ir.getValue()

            leftSpeed=max_speed
            rightSpeed=max_speed

            if (left_ir_value < 10):
                #print("Turn right")
                rightSpeed=-9.7498
                leftSpeed=9.7498
            elif (right_ir_value < 10):
                #print("Turn left")
                leftSpeed=-9.7498
                rightSpeed=9.7498

        wheels[2].setVelocity(leftSpeed)
        wheels[3].setVelocity(rightSpeed)
        wheels[0].setVelocity(leftSpeed)
        wheels[1].setVelocity(rightSpeed)

if __name__=="__main__":
    my_robot=Robot()
    run_robot(my_robot)
```

A.4 Platooning control of a robot cluster

```
from controller import Robot, RadarTarget
import struct

def run_robot(robot):
    timestep=32
    max_speed=15
    #Motors
    wheels = []
    wheelsNames = ['wheel1_motor', 'wheel2_motor',
                   'wheel3_motor', 'wheel4_motor']
    for i in range(4):
        wheels.append(robot.getDevice(wheelsNames[i]))
        wheels[i].setPosition(float('inf'))
        wheels[i].setVelocity(0.0)

    #Enable IR sensors
    left_ir=robot.getDevice('ir2')
    left_ir.enable(timestep)
    right_ir=robot.getDevice('ir3')
    right_ir.enable(timestep)

    #Enable radar
    radar=robot.getDevice('radar')
    radar.enable(timestep)

    #Enable receiver
    receiver=robot.getDevice('receiver')
    receiver.enable(timestep)

    #Create instance of emitter
    emitter=robot.getDevice('emitter')

    print('Radar-Emitter-Receiver-enabled Platooning-like behaviour')
    #Step simulation
    while robot.step(timestep) != -1:
        target_number=radar.getNumberOfTargets()
```

```
#print('number of targets seen: ', target_number)
if target_number==0:
    #read IR sensors
    left_ir_value=left_ir.getValue()
    right_ir_value=right_ir.getValue()

    leftSpeed=max_speed
    rightSpeed=max_speed

    if (left_ir_value < 10):
        #print("Turn right")
        rightSpeed=-9.7498
        leftSpeed=9.7498
    elif (right_ir_value < 10):
        #print("Turn left")
        leftSpeed=-9.7498
        rightSpeed=9.7498

else:
    if receiver.getQueueLength()>0:
        message=receiver.getData()
        data=struct.unpack("i",message)

    target=RadarTarget()
    target=radar.getTargets()
    #print(target[0].distance)
    #rint(target[0].azimuth)
    #print(target[0].speed)
    #print('============')
    #if the robot is too close, it must slow down
    if (target[0].distance)>0.7:
        left_ir_value=left_ir.getValue()
        right_ir_value=right_ir.getValue()

        leftSpeed=max_speed
        rightSpeed=max_speed

        if (left_ir_value < 10):
```

```
                    #print("Turn right")
                    rightSpeed=-9.7498
                    leftSpeed=9.7498
                elif (right_ir_value < 10):
                    #print("Turn left")
                    leftSpeed=-9.7498
                    rightSpeed=9.7498

            else:
                leftSpeed=data[0]
                rightSpeed=data[0]
                #print(rightSpeed)
                if (target[0].azimuth) <-0.1:
                    leftSpeed=-9.7498
                    rightSpeed=9.7498
                    #print('turning left')
                elif (target[0].azimuth) >0.1:
                    leftSpeed=9.7498
                    rightSpeed=-9.7498
                    #print('turning right')

        wheels[2].setVelocity(leftSpeed)
        wheels[3].setVelocity(rightSpeed)
        wheels[0].setVelocity(leftSpeed)
        wheels[1].setVelocity(rightSpeed)

        message = struct.pack('i',max_speed)
        emitter.send(message)

if __name__=="__main__":
    my_robot=Robot()
    run_robot(my_robot)
```

A.5 Obstacle avoidance control

```
import math
from controller import Robot

robot=Robot()
TIME_STEP=int(robot.getBasicTimeStep())

