UNED

EIDUNED
Escuela
Internacional
de Doctorado

TESIS DOCTORAL

2023

DEVELOPMENT, CONTROL AND EVALUATION OF A HETEROGENEOUS MULTI-AGENT ROBOTIC PLATFORM

Francisco José Mañas Álvarez

PROGRAMA DE DOCTORADO EN INGENIERÍA DE SISTEMAS Y CONTROL

Dra. María Guinaldo Losada
Dra. Raquel Dormido Canto

To my supports

Abstract

Multi-agent robotic system (MARS) is an extensive field of research in different application domains since they can perform complex tasks even if the behavior of each agent seems simple thanks to cooperation between them. The development of these systems involves several decisions: the type of agents, the control and communication architecture, the operating environment, the positioning system, the technology used in communication, or the control algorithm to implement, among others.

Experimental platforms are essential tools to thoroughly evaluate and validate MARS development in a controlled environment. The goal of this thesis is the development, control, and evaluation of Robotic Park, a multi-agent indoor physical robotic platform for multi-robot system research. Two of its main characteristics are heterogeneity at different levels and flexibility.

First, a detailed description of the overall design and operation of Robotic Park is provided. Various work modes are available on the platform (in the physical environment, in the virtual environment, or under a hybrid scheme) to carry out experiences.

Then the theoretical modelling of robots for both ground and aerial vehicles is presented in detail. The effectiveness of event-based control techniques for reducing controller executions has been demonstrated.

Finally, focusing on the problem of formation control in MARS, several experiences are presented to study the performance of Robotic Park in this context. Specifically, the scalability, the use of digital twins in mixed reality experiences, the control architecture, and the communication protocol are evaluated.

Resumen

Los Sistemas Multi-Agente Robóticos (del inglés Multi-Agent Robotic System, MARS) conforman un extenso campo de investigación en diferentes dominios de aplicación ya que pueden realizar tareas complejas, aunque cada agente lleva a cabo acciones simples gracias a la cooperación entre ellos. El desarrollo de estos sistemas implica varias decisiones: el tipo de agentes, la arquitectura de control y comunicación, el entorno de operación, el sistema de posicionamiento, la tecnología utilizada en la comunicación o el algoritmo de control a implementar, entre otros.

Las plataformas experimentales son herramientas esenciales para evaluar y validar en profundidad el desarrollo de MARS en un entorno controlado. El objetivo de esta tesis es el desarrollo, control y evaluación de Robotic Park, una plataforma de interiores para sistemas multi-agente enfocada a la investigación de sistemas multi-robot. Dos de sus características principales son su heterogeneidad en diferentes niveles y su flexibilidad.

En primer lugar, se proporciona una descripción detallada del diseño general y funcionamiento de Robotic Park. En la plataforma, se pueden utilizar diversos modos de trabajo (en el entorno físico, en el entorno virtual o bajo un esquema híbrido) para llevar a cabo experiencias.

A continuación, se presenta en detalle el modelado teórico de los robots, tanto terrestres como aéreos. Se ha demostrado la efectividad de las técnicas de control basadas en eventos para reducir las ejecuciones del controlador.

Finalmente, en relación con el problema del control de formaciones en MARS, se presentan varias experiencias con el fin de estudiar el rendimiento de Robotic Park en este contexto. Específicamente, se evalúa la escalabilidad, el uso de gemelos digitales en experiencias de realidad mixta, la arquitectura de control y el protocolo de comunicación.

Acknowledgments

En primer lugar, quiero agradecer a mis directoras María Guinaldo y Raquel Dormido estos años de trabajo. Gracias por haberme acogido en la UNED, por la paciencia que han tenido para pulirme como investigador (que no se lo he puesto fácil en ocasiones) y su confianza para el desarrollo y puesta en marcha de un laboratorio desde el inicio. Espero haber tomado buena nota para poder transmitir tan buen ejemplo a los que vengan después.

Asímismo, quiero agradecer a mis compañeros de departamento su colaboración durante estos años en el día a día. Especialmente a Sebastián Dormido Bencomo, que a pesar su jubilación, siempre ha tenido hueco para comentar ideas o dudas sobre todos los frentes de trabajo abiertos. A Cristina Cerrada, Jacobo Saenz y Ernesto Fábregas por su gran compañía en el despacho día a día siempre que las circunstancias lo han permitido. A Sebastián Dormido Canto por sus buenos consejos desde el principio como director del departamento y como director del programa de doctorado después. A los profesores del máster en Ingeniería de Sistemas y de Control de la UNED, Fernando Morilla, Dictino Chaos, Jose Manuel Díaz, Matilde Santos, ... Gracias por haberme permitido enfocar los trabajos de vuestras asignaturas sobre la línea de mi tesis.

Debo mencionar también al Grupo de Investigación de Automática, Robótica y Mecatrónica de la universidad de Almería. A Manolo Berenguel por comentarme la posibilidad de echar la beca que me ha permitido desarrollar mi trabajo. A Jose Carlos Moreno por permitirme continuar colaborando en las actividades del club de robótica. A Jose Luis Guzmán y José Luis Blanco por tener su despacho siempre abierto para comentar ideas. Y especialmente a Ángeles Hoyo y Paco García, con quienes tuve el placer de compartir la estancia en Lund (Suecia). También quiero transmitir mi agradeciemiento al Departamento de Automática y Control de la Universidad de Lund donde realicé tres meses de estancia de investigación. Gracias a Anders

Robertsson, Björn Olofsson y Tore Hägglund por la posibilidad de hacer la estancia y todos los mensajes y conversaciones que compartimos que han contribuido a mejorar la calidad e impacto de mi trabajo investigador.

Finalmente, llegar hasta aquí no habría sido posible sin el apoyo incondicional de mi familia. Han sido tres años complicados marcados por fenómenos como la pandemia de COVID. El camino no siempre ha sido fácil pero siempre habéis estado ahí para levantarme el ánimo cuando parecía estar frente a problemas sin solución o para recordarme que el cerebro necesita dormir y descansar de vez en cuando. Los logros que he podido alcanzar son tan míos como vuestros.

Francisco José Mañas Álvarez

Madrid, noviembre 2023

Table of Contents

Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
CCD	Charge-Coupled Device
CPS	Cyber-Physical System
DDWMR	Differential Drive Wheeled Mobile Robot
DM	Digital Model
DOF	degrees of freedom
DT	Digital Twin
DS	Digital Shadow
FMA	Flying Machine Arena
GPS	Global Positioning System
IAE	Integral Absolute Error
IMU	Inertial Measurement Unit
IoT	Internet of Things
IPS	Indoor Positioning System
ISE	Integral Square Error
ISS	International Space Station
ITAE	Integral of Time-weighted Absolute Error
ITSE	Integral of Time-weighted Square Error

JADE	JAVA Agent DEvelopment Framework
LPS	Loco Positioning System
MAS	Multi-Agent System
MARS	Multi-Agent Robotic System
MAV	Micro Aerial Vehicle
MCU	Microcontroller Unit
MIMO	Multiple-Input Multiple-Output
MIT	Massachusetts Institute of Technology
MoCap	Motion Capture
MP	Mega Píxel
MPC	Model Predictive Control
MR	Mixed Reality
MRPT	Mobile Robot Programming Toolkit
MRS	Multi-Robot System
NASA	National Aeronautics and Space Administration
NAV	Nano Aerial Vehicle
PAV	Pico Aerial Vehicle
PI	Proportional-Integral
PID	Proportional Integral Derivative
PoE	Power over Ethernet
PWM	Pulse Width Modulation
RFID	Radio Frequency Identification
RMSE	Root Mean Square Error
ROS	Robotic Operating System
ROS 2	Robotic Operating System 2
RP	Robotic Park

RTF	Real-Time Factor
RTOS	Real-Time Operating System
SCRIMMAGE	Simulating Collaborative Robots in a Massive Multi-agent Game Environment
SDK	Software Development Kit
SLAM	Simultaneous Localization And Mapping
SMART	Smart Machine and Assistive Robotics Technolog
SPHERES	Synchronized Position Hold Engage Reorient Experimental Satellites
SRMIST	Sri Ramaswamy Memorial Institute of Science and Technology
STAr	Sailboat Test Arena
SUAV	Small Unmanned Aerial Vehicle
TDoA	Time Difference of Arrival
ToF	Time Of Flight
TWR	Two Way Ranging
UMBRELLA	Urban Multi Wireless Broadband and IoT Testing for Local Authority and Industrial Applications
UNED	Universidad Nacional de Educación a Distancia
uROS	micro Robotic Operating System
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
UWB	UltraWide Band
VR	Virtual Reality
VSLAM	Visual Simultaneous Localization And Mapping

List of Figures

List of Tables

Development, control and evaluation of a heterogeneous multi-agent robotic platform

Chapter 1

Introduction

1.1 Multi-Agent Robotic Systems

The interest in process automation is not a recent technological trend. Early automated systems were used in manufacturing/industrial processes to optimize the resource's consumption and improve performance. Currently, our society demands the progressive integration of automation into work environments to enhance the efficiency, productivity, and quality of many industrial processes. This fact requires active research in what is known as "robotics", one of the fastest growing technological fields in the last years.

Sometimes the terms "automation" and "robotics" are used interchangeably, but there are differences between them. Automation is defined as the process of using technology to complete human tasks. In contrast, robotics is the process of developing robots to carry out a particular function. It involves the design, build, program, and use of robots. Automation and robotics have areas where they cross, such as the use of robots to automate physical tasks. However, neither all types of automation use robots (they are not required in virtual tasks and software-based applications), nor all robots are designed for process automation. As technology continues to advance, the capabilities of robots are expanding, enabling them to take on more complex tasks and work alongside humans in increasingly sophisticated ways.

In any case, robotics has become an integral part of many industries, including manufacturing, healthcare, agriculture, and logistics, among others. Applications of robotics in autonomous vehicles, large factories or warehouses that use robots to move storage from one point to another, or social and welfare robotics are found. Within the wide range of applications that can be improved by robots, those related to mobile robotics stand out, being autonomous robots of special interest. And this is the field in which this thesis is framed.

According to IEEE Std 1872-2015 - IEEE Standard Ontologies for Robotics and Automation, **a fully autonomous robot is a robot that can perform tasks without human control**

or intervention while adapting to operational and environmental conditions [1].For instance, an autonomous robot vehicle capable of moving from one point to another without requiring either a guide or teleoperator control can be considered as an autonomous robot, although human presence is necessary for safety reasons. Another example are those robots working in warehouses to load and move materials without the intervention of operators during their operation.

Robots can be made in many ways. But most robots share a great deal in common, as some parts are found in almost every robotic system. A description of the main characteristics and components of an autonomous mobile robot is given in the following. It can help to identify the possibilities and limitations of each one.

To carry out truly autonomous action, an autonomous robot must be equipped with sensors, some form of actuation, a communication system and information-processing capabilities, see Figure 1.1. The arrows in the figure represent the flow of information between the different components.

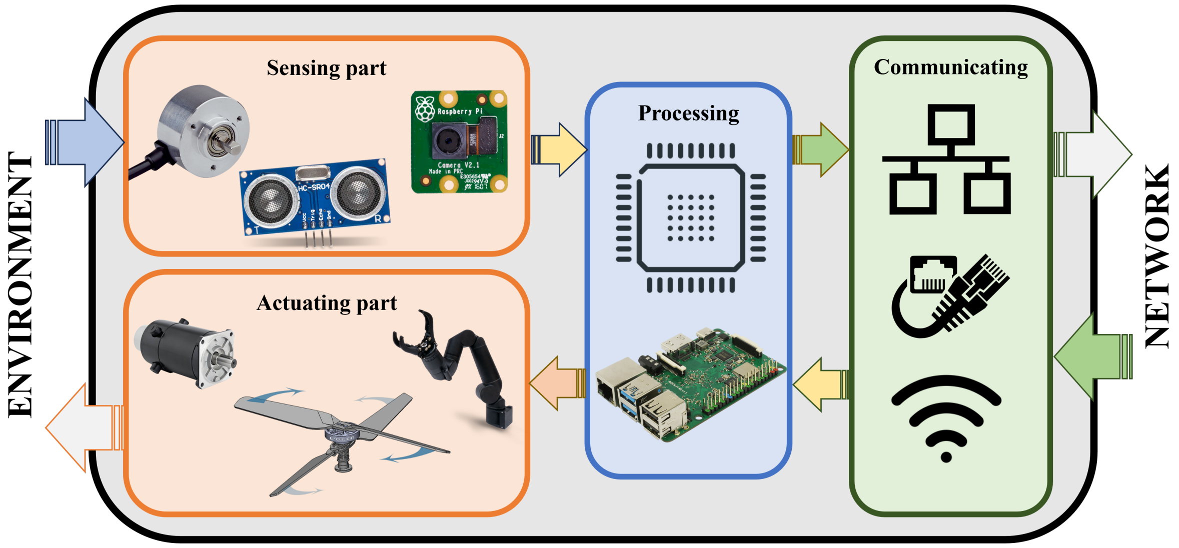


Figure 1.1. Diagram of the different parts of an autonomous robot.

The **sensing part** is a set of measuring devices that allow the robot to acquire information about its environment. With this information the robot adapts to the operating and environmental conditions. Sensors can be classified into two types: internal or external. Internal sensors are those that focus on measuring the robot's variables/states (such as where the robot is, how fast it is going, and how it accelerates, ...). Sensors that help the robot in measuring factors that directly affect its operation such as pressure, current, temperature, acceleration, etc are also considered internal. External sensors detect the effect of an object external to the robot. They collect data from the outside world. Ultrasound sensors, lidar, or contact sensors are some examples of widely used external sensors in obstacle detection tasks. Cameras are also commonly external sensors used for object recognition.

Sensors available on the robot will condition its potential applications. For example, a robot without sensors for obstacle avoidance cannot work collaboratively safely in dynamic environments with other robots, but it can navigate in isolation in structured environments.

The **actuating part** is a set of devices that allow the robot to move and act in the surrounding environment. These devices are oriented toward the fulfillment of the robot's goal. According to the nature of the actuators, autonomous mobile robots can be classified as [2,3]:

- *Wheeled robots.* They are robots that navigate around the ground using wheels to propel themselves. They are one of the most widespread robots due to its simplicity and stability during use. Environmental considerations such as the roughness of the terrain, stability, irregularity, or continuity must be considered in their mechanical design.

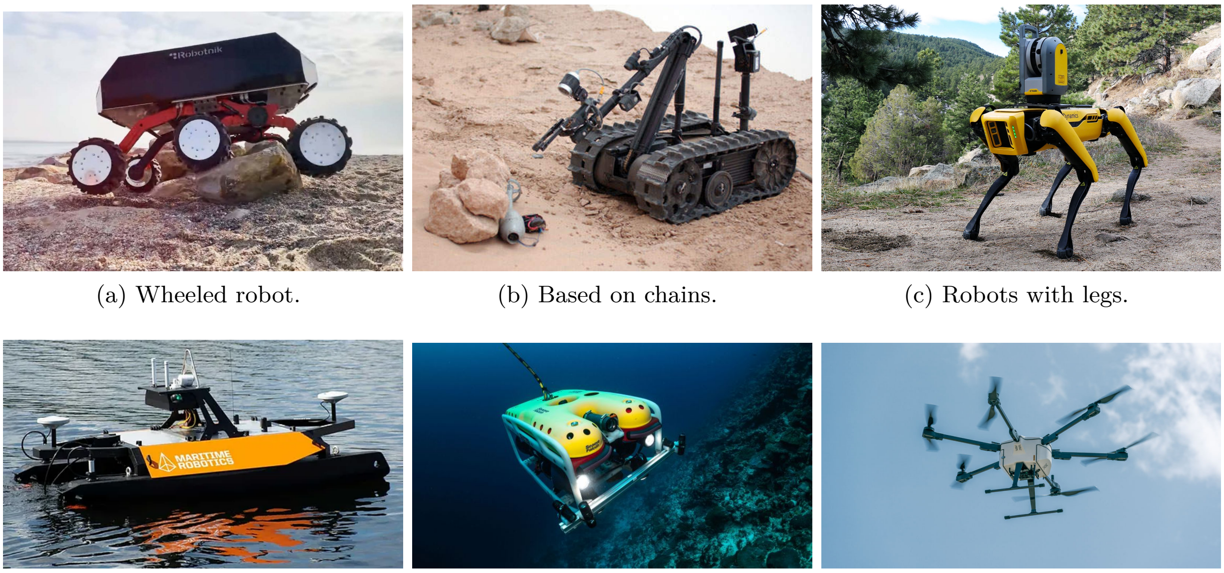
- *Based on chains or belts robots.* They are a modification of wheeled systems designed to operate on soft and uneven terrain, where wheels may slip and lack clearance for proper robot operation.

- *Robots with legs.* This type of robot has the most complex locomotion system. Depending on their design inspiration, they can be anthropomorphic if they replicate the human model or zoomorphic. Their greatest advantage is their ability to operate in very irregular and discontinuous terrain, where other terrestrial mobile robots have more limitations.

- *Marine robots.* This group of robots is made up of those whose environment of displacement is aquatic. They can be classified into surface robots, such as those dedicated to coastal patrol tasks, or underwater robots, which allow, among other uses, oceanographic exploration where other devices cannot access due to the harsh conditions of pressure and risk.

- *Aerial robots.* They are capable of sustained flight with no direct human control. Aerial robots provide unmatched maneuverability, making them the superior choice over ground robots. For instance, they can access areas that are more difficult for ground robots to reach, such as mountainous or rugged areas. Aerial robots can be classified attending to the position of their engines (fixed-wing, multi-rotor, and the self-rotary-winged) or according to their size and weight (Pico Aerial Vehicle (PAV), Nano Aerial Vehicle (NAV), Micro Aerial Vehicle (MAV), Small Unmanned Aerial Vehicle (SUAV) and Unmanned Aerial Vehicle (UAV)) [4], as it is shown in Figure 1.3.

The **communicating part** is a set of devices that serves as instruments in a robot-robot communication process or in a human-robot communication process by allowing the robot to send (or receive) information to (or from) a robot or a human. Mainly, two types of communication can be distinguished: wired and wireless. Each system should implement a solution that offers a compromise between cost, complexity, and transmission efficiency. The choice



(a) Wheeled robot.

(b) Based on chains.

(c) Robots with legs.

(d) Surface marine robot.

(e) Underwater robot.

(f) Multirotor UAV.

Figure 1.2. Classification of autonomous robots according to their method of locomotion.

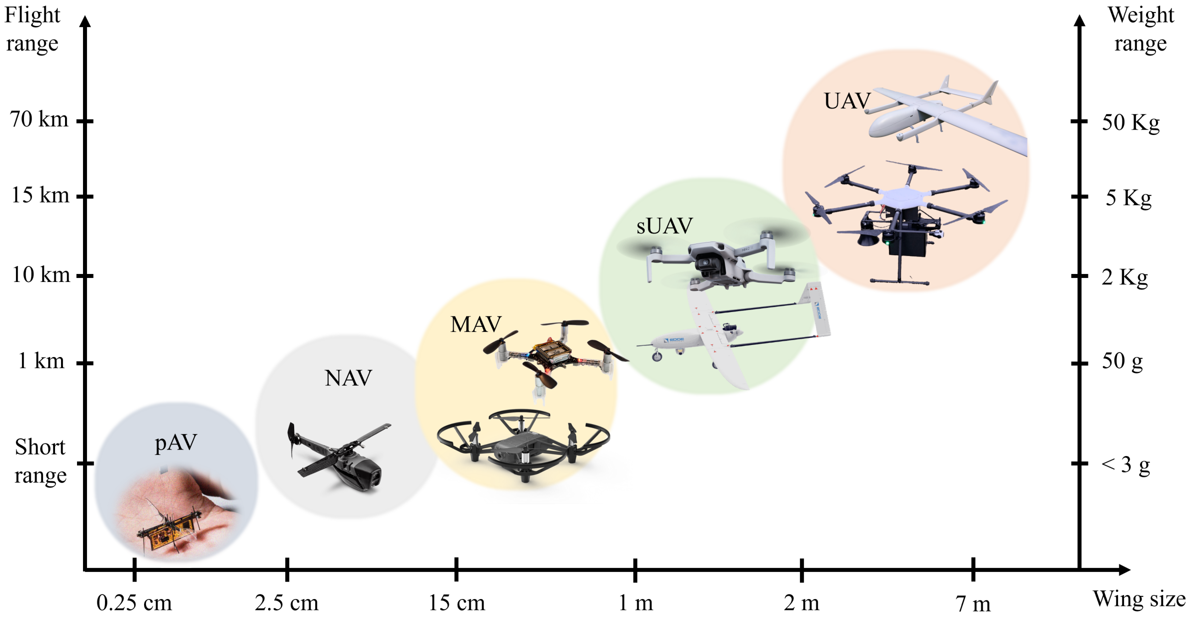


Figure 1.3. Comparison of different types of UAV based on flight range, weight range and wing size.

of the technology, communication architecture, and protocol are relevant aspects that should be considered in the system design. In this regard, three architectures can be distinguished: centralized, descentralized and distributed. Centralized architectures are built around a single network node known as the master that manages information flows. Among the advantages of centralized architectures simple and fast implementation and maintenance are included. However, scalability is limited and generates a risk to security and continuity of service in the case of failure. On the other hand, in decentralized architectures, it is the agents themselves who make their decisions, but without communicating with each other. Finally, in a distributed network computing resource and data are spread across multiple nodes. It provides greater flexibility and scalability to extend the network. However, maintenance costs also increase, and coordination problems can occur in large networks. They also present the challenge of distributing the load evenly, which is not always possible.

The **processing part** is the set of devices that allows the robot to address the information (from sensors or communication with other devices or users in the environment). The capacity of this component determines the complexity of the task that the robot can perform. For instance, systems with microcontrollers with few input and output channels and no wireless communication are an example of embedded systems with a simple architecture and reduced computational capabilities and suitable for low-level control architecture implementation. A step forward in terms of capacity are those systems based on, for example, *Arduino* or *Raspberry Pi* that maintain a reduced size but offer more sophisticated solution allowing sensor fusion algorithms and communication with other devices; or *Jetson Nano* that presents and increased computational capabilities thanks to the use of graphics cards that allows more complex algorithms such as neural networks, images, or point clouds. Finally, robots using more sophisticated systems such as multiple cameras (monocular, RGB-D, stereoscopic) or lidars require more complex equipment, i.e., computers with much higher processing capabilities than the embedded systems mentioned before.

For the correct working of autonomous robots, there should exist a synergy between all the aforementioned elements. In fact, in the last decades, the scientific community has required the development of a new generation of devices that consider the importance of the interconnection between hardware components, software, and communication with other devices in the network. These new systems are known as **Cyber-Physical System (CPS)** [5]. CPSs integrate monitoring, communication, and computation of a process [6]. The use of CPSs aims to increase the deployment of large-scale systems by improving the adaptability, autonomy, efficiency, functionality, reliability, and security of traditional implementations [7]. However, the achievement of these improvements is associated with the emergence of new challenges that should be addressed in their design, such as the emergence of new paradigms about the dispersion of systems [8], the large volume of data that needs to be processed to obtain useful information [9], or the vulnerabilities inherent in the increased exposure of systems through the network to a wide variety of types of attacks [10]. The heterogeneous nature of CPS requires advanced knowledge in several disciplines for their design and construction. Among the sectors with the highest impact of CPSs are the healthcare sector where the correct integration of devices with patients and data confidentiality is crucial [11]; the automotive industry to improve the adaptation to environmental changes, efficiency, and the safety of vehicles [12]; and smart-cities or energy grids where robustness, efficiency, and security are fundamentals to reduce the entry gap to global implementation [13, 14].

The increasing capabilities of CPS has made possible the reduction of size and cost of autonomous systems and hence, the use of more than one of these devices to work cooperatively enhances the complexity of the achievable tasks and widens the fields of applications. Thus, a set of CPS can perform as a **Multi-Agent System (MAS)** [15]. This concept refers to a system composed of autonomous entities (or agents) set that cooperate to achieve a goal. “Agent” means any element responsible for collecting information from the environment and that works autonomously without direct or continuous supervision by a centralized controller [16]. In highly complex tasks, compared to single-agent implementations, MASs offers advantages such as a lower cost, more flexible configurations, and an increased robustness against failures. Indeed, the overperformance of MASs results from breaking down complex tasks into smaller ones and solving them in parallel with different agents [17]. Due to the wide range of applications in which the implementation of a MAS is possible, the most relevant characteristics that allow its classification are: leadership, decision function, heterogeneity, agreement parameters, delays, topology, communication and mobility [15]. The most notable challenges need to be addressed in the literature are coordination between agents [18], fault detection [19], and security issues such as authentication or integrity and confidentiality of communications [20].