#define constants
CRUISING_SPEED=5.0
MAX_SPEED=6.4
OBSTACLE_THRESHOLD=0.1
DECREASE_FACTOR=0.9
BACK_SLOWDOWN=0.9

def gaussian(x,mu,sigma):
    return (1/(sigma*math.sqrt(2*math.pi)))*math.exp(-((x-mu)
            *(x-mu))/(2*sigma*sigma))

#Enable Lidar
lidar=robot.getDevice('lidar')
lidar.enable(TIME_STEP)
lidar.enablePointCloud()

lidar_width=lidar.getHorizontalResolution()
half_width=int(lidar_width/2)
max_range=lidar.getMaxRange()
range_threshold=max_range/80

#initialize Braitenberg coefficients
braiten=[]
for i in range(lidar_width):
    braiten.append(0)

for i in range(lidar_width):
    braiten[i]=gaussian(i, lidar_width/2, lidar_width/5)
```

```
#Motors
wheelsNames=['wheel1_motor','wheel2_motor',
             'wheel3_motor','wheel4_motor']
wheels=[]
for i in range(4):
    wheels.append(robot.getDevice(wheelsNames[i]))
    wheels[i].setPosition(float('inf'))
    wheels[i].setVelocity(0.0)

#initialize speed for each wheel
backleftspeed=0.0
frontleftspeed=0.0
frontrightspeed=0.0
backrightspeed=0.0

#initialize dynamic variables
left_obstacle=0.0
right_obstacle=0.0
avoidwall=0
#main Loop
while robot.step(TIME_STEP)!=-1:
    #get lidar values
    lidar_values=lidar.getRangeImage()

    #apply Braiteberg coefficients on lidar values:
    #near obstacle sensed on the left side
    for i in range(half_width):
        #far obstacles are neglected
        if lidar_values[i]<range_threshold:
            left_obstacle+=braiten[i]*(1-lidar_values[i]/max_range)

        #near obstacle sensed on the right side
        j=lidar_width-i-1
        if lidar_values[j]<range_threshold:
            right_obstacle+=braiten[j]*(1-lidar_values[j]/max_range)

    #overall front obstacle
    obstacle=left_obstacle+right_obstacle
```

```
#compute speed according to information on obstacles
if obstacle>OBSTACLE_THRESHOLD:
    speed_factor=(1-DECREASE_FACTOR*obstacle)*MAX_SPEED/obstacle
    frontleftspeed=speed_factor*left_obstacle
    frontrightspeed=speed_factor*right_obstacle
    backleftspeed=BACK_SLOWDOWN*frontleftspeed
    backrightspeed=BACK_SLOWDOWN*frontrightspeed

else:
    backleftspeed=CRUISING_SPEED
    frontleftspeed=CRUISING_SPEED
    backrightspeed=CRUISING_SPEED
    frontrightspeed=CRUISING_SPEED

if avoidwall>0:
    avoidwall-=1
    if lidar_values[0]>lidar_values[179]:
        #Turn left
        backleftspeed=-1.0
        frontleftspeed=-1.0
        backrightspeed=1.0
        frontrightspeed=1.0
    else:
        #Turn right
        backleftspeed=1.0
        frontleftspeed=1.0
        backrightspeed=-1.0
        frontrightspeed=-1.0
else:
    if backleftspeed<1.1 and backrightspeed<1.1
    and frontrightspeed<1.1 and frontleftspeed<1.1:
        avoidwall=100

#set velocity to the wheels
wheels[0].setVelocity(frontleftspeed)
wheels[1].setVelocity(frontrightspeed)
wheels[2].setVelocity(backleftspeed)
wheels[3].setVelocity(backrightspeed)
```

A.6 Wall following

```
from controller import Robot, RadarTarget

def run_robot(robot):
    timestep=32
    max_speed=10
    #Motors
    wheels = []
    wheelsNames = ['wheel1_motor', 'wheel2_motor',
                   'wheel3_motor', 'wheel4_motor']
    for i in range(4):
        wheels.append(robot.getDevice(wheelsNames[i]))
        wheels[i].setPosition(float('inf'))
        wheels[i].setVelocity(0.0)