Robotics is one of the fields in which the concept of MASs can be applied. If all the agents in the system are robots working cooperatively, the system is called **Multi-Robot System (MRS)** [21]. The interaction between robots influences the collective behavior of the system, and the following relationships between agents can be defined: Indifference, Cooperation and Competition [22]. When the number of agents in an MRS is high, the system is known as a **swarm**, since its design is inspired by associations of individuals observed in nature (e.g. flocks of birds) [23]. The main characteristics of the agents of these systems are:

- Autonomy,
- Interaction with the environment,
- Local sensing and communication capabilities,
- Local control or unavailability of global knowledge to perform a given task [24].

However, experience shows that homogeneous systems have a limited scope of application and flexibility than a MAS that joins a MRS with software agents [25]. This type of system has the advantage of combining the ability to move and interact with the environment of MRSs [26] with a less restrictive computational capacity of software agents, and it is called **Multi-Agent Robotic System (MARS)** [27]. Given their potential, they have been successfully applied to a wide variety of areas [28]. For instance, in computer networks, cutting-edge technologies such as cloud computing [29], the high service demands, and the complexity of social networks [30] or routing services in smart grids are some cases where MAS can reduce the complexity of tasks. Also, in Smart Grids (see Figure 1.4), the application of a MAS framework for the power distribution control algorithm results in a better performance [31].

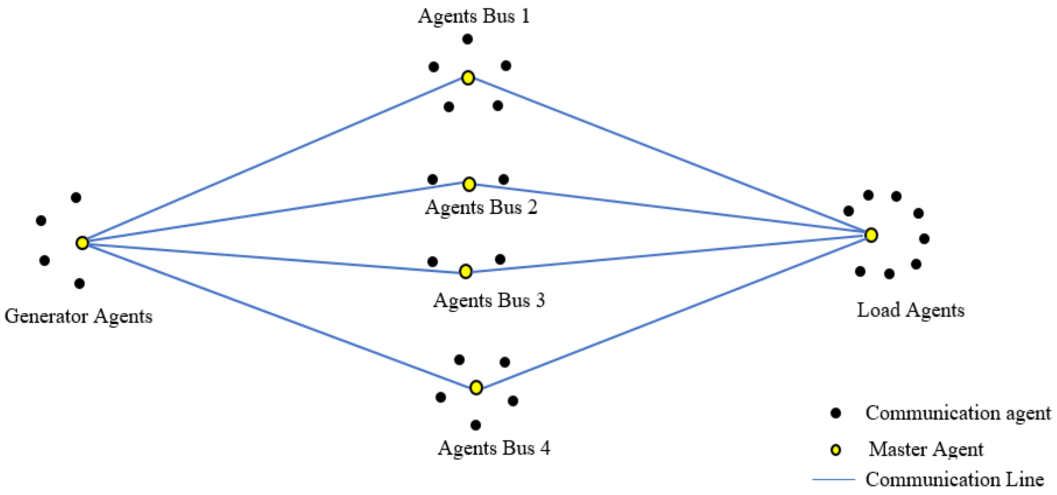


Figure 1.4. MARS applied to Smart grids. Construction of a microgrid bus system-based communication network

In the robotics sector, one of the most prominent applications of MRS includes complex military tasks in escort missions [32], where the tasks can be defined in such a way that enable the robots to execute simple motions to achieve the desired formation behavior. Other applications include localization and mapping of targets [33], security surveillance [34, 35], air-sea scenarios surveillance [36], search and rescue operations [37], urban exploration [38], agriculture [39], space exploration, and communications broadcast among others [40]. For example, in [41] the authors present a detailed development of the design, architecture (both hardware and software), and implementation of a MRS that is specifically designed to collect crop field data faster and in a more efficient way (see Figure 1.5).

1.2 Control of MARS

The full control of MARS is a challenging problem. A MARS may contain a number and type of components that can vary considerably over time, for instance, several heterogeneous robots, diverse sensors, and a certain number of cooperating devices. This fact, together with their wide range of applications, makes the design of its control architecture a very important

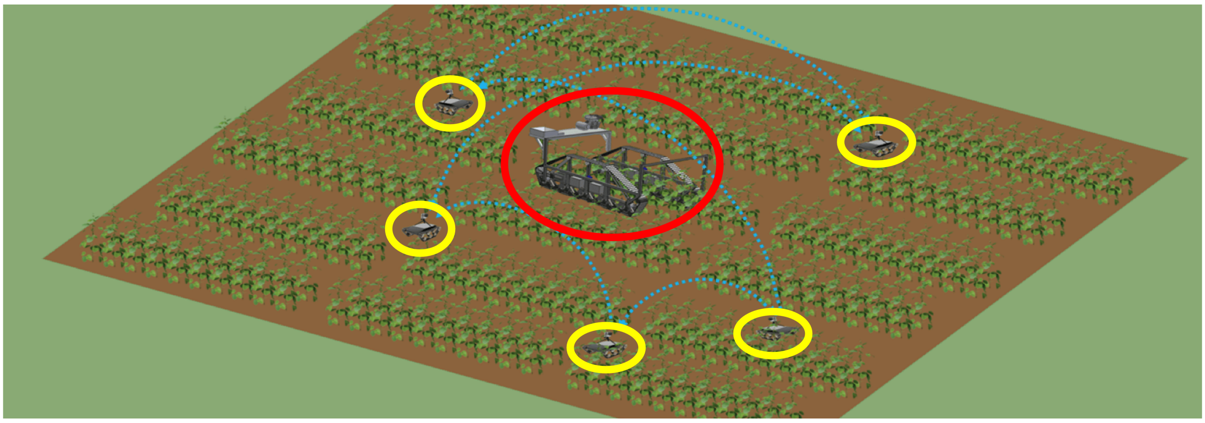


Figure 1.5. MARS applied to Agriculture. Yellow: Smaller lightweight robots equipped with low-resolution sensors that acquire frequent measurements, and infer changes that take place at small-time scales. Red: A mobile platform carrying high-resolution sensors for accurate plant disease detection and analysis of the spread of disease.

decision. A hierarchical control architecture built at different levels provides a good approach as nonhierarchical architectures often face shortcomings in terms of a lack of modularity and scalability. A generic diagram of the different layers into which a MARS control architecture can be segmented and the flow of information between them is shown in Figure 1.6. The four upper levels belong to the control software architecture, whereas the fifth corresponds to the physical layer (hardware) of the system. Decisions about defining the constraints of the task are taken at the higher level and navigation is below; the two intermediate levels are dedicated to the control (individual or coordinated). The low-level deals with everything related to instrumentation, such as multiple sensors/actuators existing on the robots. A description of each layer is given next emphasizing each part in the MARS formation problem, as it is a central part of this thesis.

Task level. Current applications of MAS involve more complex tasks that may not be cast as classic control objectives, but rather involve a higher level of specification definition and planning. For instance, including logical, spatio-temporal specifications or the adaptation to a dynamical environment are different aspects that can be handled at this level. As outputs, it can generate a set of constraints that will be considered in the level below.

Path planning level. Once the constraints are mathematically defined by the upper level, the path planning is in charge of generating the trajectory that the system as a whole should follow. Usually, in multi-agent applications, this desired trajectory is only known by a subset of the agents called leaders. Also, if agents perform perception tasks, changes in the environment can be informed to the task level.

The first two layers, Task and Path planning levels, usually run on CPUs with medium/high computational resources, since the volume of data they operate with (such as cost maps, point clouds, images, etc.) is too large to run on simpler microcontrollers.

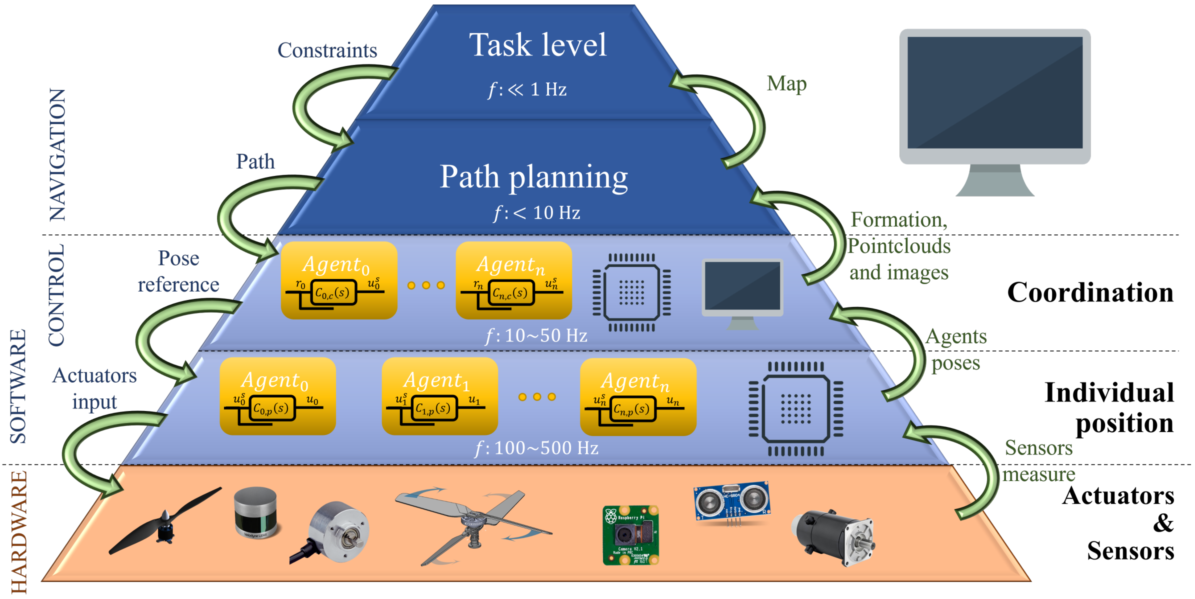


Figure 1.6. Hierarchical control pyramid and information flow in MARS

Coordination level. This level will provide the control law for each agent so that the overall control objective is achieved. Depending on the control architecture, the information at this level can be global (centralized) or local (decentralized and distributed), and can run on a central unit or onboard the agent's microcontroller.

Individual position level. This level is in charge of the stabilization control of the agent. Depending on the complexity of the agent's dynamics, so will be the controller. It is generally implemented onboard as it demands a high frequency execution, especially on agents with fast and unstable dynamics.

Actuators and Sensors level. The lower layer is the MARS hardware level. The input signals of this level are the operating commands of the actuators. These signals are transmitted to the final devices through integrated circuits that adapt the setpoint to the corresponding output, typically Pulse Width Modulation (PWM) signals for electric motors. The raw measurements captured by sensors can have different natures depending on the sensor technology. These measurements include analog voltage, resistance changes, electrical pulses, or PWM frequency variations. Integrated circuits then convert these measurements into digital values for transmission to higher levels.

The coordination layer, regardless of the control law used, allows for different architectures as shown in Figure 1.7, such as centralized, decentralized, or distributed. Each of these implementations has several benefits and drawbacks that make them more suitable for specific applications [42]. All of them must ensure that the group's objective is achieved, maintaining the stability of the formation, and being able to be implemented in real-time in the different agents of the system. In the case of **centralized architectures**, see Figure 1.7a, the advantages include the synchronous management of the control actions of all the MARS controllers and the reduced number of messages transmitted on the network. Among its main drawbacks is the loss of autonomy in the operation of the MARS, since a failure in the computer where the controller is running would cause the whole MARS to stop working, the presence of dominant delays or computational performance problems when the system increase its number of agents.