    #Enable Sonar sensors
    left_sonar=robot.getDevice('sonar0')
    left_sonar.enable(timestep)
    front_sonar=robot.getDevice('sonar1')
    front_sonar.enable(timestep)

    #Enable radar
    radar=robot.getDevice('radar')
    radar.enable(timestep)

    goal=1
    dest=0
    print('Sonar-based Wall Following behaviour')
    #Step simulation
    while robot.step(timestep) != -1:
        target_number=radar.getNumberOfTargets()

        #read Sonar sensors
        distance_left=left_sonar.getValue()
        distance_front=front_sonar.getValue()

        left_wall=left_sonar.getValue()<50
```

```
front_wall=front_sonar.getValue()<50
leftSpeed=max_speed
rightSpeed=max_speed

if front_wall:
    #print('Turn right')
    leftSpeed=9.7498
    rightSpeed=-9.7498

else:
    if left_wall:
        #print('Drive forward')
        leftSpeed=max_speed
        rightSpeed=max_speed
        if distance_left<10:
            #print('Too close')
            leftSpeed=9.7498
            rightSpeed=-9.7498

    else:
        #print('Turn left')
        leftSpeed=-9.7498
        rightSpeed=9.7498

if target_number>0:
    target=RadarTarget()
    target=radar.getTargets()
    #print(target[0].distance)
    if (target[0].distance)<0.5:
        leftSpeed=0.0
        rightSpeed=0.0
        goal=0

if goal==0:
    if dest==0:
        print("Arrived to the target")
        dest=1
```

```
        wheels[2].setVelocity(leftSpeed)
        wheels[3].setVelocity(rightSpeed)
        wheels[0].setVelocity(leftSpeed)
        wheels[1].setVelocity(rightSpeed)