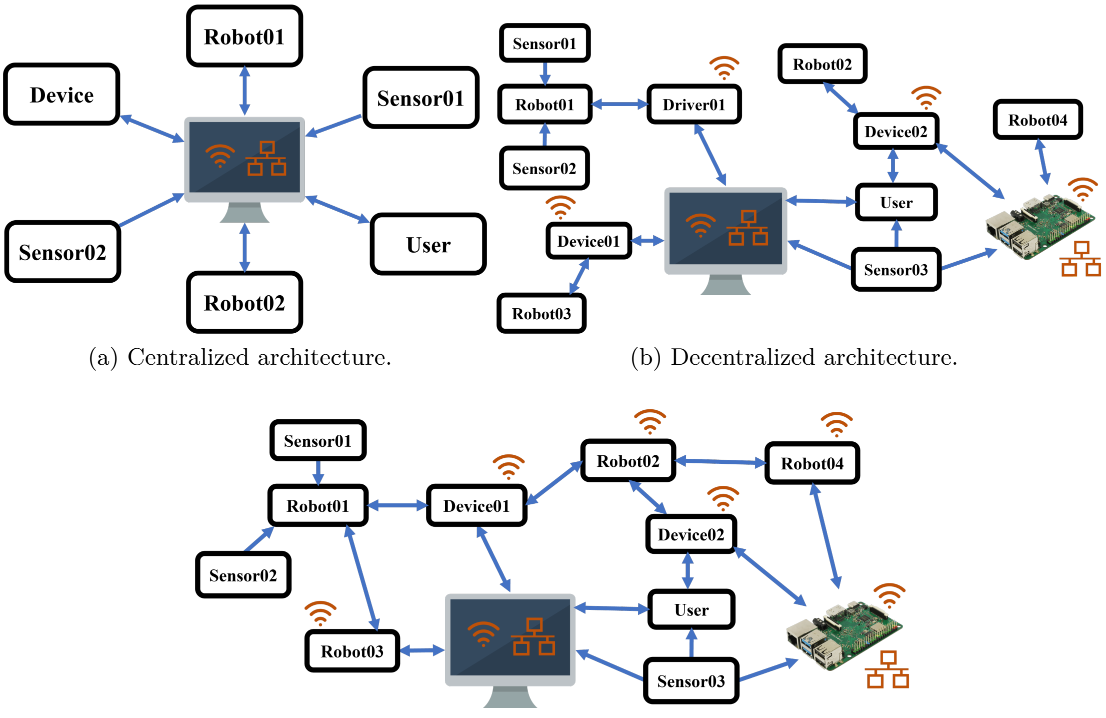
Decentralized architectures, see Figure 1.7b, are characterized by agents making control decisions independently, without the need to communicate with other agents. This approach is more robust to system failures than centralized architectures. In this case, each agent has to communicate with the upper control layers to receive the path and constraint instructions. With this data, at the local level, it generates the instructions for the "Individual position" layer. This implementation is directly related to the sensory capacity of the agents and the way the system's formation is defined. For example, if the agents have equipment to detect obstacles around them and the constraint is set to maintain a distance from the neighboring agents, each robot can move around without the need to communicate with the other agents.

Distributed architectures, Figure 1.7c, address many drawbacks of centralized systems by offering greater flexibility in the network, allowing each agent to implement its own control strategy. In the same way, in case of failure in one controller, the operation of the rest of the agents is not interrupted. On the other hand, the scalability of the system is much more efficient and predictable because the consumption of each controller is the same regardless of number of agents in the network. Whatever the delays introduced by the communication might be, they are generally reduced compared to centralized communication, as the controller needs less time to communicate with all its agents (which are not all the agents in the system as in the centralized case).

Distributed controllers in MARS can be subdivided into two depending on where the controller is implemented: on an external device that connects the robot to the system's communication network, or onboard at the robot's microcontroller. The main advantage of the onboard implementations is the elimination of the delay in communication between the robot and the external device.

Due to its importance in large-scale systems, the concept of scalability requires further comment. **Scalability** refers to the system's ability to add more agents without any negative impact on performance. Adding more agents can improve performance by reducing completion time and simplifying agents. It is essential to examine the following aspects to determine if scaling the system is feasible:

- *Computational cost.* The computational cost of the system depends on the CPU usage of each agent and node. In centralized architectures, all processes are on the same computer, governed by its capacity. However, in distributed architectures, it is crucial to balance the load across the computers involved to prevent bottlenecks that could harm the MARS's overall integrity. It requires avoiding exceeding the capacity of the computers and ensuring



(a) Centralized architecture.

(b) Decentralized architecture.

(c) Distributed architecture.

Figure 1.7. Communication architectures in MARS.

a homogeneous distribution of the load.

- *Network bandwidth.* As the number of agents increases, so does the amount of network transmissions. To predict bandwidth occupancy when there are simultaneous transmissions, it's important to know the sampling and execution frequencies of different processes, as well as the message size. Systems that operate at high frequencies periodically are more likely to have all simultaneous transmissions, making them less favorable. However, event-driven sampling and control strategies can decrease the frequency and number of transmitted messages, which can reduce pressure on the system.

Formation control problem

Among the challenges to address in MARS, one of the most relevant is the control of formation. A formation is understood as a predetermined spatial configuration for a set of agents. A formation can be considered as the tern composed of a set of agents, the interactions between them, and their positions [43]. In [44], the authors characterize the formation control schemes in terms of the agents' detection capacity and interaction topology. The study is carried out taking into account the variables detected and those controlled to maintain the formation. According to [45], they classify the formation control schemes as:

- **Position-based control**. In this approach, agents detect their position in the global reference frame. The variables controlled are their position to achieve the desired formation, which is prescribed by the desired positions in the global frame (either concerning the fixed frame or other agents).
- **Displacement-based control**. In this approach, agents are not required to know their global position, only the orientation of the global reference frame and the relative positions of their neighbors. Each agent actively controls the relative displacements of its neighboring agents to achieve the desired formation, which is specified by the desired displacements with respect to a global coordinate system.
- **Distance-based control**. In this approach, the distances between agents are actively controlled to achieve the desired formation which is given by the desired inter-agent distances. It is assumed that individual agents can detect the relative positions of their neighboring agents concerning their local coordinate systems. These problems can in turn be divided into two approaches: regulation when they maintain the formation and tracking when the movement of the formation along a trajectory is pursued.

When designing formations, there are three main issues that need to be addressed: generation [46], reconfiguration [47], and tracking [48]. In terms of tracking, the agents in the system need to coordinate their movements to achieve several objectives, such as maintaining and acquiring formation, maneuvering the formation, and intercepting or flocking towards a target [49]. There are various formation control strategies available in the literature that can adjust the system's behavior according to the task at hand. Some of the most common strategies include [50]:

- **Behavior-based approach** [51]. In this case, the behavior of each robot is set individually by combining sub-behaviors such as moving towards the goal, avoiding obstacles, or maintaining formation. The main advantage of this approach is that its architecture is parallel and distributed, and the necessary communicated information is less between robots. The main problem is that the mathematical analysis is difficult. In consequence, the convergence of the formation to a desired configuration cannot be guaranteed. In [52], Figure 1.8 demonstrates an example of how this approach is utilized. Each robot has certain predetermined areas with particular behaviors assigned to them. Based on the location of the detected obstacles or neighbor robots, robot "i" will present attraction, repulsion, or indifference behavior.
- **Leader-follower approach** [53–55]. he movement of the formation is determined entirely by the leader's trajectory, and the rest of the agents (the followers) preserve relative distance or positions between them and/or the leaders. This fact implies a simpler individual control based on following a reference with a predetermined margin. However, failures in the leaders might comprise the achievement of the formation.
- **Virtual structure approach** [56,57]. In this approach, the whole formation is considered a rigid body. The motion of the system sets out the overall formation. Then the corresponding

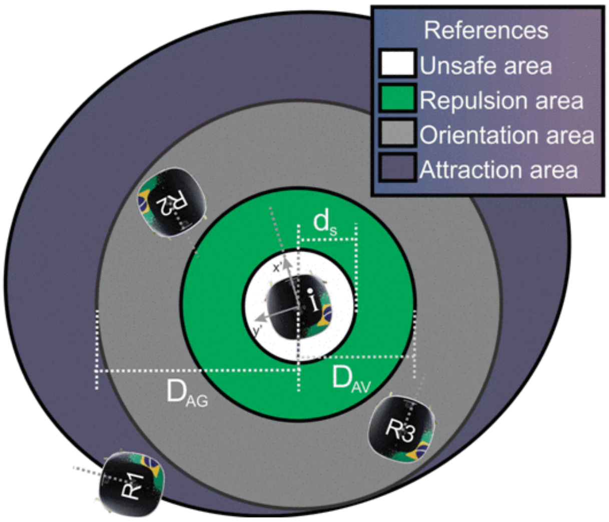


Figure 1.8. Example of Behavior-based approach. Robot "i" with the areas for behavior reunir.

transformation is applied to each agent. The main advantage of this approach is that it is easy to fix the coordinated behavior of the group (including in the maneuvers). The main disadvantage is that the formation must be rigid and therefore loses flexibility and adaptability, especially when the formation shape must change its configuration frequently. In addition, it requires more use of the communication channel between agents. Figure 1.9 depicts the experimental validation of the performance of a formation with three mobile robots for three different trajectories [58].

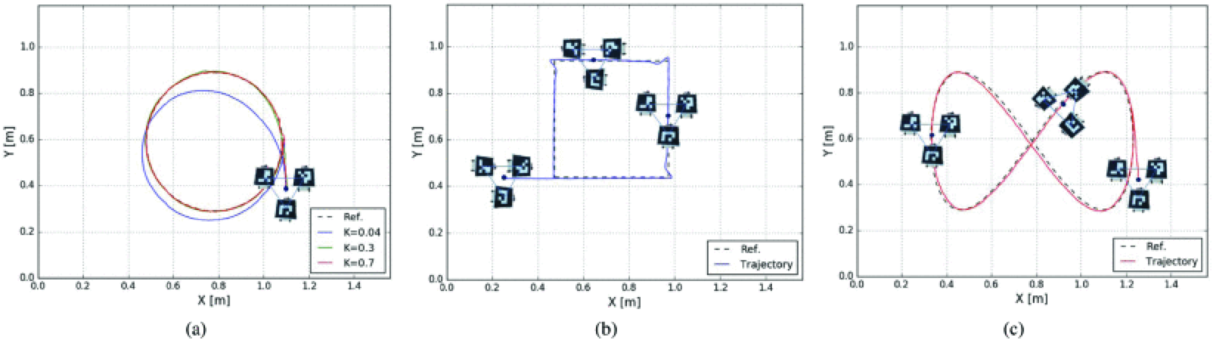


Figure 1.9. Example of virtual structure approach. Results for trajectory tracking (a) Circular, (b) Square, and (c) Double frecuency.

- **Artificial potential functions** [59, 60]. In this approach, the control signal of the robot is derived from a potential function that, on the one hand, drives the system to the desired formation, and on the other hand, avoids collisions between agents. This method has the advantage of requiring fewer calculations and simpler implementation. However, it only ensures local stability.
- **Graph theory based approach** [61]. This approach considers the system as a graph

where the nodes represent the agents or robots, and the edges represent the connections/relationships between them. This approach studies the combined use of graph theory, control, and dynamical systems to analyze formations and their stability. The main advantages of this approach are its ability to model any system regardless of its complexity and the solid theoretical development that exists.

Event-based control

Event-based sampling and control is an alternative to periodic sampling method that has been shown to be effective in reducing the number of samples, updating the control signal, etc [62]. The main idea is that it is the state/the output of the system and not the time what determines when to sample the system. The average rate of event-triggered transmissions is usually much lower than the sampling frequency in sampled-data systems, which implies the superiority of the event-based techniques in resource-constrained applications. However, the sampled-data theory is no longer applicable for the event-triggered control since the fundamental assumption of equidistant sampling is violated, and new theoretical developments have been necessary to derive stability conditions (see [63] and references therein).

Its rise in the last twenty years has been motivated by the advantages it provides in CPS, in which there is a strong coupling between control, computation, and communication. Therefore, the design of strategies must be carried out in an integrated manner to obtain adequate results. In the case of battery-powered systems, such as robots, the autonomy of the agents is a bottleneck in their development. Efficient use of available resources has a particular impact as it allows the extension of the system's autonomy. In MARS, this strategy can improve computational efficiency by implementing it at the individual and coordination control layers, as well as optimizing inter-layer transmissions at the hardware layer.

Given the large number of applications where these systems are implemented, different schemes are required to suit the configuration of the work environment. Figure 1.10 shows the most basic configuration where sampling is performed at the sensor's output in MARS. In

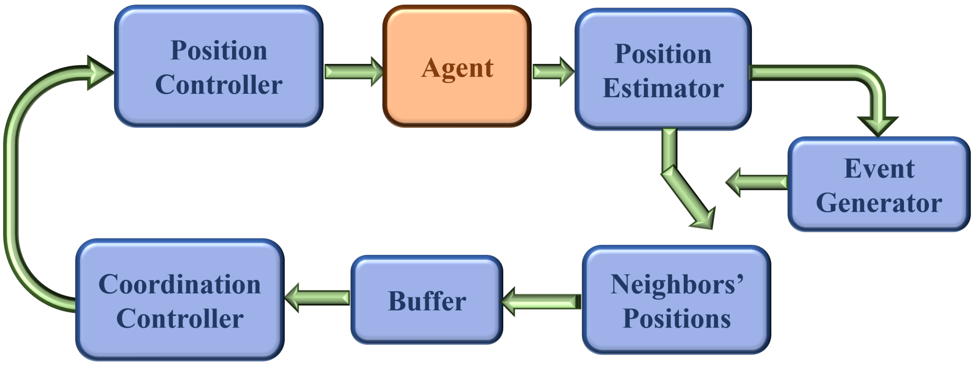


Figure 1.10. Event-based sampling basic diagram.

this case, event-based sampling applied to the position estimator of each agent reduces the use of the communication channel. The event detector is implemented together with the estimator in the hardware layer of the system. The coordination control layer includes a buffer that stores the last position received from the associated neighbors. In this way, the controller can be executed every time new data arrives with the stored values of the rest of the neighbors. Depending on the location of the event detector and the signals involved, the event-based architecture is adapted to the different application domains. In [64] the implementation of a system with an event-based Proportional Integral Derivative (PID) control is describe where events are detected and generated for each of the controller components (proportional, integral, and/or derivative).