if __name__=="__main__":
    my_robot=Robot()
    run_robot(my_robot)
```

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisors from Politecnico di Torino, Prof. Giovanni Belingardi and Prof. Maria Pia Cavatorta, whose patient guidance, enthusiastic encouragement and support were invaluable for me to formulate and develop my thesis in such difficult times.

I would like to thank my supervisor from McMaster University, Prof. Ishwar Singh, for giving me the opportunity to join this project and for guiding me throughout the development of my thesis, providing me with the tools I needed to choose the right direction and successfully complete my dissertation.

I would like to acknowledge the Roboteurs team, especially Reiner Schmidt, Anoop Gadhrri and Yih-Chyuan Hsiao, for their incredible contribution and willingness to share their experience with me to overcome the obstacles I encountered along the way.

In addition, I would like to thank my family and my beloved ones for their wise advice and sympathetic ear, always encouraging me during these months. Finally, I could not have completed this thesis without the support of my friends, who provided stimulating discussions as well as happy distractions to rest my mind outside of my research.

Bibliography

[1] L. Belli, L. Davoli, A. Medioli, P. L. Marchini, G. Ferrari, (2019) *Toward Industry 4.0 With IoT: Optimizing Business Processes in an Evolving Manufacturing Factory*, Front. ICT 6:17. doi: 10.3389/fict.2019.00017

[2] Deloitte, (2015) *Industry 4.0 Challenges and Solutions for the Digital Transformation and Use of Exponential Technologies*, Switzerland, Zurich

[3] Plarform Industrie 4.0, *Industrie 4.0 – What is it?* (https://www.plattform-i40.de/PI40/Navigation/EN/Industrie40/WhatIsIndustrie40/what-is-industrie40.html)

[4] iSCOOP, *Industrie 4.0: the fourth industrial revolution – guide to Industrie 4.0* (https://www.i-scoop.eu/industry-4-0/)

[5] A. Rojko, (2017) *Industry 4.0 Concept: Background and Overview*, iJIM – Vol. 11, No. 5

[6] H. Kagermann, R. Anderl, J. Gausemeier, G. Schuh, W. Wahlster, (2016) *Industrie 4.0 in a Global Context: Strategies for Cooperating with International Partners* (acatech STUDY), Munich: Herbert Utz Verlag

[7] K. Schwab, (2016) *The Fourth Industrial Revolution: what it means, how to respond*, World Economic Forum

[8] M. Lorenz, M. Rüßmann, M. Waldner, P. Engel, M. Harnisch, J. Justus, (2015), *Industry 4.0: The Future of Productivity and Growth in Manufacturing Industries, BCG publications*

[9] S. Vaidya, P. Ambad, S. Bhosle, (2018) *Industry 4.0 – A Glimpse*, 2nd International Conference on Materials Manufacturing and Design Engineering, Procedia Manufacturing

[10] K. Witkowski, (2017), *Internet of Things, Big Data, Industry 4.0 - Innovative Solutions in Logistics and Supply Chains Management*, 7th International

Conference on Engineering, Project, and Production Management, Procedia Engineering

[11] M. Brettel, N. Friederichsen, M. Keller, (2014), *How Virtualization, Decentralization and Network Building Change the Manufacturing Landscape: An Industry 4.0 Perspective*, International Journal of Mechanical, Aerospace, Industrial, Mechatronic and Manufacturing Engineering, Vol. 8

[12] Deloitte The Netherlands, (2015), *Industry 4.0 – An Introduction*

[13] T. Burns et al, (2019) *A Review of Interoperability Standards for Industry 4.0*, Procedia Manufacturing 38, 646–653

[14] Y. Liao et al, (2017) *The Role of Interoperability in The Fourth Industrial Revolution Era*, IFAC PapersOnLine 50-1, 12434–12439

[15] IEEE, *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*, IEEE Std 610, pp. 1-217, 1991.

[16] OTTO Motors (2017), *Interoperability Is Key to Lean Manufacturing in Industry 4.0*

[17] D. A Zakoldaev et al, (2019): *The interoperability of cyber-physical systems in digital manufacturing of the Industry 4.0*, Journal of Physics: Conference Series 1399 044019

[18] A. Parrot, L. Warshaw, (2017) *Industry 4.0 and the digital twin - Manufacturing meets its match*, Deloitte University Press

[19] M. Marques, C. Agostinho, G. Zacharewicz, R. Goncalves, (2017) *Decentralized decision support for intelligent manufacturing in Industry*, JAISE - Journal of Ambient Intelligence and Smart Environments, IOS Press, 9 (3), pp.299-313. ff10.3233/AIS-170436ff. ffhal-01517407f

[20] F. Almada Lobo, *Manufacturing Software for Industry 4.0 – Embracing Decentralization*, Elektor Business Magazine

[21] Swisslog, *How Industry 4.0 Design Principles Are Shaping the Future of Intralogistics*

[22] N. G. Carvalho, E. W. Cazarini, (2020) *Industry 4.0 – What is it?, Industry 4.0 – Current Status and Future Trends*, IntechOpen, DOI: 10.5772/Intechopen.90068

[23] B. Vogel-Heuser, D. Hess, (2016) *Industry 4.0–Prerequisites and Visions*, Guest Editorial, IEEE Transactions On Automation Science And Engineering, Vol. 13, No. 2

[24] M. Teulieres, P. J. Tilley, L. Bolz, P. M. Ludwig-Dehm, S. Wägner, (2019), *Industrial robotics - Insights into the sector's future growth dynamics*, McKinsey & Company

[25] D. Küpper, M. Lorenz, C. Knizek, K. Kuhlmann, A. Maue, R. Lässig, T. Buchner, (2019) *Advanced Robotics in the Factory of the Future*, Boston Consulting Group

[26] A. Grau, M. Indri, L. L. Bello and T. Sauter, (2017) *Industrial robotics in factory automation: From the early stage to the Internet of Things*, IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society, Beijing, pp. 6159-6164, doi: 10.1109/IECON.2017.8217070

[27] Z. Gao, T. Wanyama, I. Singh, A. Gadhrri, R. Schmidt, (2020) *From Industry 4.0 to Robotics 4.0 - A Conceptual Framework for Collaborative and Intelligent Robotic Systems*, Procedia Manufacturing 46: 591-99

[28] B. Bayram, G. İnce, (2017) A*dvances in Robotics in the Era of Industry 4.0, Industry 4.0: Managing The Digital Transformation*, Springer International, 187-200.

[29] R. Goel, P. Gupta, (2019) *Robotics and Industry 4.0*, A Roadmap to Industry 4.0: Smart Production, Sharp Business and Sustainable Development, 157-69, Springer International, Advances in Science, Technology & Innovation

[30] J.A. Batlle, A. Barjau, (2009), *Holonomy in mobile robots*, Robotics and Autonomous Systems 57, 433–440, Elsevier

[31] M. Poienariu, M. Georgescu, V. Chiroiu, L. Munteanu, (2009), *On the dynamics of non-holonomic systems*

[32] K. Kozłowski, D. Pazderski, (2004) *Modelling and control of a 4-wheel skid-steering mobile robot*, International Journal of Applied Mathematics and Computer Science, Vol. 14, No. 4, 477–496

[33] M. Papoutsidakis, K. Kalovrektis, C. Drosos, G. Stamoulis, (2017) *Design of an Autonomous Robotic Vehicle for Area Mapping and Remote Monitoring*, International Journal of Computer Applications 167(12), 36-41, DOI: 10.5120/ijca2017914496

[34] P. McManamon, (2019) *LiDAR Technologies and Systems*, SPIE, Retrieved from https://app.knovel.com/hotlink/toc/id:kpLDARTS03/lidar-technologies-systems/lidar-technologies-systems

[35] LeddarTech, *Why LiDAR – A brief introduction to LiDAR Technology and its Market Applications*, https://leddartech.com/why-lidar/

[36] Raspberry Pi Trading Ltd, (2021) *Raspberry Pi 4 Computer Model B*, www.raspberrypi.org

[37] E. Küçükkülahlı, R.Güler, (2015) *Open Source Mobile Robot with Raspberry Pi*, Balkan Journal of Electrical & Computer Engineering, Vol.3, No. 4, DOI: 10.17694/bajece.29976

[38] S. Saeedi, M. Trentini, M. Seto, H. Li, (2016) *Multiple-Robot Simultaneous Localization and Mapping: A Review*, J. Field Robotics, 33, 3-46

[39] B. Huang, J. Zhao, J. Liu, (2020) *A Survey of Simultaneous Localization and Mapping with an Envision in 6G Wireless Networks*, Nanyang Technological University, Singapore

[40] C. Cadena et al., (2016) *Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age*, IEEE Transactions on Robotics, vol. 32, no. 6, pp. 1309-1332 doi: 10.1109/TRO.2016.2624754