Event generation

Usually, at the event-based sampling, a trigger function is defined based on an error signal and a threshold, in such a way that an event (sampling) occurs when the signal reaches that defined maximum value. For instance, if $x(t)$ denotes the state of the system at time t, usually the error is defined as $e(t) = x(t_k) - x(t)$, where t_k represents the last sampling instant. These are defined recursively as:

$$t_{k+1} = inf\{t : t > t_k, f(e(t), x(t)) > 0\} \tag{1.1}$$

where $f(e(t), x(t))$ is the trigger function. There exists in the literature several ways of defining this trigger function. One of the first proposals is to consider a constant threshold, which leads to send-on-delta strategies [65]. They present the advantage of an easy implementation but cannot guarantee asymptotic stability. By contrast, *relative threshold* strategies [66, 67], in which the bound depends on the state of the system, provide such desired property. However, the trigger function has to be carefully designed because, in certain cases, a relative threshold strategy can lead to what is called Zeno effect, that is, the occurrence of infinite events in a finite time interval [68]. To avoid that, a combination of the two mentioned strategies has been proposed [69] or a safety condition, such as a minimum time between events [70], can be used.

In the following, a summary of some of the relevant works in the field is provided:

- *Level crossing or send-on-delta* [71]. This implementation proposes a protocol whereby an event is generated when the analyzed signal crosses a preset value/level. In this case, the working range of the signal is split into equidistant levels. Additionally, hysteresis is implemented so that successive crossings of the same level do not generate events. Another equivalent interpretation that does not require predefining the working range levels is that an event is generated when the difference between the current value of the signal and the last event value exceeds a threshold. This reformulation already incorporates hysteresis. Modifications have been developed over time to obtain more specific results. One example is the *Integrated Level Crossing* [71], which gives a more useful measure of the quality of signal tracking and guarantees the triggering of a future event.

- *Model-Based Triggering* [72]. Classical systems consider the system response as a Zero Order Holder. Significant improvement has been demonstrated by changing this approach to a model-based prediction of the system to be controlled. This technique is especially useful in multi-agent systems where using a prediction based on the neighbours' model can provide a smoother response [73].

- *Sampled-Data-Based Event-Triggered* [74]. The central idea of this strategy is that event detection only occurs at sampling times rather than at continuous times. Furthermore, the minimum number of times between events is inherently limited by a sampling period, which implies that Zeno behavior is excluded. This improves computational efficiency and simplifies the hardware required for continuous sampling.

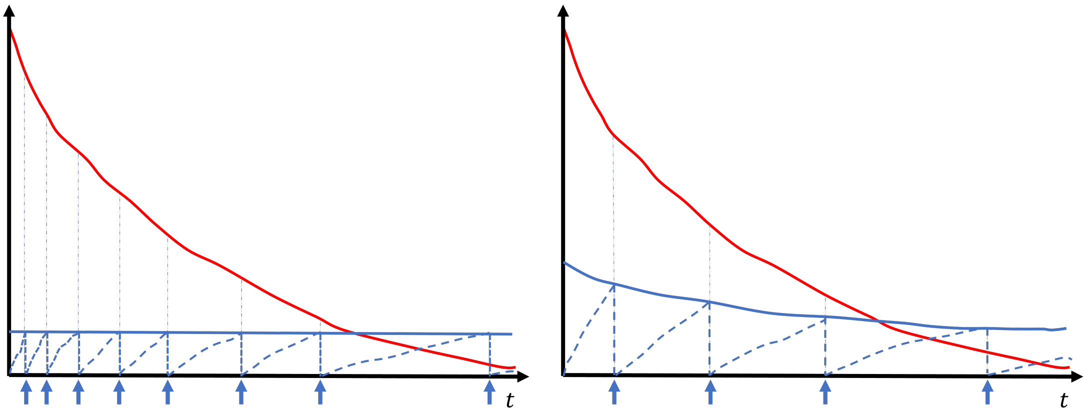
Adaptive threshold

One of the problems that present event-based sampling is that, when the output of the system is subject to noise, events can be generated even though the state might not have changed. To deal with this, adaptive thresholds have been proposed [75] so that the parameters that define the threshold change over time depending on the noise level. More specifically, let us consider the following trigger function:

$$f_i(e_i(t), x_i(t)) = |e_i(t)| - (c_0^i + a_i|x_i(t) - x_{ref}^i| + c_n^i(t)) \tag{1.2}$$

where c_0^i y a_i are constant parameters, $c_n^i(t)$ is a parameter that adjusts to the noise level of the signal and x_{ref}^i is the reference to variable x_i. Note that in this case the threshold consists of three terms: a constant given by c_0^i that will determine the maximum error around the reference, one relative to the distance to the reference weighted by a_i, and a third term given by $c_n^i(t)$ that allows us to adapt the sampling to the presence of noise. This last term is calculated based on the noise estimation, as detailed in [75]. In this way, when the system is far from the reference state, higher values of e_i are allowed. However, when the system approaches the desired equilibrium, the threshold decreases, so that an adequate level of controller performance is guaranteed but reducing the number of control signal updates.

Figure 1.11 shows how different trigger thresholds affect the generation of events (arrows) for a system's response. The red line represents the system response, while the blue line represents different approximations of the trigger threshold. The dashed line shows the deviation of the signal since the last event. Figure 1.11a demonstrates the use of a constant threshold, while Figure 1.11b shows the use of an adaptive threshold. As seen in the second case, the adaptive threshold leads to fewer events generated at the beginning of the event as it adjusts to the system's response.



(a) Constant threshold.

(b) Adaptive threshold.

Figure 1.11. Examples of Event Generation.

1.3 Validation of MARS

MARS is a technology that can be used in various applications where individual agent-based systems are not performing well. It provides better scalability, reliability, and efficiency than traditional systems. However, developers may address several design challenges, such as distributed computing, collaboration, coordination, and real-time integration of all components. Moreover, the infrastructure that supports MARS needs to be designed and configured appropriately, which can be a complex and expensive task. Ensuring that all these aspects are well incorporated can become unmanageable and costly in the long run, especially when it concerns maintenance. These complexities can be reduced by a middleware layer. Middleware provides programming abstractions for a developer so that they can focus on application logic instead of low-level details [76]. Effective middleware for MARS must satisfy the following objectives [77]:

- *Hardware and software abstractions.* This implementation allows developers to focus on high-level work and application requirements. In this way, they avoid dealing with low-level hardware and network issues. It is a good policy to standardize these abstractions as much as possible to enable their use in different applications.

- *Interoperability.* Heterogeneity in MARS can occur due to the difference in hardware (sensors and actuators) or software (communication protocols, libraries, etc.) of multiple agents. Middleware should provide abstractions for developers to work in a homogeneous way ensuring interoperability between heterogeneous agents. That is, it should allow agents to communicate with each other effectively regardless of the platform on which they have been developed or their components.

- *Real-time* support. In many aspects of the robot use, such as control for obstacle avoidance, or teleoperation in medicine, execution time is critical, and real-time systems are required.

- *Elasticity.* The MARS system is designed to be scalable, meaning that agents can be added or removed, and their configurations can be changed as needed. Any growth in the number of messages or entities at any given time should not affect the agents. Moreover, to improve the system's robustness, the middleware should enable autonomous detection and recovery from any network or software failures.

- *Flexibility and reusability of software.* Middleware must allow services to be defined by their functionalities and not according to the hardware, software, or applications for which they are used. It means that a developer does not have to redevelop everything from scratch every time there is a change in hardware or software (libraries, operating systems, languages, the application itself, etc.).

- *Integration with other technologies.* If middleware is developed hermetically, it is hard for it to be reusable when new technologies emerge. In this regard, the integration of standard protocols will facilitate the inclusion of new technologies that allow these systems to be updated. Examples of these recent technologies are the Internet of Things (IoT) or Cloud Computing which use standard protocols such as MQTT or HTTP to facilitate their use.

- Effective *management and monitoring tools* are crucial for any middleware. They should enable developers to monitor and view the entire system, component by component. It helps to understand how the system operates and identify any potential issues. Additionally, such functionality makes the system more user-friendly, allowing even non-expert users to use it. This feature expands the scope of MARS and makes it more accessible to a wider range of users.

All these requirements have been analyzed for as long as MARS has been in use, but the trend in the last decades has been the development of specific systems. These systems show high performance in the applications for which they are designed but fail in some requirements such as integration with other systems, flexibility and reusability or interoperability for new components. One of the most widely used middleware in MAS is JAVA Agent DEvelopment Framework (JADE). It is the general-purpose framework developed by Italian Telecom and the University of Parma. It is the *de facto* solution for the development of multi-agent systems. JADE implements the FIPA standard [78], which promotes interoperability with other platforms. It currently has a high level of maturity, and therefore, it serves a wide community and is well documented. In the MRS case, the two of the middleware layers with the greatest impact have been *Player/Stage System* and *Robotic Operating System (ROS)*. On the one hand, Player/Stage System [79] is a multi-threaded TCP socket server. It provides a transparently accessible network interface. On the other hand, ROS is the middleware *par excellence* in robotics. Communication between devices uses both publisher-subscriber and client-server modes. The system is distributed in *nodes* but they are dependent on a central master node. It implies a significant risk when a failure occurs.

Robotic Operating System 2 (ROS 2)

ROS is a set of open-source software libraries and standard tools that help in building robot applications. Its most recent generation, ROS 2, includes new features that make ROS easier to learn, use and improve its suitability for MARS [80–82]. These improvements enable a more efficient communication protocol with a better real-time performance than ROS, it allows distributed architectures, an adaptation to the most recent language libraries, such as Python 3, and native multi-platform development. This last feature brings its use closer to Mac or Windows users, two systems with a wide presence among non-professional users. The new DDS-based network architecture [83] allows a network to be divided into different domains, independent of each other. In this way, in conjunction with the use of standards in the topics, interoperability between agents is guaranteed.

A network defined in this middleware is composed of *nodes*, *topics*, and *services*. Figure 1.12 shows a conceptual graph of a system based on ROS. Nodes (ellipses) are the processing units of the network with specific purposes, such as reading sensors, controlling motors, etc. Communication between nodes is carried out through topics and services (rectangles). Topics are variables based on the publisher-subscriber model. When a call-and-response-based communication is required, services are used. Complementary, when it is necessary long run a task, the *actions* use is a better option. They start with a service. When they receive the response, they provide steady feedback (as opposed to services that return a single response) and it is possible to cancel them while executing. In the ROS 2 architecture, the network can be split into separate groups known as *namespaces* (dashed rectangles). These subspaces allow users to reuse the same nodes for different agents, supporting the same functionality across multiple areas. It allows for easy scalability of the system without the need for manual adjustments to node, topic, and service names. The RQT software framework, available in ROS/ROS 2, allows for graphical monitoring and management of topics and services.