[41] A. Birk, M. Pfingsthorn, (2016) *Simultaneous Localization and Mapping (SLAM)*, Encyclopedia of Electrical and Electronics Engineering: Wiley

[42] Y. Chen, J. Tang, C. Jiang, L. Zhu, M. Lehtomäki, H. Kaartinen, R. Kaijaluoto, Y. Wang, J. Hyyppä, H. Hyyppä, H. Zhou, L. Pei, R. Chen, (2018) *The Accuracy Comparison of Three Simultaneous Localization and Mapping (SLAM)-Based Indoor Mapping Technologies*, Sensors, 18 (10): 3228. https://doi.org/10.3390/s18103228

[43] Mathworks, *What is SLAM? 3 things you need to know*, https://it.mathworks.com/discovery/slam.html

[44] G. Jiang, L. Yin, S. Jin, C. Tian, X. Ma, Y. Ou, (2019) *A Simultaneous Localization and Mapping (SLAM) Framework for 2.5D Map Building Based on Low-Cost LiDAR and Vision Fusion*, Applied Sciences

[45] C. Chen, L. Pei, C. Xu, D. Zou, Y. Qi, Y. Zhu, T. Li, (2019) *Trajectory Optimization of LiDAR SLAM Based on Local Pose Graph*, China Satellite Navigation Conference (CSNC)

[46] F. Jiménez, M. Clavijo, J. Juana, (2018) *LiDAR-based SLAM algorithm for indoor scenarios*, The Seventh International Conference on Advances in Vehicular Systems, Technologies and Applications

[47] A. Polenghi, L. Fumagalli, I. Roda, (2018) *Role of simulation in industrial engineering: focus on manufacturing systems*, ScienceDirect: IFAC PapersOnLine 51-11, 496–501

[48] F. Hosseinpour, H. Hajihosseini, (2009) *Importance of Simulation in Manufacturing*

[49] C. Murphy, T. Perera, (2001) *Role of simulation in industries: the definition and potential role of simulation within an aerospace company*, Conference: Proceedings of the 33nd conference on Winter simulation, 829-837, DOI:10.1145/564124.564241

[50] P. Neto, J. N. Pires, A. Moreira, (2010) *Robot path simulation: A low cost solution based on CAD*, 2010 IEEE Conference on Robotics, Automation and Mechatronics, 333 - 338

[51] Cyberbotics Ltd, *Webots User Guide*, http://www.cyberbotics.com

[52] O. Michel, (2004) *Professional Mobile Robot Simulation*, International Journal of Advanced Robotic Systems, DOI:10.5772/5618

[53] W. A. Isop, C. Gebhardt, T. Nägeli, F. Fraundorfer, O. Hilliges, D. Schmalstieg, (2019) *High-Level Teleoperation System for Aerial Exploration of Indoor Environments*, Frontiers in Robotics and AI, DOI:10.3389/frobt.2019.00095

[54] R.G. Boboc, H. Moga, D. Talabă, (2012) *A Review of Current Applications in Teleoperation of Mobile Robots*, Bulletin of the Transilvania University of Braşov Series I: Engineering Sciences, Vol. 5 (54), No. 2

[55] S. Oswal, D. Saravanakumar, (2021) *Line following robots on factory floors: Significance and Simulation study using CoppeliaSim*, IOP Conference Series: Materials Science and Engineering, 1012 012008

[56] M. S. Islam, M. A. Rahman, (2013) *Design and Fabrication of Line Follower Robot*, Asian Journal of Applied Science and Engineering, Vol. 2, No. 2

[57] F. Real, F. Berry, (2010) *Smart Cameras: Fundamentals, Technologies and Applications*, Smart Cameras, DOI: 10.1007/978-1-4419-0953-43

[58] M. Della Vedova, T. Facchinetti, A. Ferrara, A. Martinelli, (2009) *Real-time Platooning of Mobile Robots: Design and Implementation*, Proceedings of 12th IEEE International Conference on Emerging Technologies and Factory Automation DOI: 10.1109/ETFA.2009.5347246

[59] B. Brogliato, C. C. de Wit, (1999) *Stability issues for vehicle platooning in automated highway systems*, IEEE International Conference on Control Applications

[60] E. V. Filho et al., (2020) *Towards a Cooperative Robotic Platooning Testbed*, IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC) DOI: 10.1109/ICARSC49921.2020.9096132.

[61] ACEA, (2017) *What is truck platooning?* https://www.acea.be/uploads/publications/Platooningroadmap.pdf