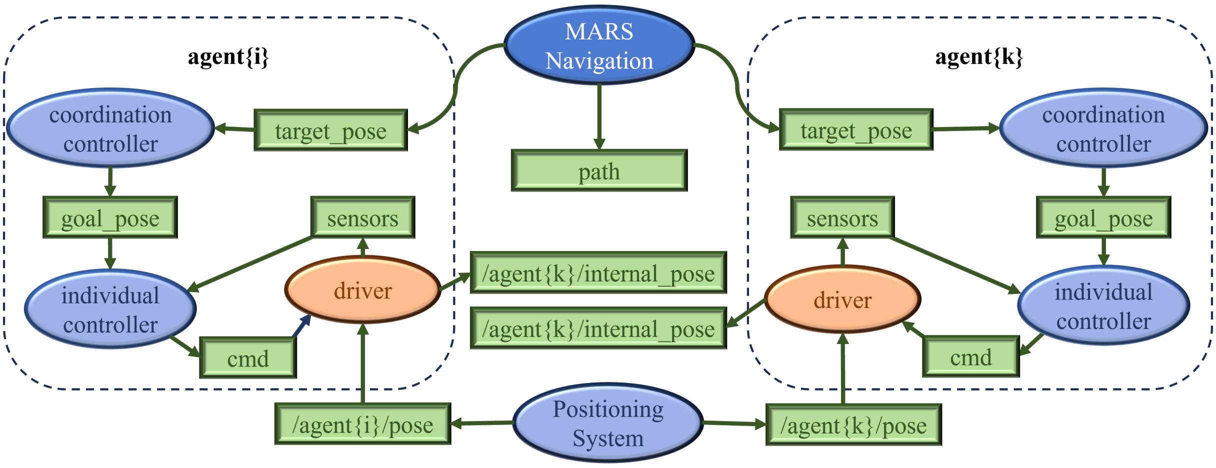


Figure 1.12. Example of ROS diagram.

In recent years, there has been a high demand for ROS/ROS 2, resulting in the development of a new version specific version for embedded systems. *micro Robotic Operating System (uROS)* was developed in 2018 as a result of the rise of distributed systems [84]. This system arose in response to the need to incorporate ROS in a Microcontroller Unit (MCU). These devices are used in almost all robotic products because of their low cost, real-time, low latency, and energy savings. Key features of uROS include seamless integration with ROS 2, resource-limited but flexible middleware, multiple Real-Time Operating System (RTOS) support, and layered and modular architecture. Due to resource constraints, devices that typically require centralized communication architectures with base stations to connect with a ROS 2 network can now function as independent agents.

Considering the features of ROS 2 and the objectives that MARS middleware must satisfy, we can state that ROS 2 is the most complete middleware for heterogeneous distributed MARS that offers the best long-term solution at the current time [85]. Additionally, the development in ROS can benefit from the ROS community developers. This community provides different working groups focusing on specific topics such as education, navigation, drivers, aerial vehicles, etc. If you can spark community interest on your particular project, you can even engage developers over time.

Simulators and physical platforms

Simulators are virtual experimental platforms that allow the emulation of the behavior of physical processes. These tools have proven to be extremely useful in both the academic and research communities. They are highly economical and accessible platforms that do not require a physical space. The only limitation they have is the computing capacity of the system. Due to their delocalization, they are ideal for remote and collaborative work. Different users can work with identical systems under the same conditions, without any time restrictions, and can repeat the experiments as many times as they need without wearing out the system components. These tools are also a fundamental factor in the development, verification, and evaluation of work before it is executed on physical systems.

Like middleware, there is no single powerful tool. Current practice focuses on developing simulators for specific processes (marine robotics, chemical processes, communication networks, etc.). Systems that emulate physical environments have one common feature, the need to develop a working world previously. This virtualization of the environment is essential for the correct working of systems that use sensor data to feedback on their different levels of control. The fidelity with which a physical environment is reproduced determines the quality of the simulator. However, increasing the surface detail of an object improves sensor perception and accuracy, but requires more computational resources to be rendered by the simulators' physics engines. Therefore, in choosing the better simulator, developers should find a compromise between the available computational resources and the accuracy of the implemented model. This graphical visualization also provides important feedback to users, especially non-expert users. For them, the visualization of final data can only be an excessive level of abstraction. Therefore, a correct visualization in real-time of the experiences allows for maintaining the physical sense of the processes, facilitating their comprehension and improving the user-friendliness of the system.

However, simulators are not a panacea for experimental research. When deciding to develop a simulator, researchers and developers need to consider and tackle the following challenges [86]:

- Reality gap. This term refers to the difficulty of transferring simulated experiences into the real world. When designing control software for MARS a difference between reality and simulation models used is always present. There is a trade-off between the accuracy of the models and the computational costs involved in the design process that should be considered

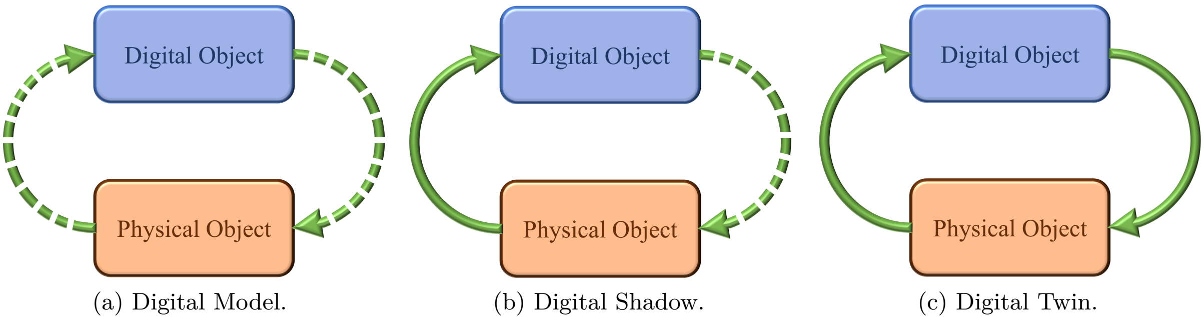
- Computation time and Scalability. Both parameters are closely related. The greater the number of agents involved, the greater the computational time required to perform a simulation step. If users set high specifications for both terms, it can be difficult to achieve without compromising the stability of the physics engine. This problem can be addressed by choosing resolution methods and the simulator step time.

- Users' knowledge. Building models and worlds is often complex and requires a deep understanding of the software. This fact often makes it difficult to use the tools for those who lack this knowledge, especially those who only need to validate their work on the simulator.

As an alternative to these challenges, physical experimental platforms provide a complete and valuable complementary tool to simulators. With this type of platforms, users do not need to develop models for all components of the environment and process under study, thus reducing the initiation gap. Additionally, experimental problems cannot be simplified. A large number of issues must be taken in account, such as irregularities in hardware components, uncertainties in models, limitations in communication channels, interferences, environmental disturbances, and others. Exposing the developed work to these conditions adds great value to the obtained results.

Among the main challenges that developers of these platforms must face are the choice of compatible hardware components and security. At the hardware level, elements from different manufacturers may have different needs (power supply voltages, communication protocols, interferences between systems, etc.). The structural aspects of the environment can impact the functionality of systems. For instance, reflections, limitations in the placement of sensors due to other elements such as electrical systems, heating, ventilation, communications, etc., interference from external signals, legal restrictions, and so on, can all cause problems. From a security perspective, in addition to cybersecurity against external attacks, the physical integrity of the system is also relevant. The platform should guarantee the agents' operation without risk of physical damage. This applies both to the agents themselves (which would present an additional cost) and to the users who could be injured by a system failure or misuse. When operating remotely, the user's risk is eliminated. However, restoring the system to its original state becomes a challenge. If the experiment has been executed well, the system can be configured to return to its initial state upon completion or specific conditions like low battery levels. But in case of a critical failure, the system may not be able to restore itself automatically. It is important to note that these issues are inherent to physical systems since simulators can fix any failure without any consequences by restarting.

The synergy between physical systems and simulators is not new, but new terms have arisen to delimit the different cases in which the two systems are used together. Two digital components available to CPS that are experiencing a boom are *DT* and *Digital Shadow (DS)* [87, 88]. Both DT and DS are technologies that enable the creation of accurate virtual models of robots or processes in simulated environments. The data flow direction between physical and digital systems distinguishes these components from classical Digital Models (DMs). A DM is a computerized replica of a physical object. It is important to note that there is no real-time synchronization between the physical system and the DM (see Figure 1.13a). It means that any changes made to the physical system will not affect the DM once it has been created. In DT and DS, there is real-time communication between the physical system and the digital counterpart. When this communication is bidirectional, the concept that applies is DT (see Figure 1.13c), whereas DS corresponds to the case where the communication is unidirectional from the physical system to the digital system [89] (see Figure 1.13b). Both DT and DS allow the use of real-time data to perform parameter estimation, predictive controllers, fault detection, etc. on the physical system. The exploitation of twins, shadows, and DMs in the framework of CPS gives rise to a wide range of challenges and cases in the field of robotics and control [90].



(a) Digital Model.

(b) Digital Shadow.

(c) Digital Twin.

Figure 1.13. Levels of integration according to data flow (arrow). Solid: Automatic; Dashed: Manual.

Firstly, the DT models' development requires a detailed knowledge of the physical system and an infrastructure that supports the communication and integration needs between subsystems. These two factors will determine the extent and features of the DT's coverage. If highly accurate models are available and computational and communication speeds are adequate, more challenging tasks such as parameter estimation or controller add-ons may be used. However, for applications where accurate models are not necessary or where technical limitations are a factor, supervisory applications such as preventive fault detection or safety backup in the event of a system failure are more appropriate.

Another fundamental aspect directly related to the exposure of systems to a more connected world is cyber security. The systems that use DTs and DSs require data that can be trusted from the point of view of integrity and confidentiality. Data injection attacks can cause controllers to become unstable or erratic, which can result in physical system damage and adverse economic consequences.

Finally, one of the most recurrent problems hindering its implementation is the lack of standards. As it is a technology developed from classical simulation tools and there is no consensus on its definition, developers have opted for ad hoc designs for their applications. This makes integration with other technologies difficult and forces own development, which leads to an entry gap in terms of the required skills.

In the field of robotics, Augmented Reality (AR) is a relatively new technology that has significantly contributed to the integration of physical and digital environments [91]. Unlike Virtual Reality (VR) which generates representations of the physical world in the virtual world, AR is focused on collaborative robotics that allows digital information to be displayed on physical objects. Mixed Reality (MR) is a combination of VR and AR [92], widely used in collaborative systems [93], which allows interaction and manipulation of both physical and digital elements and environments. This interaction is completely bidirectional, meaning that the behavior of DMs affects the behavior of physical systems and vice versa.

Among the most extended applications of this concept is the teleoperation of systems by DT. In these cases, users operate a virtual environment (DT) that transmits the commands to the physical system. The user receives feedback from the physical system via the DT through periodic communication. MR provides greater flexibility to its systems, facilitating the combination of physical and virtual elements to perform new experiences. The challenge for systems that incorporate this approach is the development of a software infrastructure to run the virtual elements in real-time and to communicate seamlessly with the physical elements. This aspect is covered by the middleware's features. The main benefits of middleware include:

- Spatial flexibility. The integration of physical and virtual environments enables the remote operation of robots. It is particularly useful in collaborative settings where the participants are in different geographical locations. Two physical facilities can have simultaneous joint experiences in a shared centralized virtual environment.
- Elimination of safety risks. One of the main aspects to be addressed in collaborative work is physical safety between elements. In the case of human-robot interaction, the risk of damage due to a malfunction of any part can have serious consequences. MR offers a safer, lower-risk experimental space where pre-validations can be performed using virtual robots with physical humans or real robots with virtual humans.

- Simplified debugging. MR creates a rich environment where it is possible to monitor in detail a virtual system in a physical environment to detect early failures.
- Sensory flexibility in the virtual environment. Sometimes, the characteristics of the experimental agents do not allow having much equipment on-board (e.g. reduced size vehicles such as MAV). In these cases, if a faithful virtual replica of the physical environment is available, these agents can be equipped with additional virtual sensors such as lidars or cameras without affecting the physical characteristics of the agent.
- Scalability. In cases where the physical space has limitations to operate a large numbers of agents, the use of virtual spaces and agents allows increasing the size of the MARS.

1.4 Outlines and Contributions

The different contributions of this thesis are applications of advanced identification and control techniques in Heterogeneous MARS. A summary of the most relevant ones and the thesis' structure follows below:

- **Chapter 2. Robotic Park (RP)**. Chapter 2 describes the developed physical experimental platform for indoor MARS. The architecture and all the components are described, including the different types of robots (aerial, mobile), positioning systems (MoCap, UltraWide Band (UWB), Infrared, based on Vision), and communication architectures (centralized, distributed, or hybrid). The system is integrated in ROS 2 that interconnects all the elements. Several simulators have been developed and are integrated in the platform. This work has been published in part in the IEEE Access Journal (see [94]).
- **Chapter 3 Differential Drive Mobile Robots**. This chapter presents a detailed study of the theoretical modeling of mobile robots. Then, these results are validated experimentally on real robots. It also explains the individual control architectures of them.
- **Chapter 4 Aerial Robots**. This chapter focuses on the theoretical modeling of aerial. The control architecture, consisting of several levels, is also explained. The effectiveness of using event-based techniques for reducing controller executions in microcontrollers is demonstrated. This work was presented in part in two conferences: the XLII Jornadas de Automática [95] and the 13th IFAC Symposium on Advances in Control Education [96].
- **Chapter 5 Formation Control: Experimental Evaluation**. This chapter studies the formation control problem, as it is one of the fundamentals of cooperative tasks in MARS. First, a control law to drive the system to a formation in the 3D space but restricted to constraints is presented. Then, different experiences are presented to study the performance of Robotic Park in the context of multi-agent systems. More specifically, the scalability, the control architecture, and the communication protocol are evaluated. With respect to

scalability, different experiences are carried out to analyze the impact of an increasing number of agents (both virtual and real) on CPU usage and system performance. The behavior of two of the developed simulators is compared. After that, three different architectures (centralized, distributed in ROS 2, and distributed onboard) are evaluated jointly with the communication protocol. This latter case studies the impact of the communication frequency and compares the results with an event-based protocol in which the position of the robots is only broadcasted in the network when the trigger condition is satisfied. The study of the scalability of the platform has been published in [97], whereas the impact of the architecture and the protocol can be found in [98]. A Preliminary study that only evaluated the platform with aerial robots was presented in the XLIV Jornadas de Automática [99].

- **Chapter 6**. Finally, the conclusions and future research steps related are given.

1.5 Publications and projects

Journal papers

1. F.J. Mañas-Álvarez, M. Guinaldo, R. Dormido, S. Dormido. Robotic Park. Multi-Agent Platform for Teaching Control and Robotics. *IEEE Access*, 11: 34899-34911, 2023

2. F.J. Mañas-Álvarez, M. Guinaldo, R. Dormido, S. Dormido-Canto. Scalability of Cyber-Physical Systems with Real and Virtual Robots in ROS 2. *Sensors*, 23(13): 6073, 2023.

3. F.J. Mañas-Álvarez, M. Guinaldo, R. Dormido, S. Dormido. Muestreo y comunicación: impacto en el control de formaciones en sistemas multi-robot heterogéneos. *Revista Iberoamericana de Automática e Informática industrial.*

Conference papers

1. F.J. Mañas-Álvarez, M. Guinaldo, R. Dormido, R. Socas, S. Dormido. Control basado en eventos mediante umbral relativo aplicado al control de altitud de cuadricópteros Crazyflie 2.1. *XLII Jornadas de Automática*, September 2021, Castellón, Spain.

2. F.J. Mañas-Álvarez, M. Guinaldo, R. Dormido, R. Socas, S. Dormido. Control de formacion basado en eventos para Crazyflie 2.1. *I Simposio Conjunto de los Grupos Temáticos de CEA: Modelado, Simulación, Optimización e Ingeniería de Control*, April 2022, Burgos, Spain.

3. R. Socas, R. Dormido, M. Guinaldo, F.J. Mañas-Álvarez, S. Dormido. Indirect Method for Calibrating Quadrotor Sensors: A Case Study applied to the Crazyflie 2. X. *IEEE IAS Global Conference on Emerging Technologies* (GlobConET), May 2022, Arad, Romania.

4. F.J. Mañas-Álvarez, R. Dormido, M. Guinaldo, R. Socas, S. Dormido. A Vision Based Navigation Platform for Control Learning. *13th IFAC Symposium on Advances in Control Education* (ACE2022), July 2022, Hamburg, Germany.

5. F.J. Mañas-Álvarez, M. Guinaldo, R. Dormido, R. Socas, S. Dormido. Formation by Consensus in Heterogeneous Robotic Swarms with Twins-in-the-Loop. *ROBOT2022: Fifth Iberian Robotics Conference* (ROBOT2022), November 2022, Zaragoza, Spain.

6. F.J. Mañas-Álvarez, M. Guinaldo, R. Dormido, R. Socas, S. Dormido. Control de formacion basado en eventos para Crazyflie 2.1. *II Simposio Conjunto de los Grupos Temáticos de CEA: Modelado, Simulación, Optimización e Ingeniería de Control*, April 2023, Madrid, Spain.

7. F.J. Mañas-Álvarez, M. Guinaldo, R. Dormido, S. Dormido. Análisis de la frecuencia de muestreo en sistemas multi-robot. *XLIV Jornadas de Automática*, September 2023, Zaragoza, Spain.

Other publications

1. A. Hoyo, F.J. Mañas-Álvarez, E. Rodríguez-Miranda, J.D. Gil, M. Castilla, J.L. Guzmán. Bringing Automatics and Robotics closer to pre-university students. *13th IFAC Symposium on Advances in Control Education* (ACE2022), July 2022, Hamburg, Germany.

2. A. López-Gázquez, F.J. Mañas-Alvarez, J.C. Moreno, F. Cañadas-Aránega, J.A. Sánchez. Navigation of a Differential Robot for Transporting Tasks in Mediterranean Greenhouses. *GreenSys2023: International Symposium on New Technologies for Sustainable Greenhouse Systems*, October 2023, Cancun, Mexico.

3. J.L. Blanco-Claraco, B. Tymchenko, F.J. Mañas-Álvarez, F. Cañadas-Aránega, Á. López-Gázquez, J.C. Moreno. MultiVehicle Simulator (MVSim): Lightweight dynamics simulator for multiagents and mobile robotics research. *SoftwareX*, 23: 101443, 2023.

4. M. Guinaldo, J. Sanchez-Moreno, F.J. Mañas-Álvarez, S. Zaragoza. Distributed multi-UAV shield formation based on virtual surface constraints. *Robotics and Autonomous Systems (submitted)*

Research projects

The results obtained in the framework of this dissertation have been supported by different research projects:

1. *ECoDiC: diseño Eficiente y Control Distribuido de sistemas Ciber-físicos* (2019-2020). Agencia Estatal de Investigación (AEI) CICYT (Ref. RTI2018-094665-B-I00). Participants: UNED (Spain). Directed by Prof. Raquel Dormido Canto.

2. *RECOVERY: REsilient and secure COntrol of cooperatiVE cybeR-phYsical systems* (2021-2024). Agencia Estatal de Investigación (AEI) (Ref. PID2020-112658RB-I00). Participants: UNED (Spain). Directed by Prof. María Guinaldo Losada and Prof. José Sánchez Moreno.

Chapter 2

Robotic Park

2.1 Introduction

The growing interest in research communities and industries in MARS is largely due to its potential real-world applications. However, one of the major problems of MARS lies in real-world performance evaluation. That is, how to validate and evaluate different approaches comprehensively. In this sense, the demand for flexible, robust, and precise MARS testbeds increases.

Depending on their nature, the experimental platforms or testbeds that do exit for testing MARS developments can be classified into three types: physical, virtual (simulators), or hybrid. On the one hand, **physical platforms** allow a real-world multi-robot system operation in a physical environment for realistic performance evaluation. On the other hand, virtual platforms allow for determining the experiment's feasibility and to define the robustness of the novel algorithms in a variety of complex interaction scenarios, although it often does not work under real-world conditions. Finally, using hybrid platforms allows the use of Hardware-in-the-Loop, Software-in-the-Loop, and Mixed Reality (MR) architectures [100] combining real and virtual agents.

Currently, most of the distributed MARS results are still only validated in simulated environments and therefore it is unknown how these methods work in real-world scenarios. In this way, it is essential to build new experimental platforms, or testbeds, to carry out MARS development validations, beyond simulations, both from a research and teaching point of view.

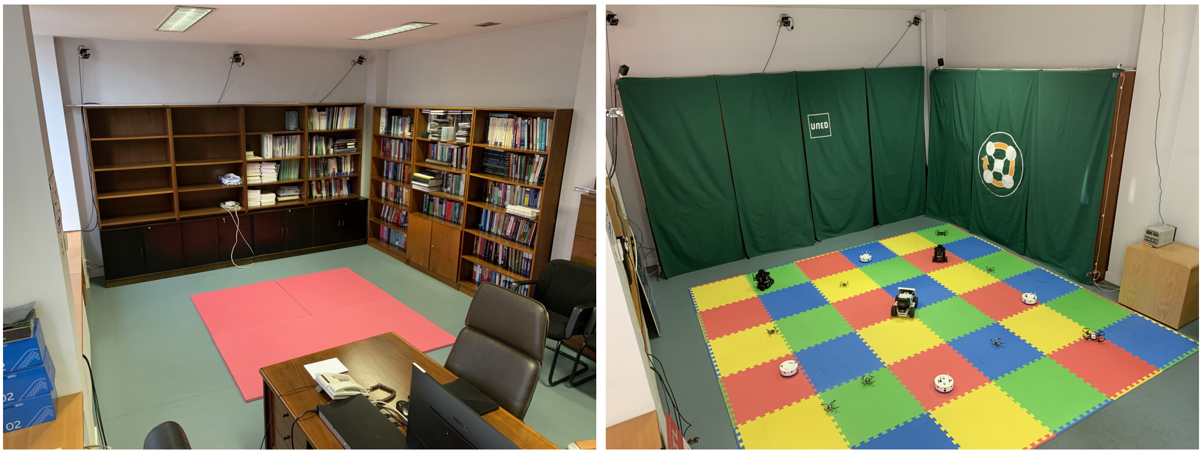
Motivated by this need, the experimental platform Robotic Park (RP) has been developed as a central part of the work of this thesis. The overall solution perfectly covers our research interest for experimentation tasks with MARS, involving restrictions in the computation time and data transmission, and also our teaching purposes in the Department of Computer Science and Automatic Control of the Universidad Nacional de Educación a Distancia (UNED).

Building a MARS indoor experimental platform such as Robotic Park is more than challenging. It involves many different issues: the design and choice of hardware components of the system, the robots' selection, the system architecture and communication, the positioning systems, or the simulation software selection from those available. On one hand, at the hardware level, a well-designed platform needs to include different components (PCs, switches, etc) that ensure the overall operation of the system. Moreover, the choice of the agents directly affects the types of experiences affordable, so it is important to consider features like their mechanical, size, mass, or traction method. At the software level, the choice of the communication architecture (see Figure 1.7) and the associated technologies are fundamental in enabling interoperability among heterogeneous agents in RP operations to ensure the achievement of shared missions. Moreover, the selection of the proper simulation tools must be accomplished to get the best seamless integration of simulation environments and the real-world platform. Finally, the estimation of the position of the robots in the motion area is also an important question to address in the design process of RP. How to achieve high indoor positioning accuracy [101] is the premise in the development to ensure that the agents can complete the proposed task. The decision about what IPS to use, either internal or external to the robot, and the mainstream indoor positioning technologies applied (such as Wi-Fi, Radio Frequency Identification (RFID), etc) will be defined. This decision will be largely determined by the type of robot and the communication function available for each one. Anyway, to verify the feasibility of the positioning methods experiments in both the RP real environment and in simulation will be analyzed. Throughout this chapter an insight into each of these aspects and how RP integrates all of them will be provided.

In this regard, **Robotic Park (RP)**, a heterogeneous experimental indoor platform for control and robotics. It is designed to support MARS experiences combining different types of vehicles: micro-aerial quadcopters and differential mobile robots. The architecture implemented allows users to work in a virtual environment (simulation of the system), in a physical environment (robots in a physical environment), or in a hybrid scheme. The virtual environment includes different simulators, based on Gazebo [102], Webots [103], and Matlab. Gazebo is a popular open-source 3D robotics simulator with broad compatibility with ROS 2 and supports most sensors. However, its environment can be challenging for those unfamiliar with it. As a solution, RP also includes Webots and Matlab, which are easier to use and have a friendly user interface but with a tougher integration in ROS 2.

Furthermore in the physical environment of RP, three positioning systems based on different technologies are available: Lighthouse Positioning System, LPS, and Vicon's Motion Capture (MoCap). All elements are connected through a ROS 2 network. This system is more suitable when implementing decentralized architectures. Moreover, it also facilitates MR experiences.

RP is located in the 6.13 laboratory of the Escuela Técnica Superior de Ingeniería Informática of the UNED in Madrid, Spain. Figure 2.1 shows the two versions that have been worked with in RP. Initially, the laboratory working volume was of $2 \times 2 \times 2\ m$, and the only operational positioning system was Vicon's MoCap (see Figure 2.1a). Currently, the working volume has been increased to $3.8 \times 3.8 \times 3\ m$, and the other positioning systems added (see Figure 2.1b). In this new configuration, it is possible to work with a greater number of agents.



(a) Initial version.

(b) Current version.

Figure 2.1. Robotic Park laboratory.