[62] G. Klančar, Gregor, D. Matko, S. Blazic, (2009) *Wheeled Mobile Robots Control in a Linear Platoon*, Journal of Intelligent and Robotic Systems DOI:10.1007/s10846-008-9285-7

[63] M. I. Skolnik, (2020) *Radar*, Encyclopedia Britannica, https://www.britannica.com/technology/radar

[64] J. A. Scheer, W. A. Holm, (2010) *Introduction and Radar Overview*, Principles of Modern Radar: Basic Principles

[65] D. Hutabarat, D. Purwanto, M. Rivai, H. Hutomo, (2019) *Lidar-based Obstacle Avoidance for the Autonomous Mobile Robot*, 12th International Conference on Information & Communication Technology and System

[66] X. Yang, R. V. Patel, M. Moallem, (2006) *A Fuzzy–Braitenberg Navigation Strategy for Differential Drive Mobile Robots*, J Intell Robot Syst, 47: 101-124, DOI: 10.1007/s10846-006-9055-3

[67] I. Rano, (2012) *A model and formal analysis of Braitenberg vehicles 2 and 3*, IEEE International Conference on Robotics and Automation DOI: 10.1109/ICRA.2012.6224583

[68] S. M. Antoun, P. J. McKerrow, (2010) *Wall following with a single ultrasonic sensor*, Conference: Intelligent Robotics and Applications - Third International Conference, ICIRA 2010, Shanghai, China, November 10-12, 2010. Proceedings, Part II, DOI: 10.1007/978-3-642-16587-013

[69] L. Kleeman, R. Kuc, (2008) *Sonar Sensing*, Springer Handbook of Robotics, Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-30301-522

[70] Design World Staff, (2011) *The Search for a Better Proximity Sensor Starts Here*, https://www.designworldonline.com/the-search-for-a-better-proximity-sensor-starts-here/

[71] K. G. Panda, D. Agrawal, A. Nshimiyimana, A. Hossain, (2016) *Effects of environment on accuracy of ultrasonic sensor operates in millimetre range*, Perspectives in Science, DOI: 10.1016/j.pisc.2016.06.024

[72] H. Suwoyo, C. Deng, Y. Tian, A. Adriansyah, (2018) *Improving a Wall-Following Robot Performance with a PID-Genetic Algorithm Controller*, Proceeding of EECSI 2018

[73] S. Kohlbrecher, O. Von Stryk, J. Meyer, U. Klingauf, (2011) *A flexible and scalable SLAM system with full 3D motion estimation*, IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), DOI: 10.1109/SSRR.2011.6106777

[74] M. Rivai, D. Hutabarat, Z. M. J. Nafis, (2020) *2D mapping using omni-directional mobile robot equipped with LiDAR*, TELKOMNIKA Telecommunication, Computing, Electronics and Control, Vol. 18, No. 3, DOI: : 10.12928/TELKOMNIKA.v18i3.14872

[75] Z. Xuexi, L. Guokun, F. Genping, X. Dongliang, L. Shiliu, (2019) *SLAM Algorithm Analysis of Mobile Robot Based on Lidar*, Proceedings of the 38th Chinese Control Conference

[76] J. Wen, C. Qian, J. Tang, H. Liu, W. Ye, X. Fan, (2018) *2D LiDAR SLAM Back-End Optimization with Control Network Constraint for Mobile Mapping*, Sensors 2018, 18, 3668; DOI:10.3390/s18113668

[77] J. M. O'Kane, (2013) *A Gentle Introduction to ROS*, http://www.cse.sc.edu/ jokane/agitr/

[78] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs,(2009) R. Wheeler, A. Ng, *ROS: an open-source Robot Operating System*, ICRA workshop on open source software

[79] ROSwiki, *What is ROS?* http://wiki.ros.org/ROS/Introduction

[80] A. Palsson, M. Smedberg, (2017) *Investing Simultaneous Localization and Mapping for AGV systems*, University of Gothenburg

[81] S. Kohlbrecher, J. Meyer, T. Graber, K. Petersen,U. Klingauf, O. von Stryk, (2014) *Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots*, Conference: Robot Soccer World Cup, DOI: 10.1007/978-3-662-44468-958

[82] Wim Meeussen, (2010) *Coordinate Frames for Mobile Platforms*, https://www.ros.org/reps/rep-0105.html

[83] D. Calin, (2020) *How To Install ROS Melodic, rosserial, and more on Raspberry Pi 4 (Raspbian Buster)*, https://www.intorobotics.com/how-to-install-ros-melodic-rosserial-and-more-on-raspberry-pi-4-raspbian-buster/

[84] ROSwiki, *Running ROS across multiple machines*, http://wiki.ros.org/ROS/Tutorials/MultipleMachines

[85] ROSwiki, *Network Setup*, http://wiki.ros.org/ROS/NetworkSetup