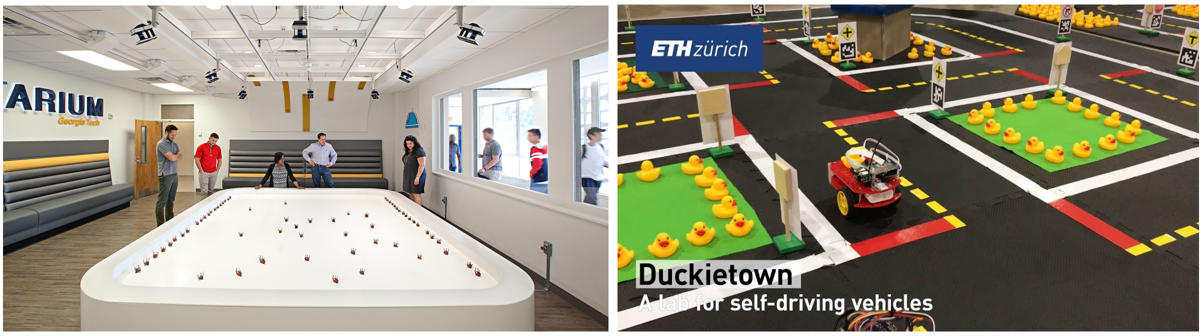
The rest of the chapter is structured as follows. Section 2.2 overviews the most relevant experimental platforms covering RP-like needs. Section 2.3 reflects the fundamental characteristics that show the potential of RP versus other similar platforms. The technical description of robots available in RP is available in section 2.4. In addition, the interest in its use is justified compared to other alternatives. Section 2.5 is a detailed description of the RP architecture. The section ends with a concise definition of experimental availability, definition of configuration files and subsequent data processing. Section 2.6 details the communication infrastructure available in RP and the use of ROS 2 as middleware. Section 2.7 shows the available positioning systems in RP, with particular emphasis on their characteristics, infrastructure, and integration in RP. Section 2.8 explains the main simulation tools available in RP, which are fully integrated and justified for use in different areas of RP exploitation. Finally, section 2.9 outlines the key findings from the development and implementation of RP and proposes future work to keep RP at the forefront of experimental platforms.

2.2 Overview of platforms

This section shows a compilation of some of the most relevant experimental platforms that use MARS and MRS.

- *Robotarium* [104–106], see Figure 2.2a. The Georgia Institute of Technology developed this platform. It began operating publicly 2017. It is one of the most complete platforms available today, focused on MRS applications using mobile robots like the GRITSBot [107]. The platform has a 3.65 × 4.25 $[m]$ arena equipped with a Vicon MoCap system, composed of 8 cameras. At the software level, it is designed using simulators for Matlab and Python, as well as a docker to help users become familiar with the environment. To allow external users to run experiments, the developers provide configuration files and a web system to manage submissions. This way, managers can run requested experiments and return logged results.

- *Duckietown* [108,109], see Figure 2.2b. This platform was created in 2016 by the Massachusetts Institute of Technology (MIT) with the goal of making teaching and research more accessible through simpler systems. The mobile robot equipment in this system relies solely on monocular cameras as feedback sensors. The goal is to find efficient solutions using simple robots, which limits the types of tasks the system can performe. It mainly focuses on Artificial Intelligence (AI) and vision tasks, such as image segmentation for recognizing traffic signs.



(a) Robotarium.

(b) DuckieTown.

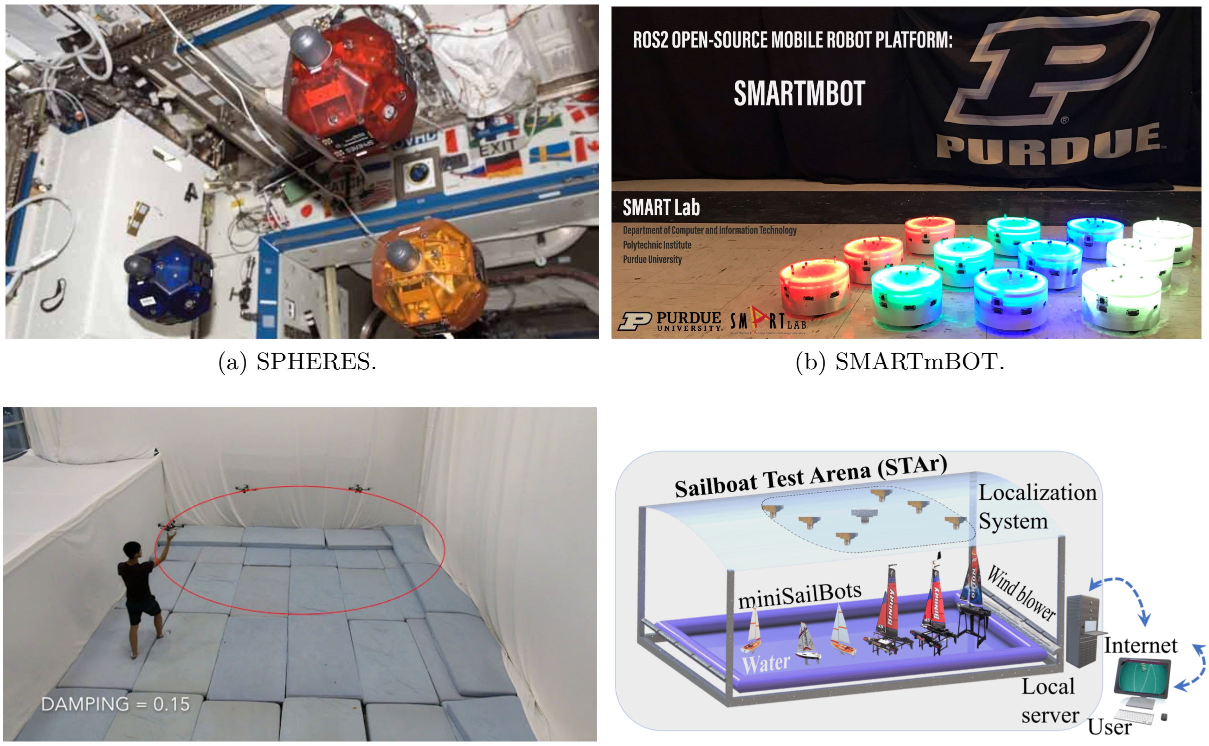
Figure 2.2. MARS testbeds (I).

- Synchronized Position Hold Engage Reorient Experimental Satellites (SPHERES) [110], see Figure 2.3a. It is a project developed at the MIT Space Systems Laboratory created to provide support to institutions such as the National Aeronautics and Space Administration (NASA). The platform has three operating environments, consisting of a simulation software, a 2D ground laboratory, and a 3D laboratory on the International Space Station (ISS). The objective of the platform is to investigate MRS satellite agent, with special emphasis on formation flight and docking algorithms in microgravity. Since the beginning of the project in 2006 several research groups have been working on the platform.

- SMARTmBOT [111], see Figure 2.3b. This testbed is a 2021 open-source project of the Smart Machine and Assistive Robotics Technolog (SMART) Lab, at the Department of Computer and Information Technology, Purdue University, Indiana, USA. This platform is focused on low-cost MRS studies. The robots are developers' designs and can be built with 3D printers. It is implemented in ROS 2 to improve its flexibility, scalability, and modularity.

- Flying Machine Arena (FMA) [112, 113], see Figure 2.3c. From 2007 to 2020, *ETH Zürich* developed and operated this experimental platform that was dedicated to aerial robotics. Its main feature is its large working volume ($10 \times 10 \times 10$ m) with a high-precision and high-frequency positioning system that allowed the analysis of the agents' behavior of various sizes under aggressive dynamics (high velocities and accelerations).

- Sailboat Test Arena (STAr) [114, 115], see Figure 2.3d. This testbed is a remotely accessible indoor platform for sailing robot design verification, validation of autonomous algorithms, and sailing control practices. It was launched in 2016 and developed by the State Joint Engineering Lab on Robotics and Intelligent Manufacturing, at the Chinese University of Hong Kong, Shenzhen. The arena is equipped with a fan system to generate stable wind fields. The positioning system implemented for the agents is of the MoCap type. Among its main strengths are that it is a compact platform with scalable results, low cost, free to use, and remotely accessible.



(a) SPHERES.

(b) SMARTmBOT.

(c) Flying Machine Arena.

(d) Sailboat Test Arena diagram.

Figure 2.3. MARS testbeds (II).

- MRComm [116]. In 2019 the Department of Informatics at King's College London, UK, created a platform called MRComm. Its main objective is to analyze inter-robot communications, stability, consistency, and faults in MRSs. Despite its concise purpose, the proposed architecture for analyzing these critical aspects has garnered significant interest.

- LOGISWARM [117]. This testbed is a low-cost platform developed at Sri Ramaswamy Memorial Institute of Science and Technology (SRMIST), Kattankulathur, India, in 2022. Its work focuses on the control of MRS in cooperative transportation tasks and analyzingits behavior for different payload form factors, robot configurations, and applications.

- Mini-sized Mobile Robots [58]. This platform was developed in 2020 by the Department of Automation and Industrial Control at the Faculty of Electrical and Electronic Engineering in Escuela Politécnica Nacional, located in Quito, Ecuador. Its main primary feature is its ability to operate very small mobile robots using a vision-based positioning system. This makes it a cost-effective solution compared to more advanced systems and easy to install without requiring a large space.

- Urban Multi Wireless Broadband and IoT Testing for Local Authority and Industrial Applications (UMBRELLA) [118]. This testbed is part of a large-scale project related to IoT and smart cities research. It is a real-world IoT testbed deployed in Bristol, UK, by Bristol Research and Innovation lab. Toshiba Europe Ltd. It is located in the South Gloucestershire region in the UK. The aim is to deepen the integration of multiple systems and sensors in large-scale IoT systems in real environments. In the field of robotics, the application focuses on industrial warehouse management and research into the booming DTs technology. In [119] the authors show a study carried out on the platform.

In recent years, the number of testbeds dedicated to MRS has been increasing. Other examples of testbeds dedicated to MRS are the following: Simulating Collaborative Robots in a Massive Multi-agent Game Environment (SCRIMMAGE) developed by Georgia Tech Research Institute [120]; DOTS developed by Bristol Robotics Laboratory [121]; MARBLER (an improvement of Robotarium) [122], and CrazyChoir from the OPT4SMART project [123]. Appendix A summarize the main information of each platform with the URL addresses of those available.

2.3 Robotic Park. Main features

Robotic Park is an indoor platform that is highly flexible, easy to use, and heterogeneous. Its development has a twofold motivation: the validation of research over an experimental system, and its use as an automatic control lab in graduate and postgraduate courses. The selection of components - both hardware and software - has been made to ensure the highest degree of accessibility and compatibility. The main features of Robotic Park are summed up to:

- *Heterogeneity.* Whereas most existing testbeds usually work with homogeneous agents, RP enables the use of heterogeneous agents. Experiences using different types of aerial and ground vehicles equipped with sensors of different natures, types of locomotion and communication modules are enabled. Additionally, several IPS, based on different technologies, are integrated into RP. This allows a comparison of the performance provided by the different IPS, to adapt to the robots or experience needs, or a combination of more than one to achieve improved performances.

- *Flexibility.* It supports virtual, physical, or hybrid scheme experiences. Most of MARS algorithms are only validated through simulations. RP offers the possibility of performing these experiments in physical or hybrid environments, which allows a higher number of agents in the system. This feature also opens the possibility of collaboration with users in different geographical locations, and to parallelize the realization of experiences.

- *Interoperability.* RP includes two hybrid frameworks (Gazebo and Webots) that enable combining real and virtual agents in MR experiences, as the DT are indistinguishable from the real agents. The architecture implemented in the simulators is perfectly compatible and replicable in the physical environment, with a seamless integration of simulation environment and real-world platform. This fact allows MARS MR experiences regardless of the agents' nature (virtual or real).
- *Elasticity.* All components are integrated through ROS 2 which allows centralized, decentralized, or distributed control and communication architectures for MARS, since ROS 2 allows running distributed systems more easily. Additionally, because each agent is an encapsulated entity, they can be added or deleted (with no changes in the existing infrastructure), i.e., the size of the MARS can be dynamic.
- *Facilitates Experimental Design.* The use of ROS 2 in RP also allows experiment designs to be identical for simulation and real experimentation on hardware. The use of the physical or virtual environment only depends on the nodes or parameters that are executed, whereas the nomenclature and typology of the variables are the same in both environments.
- *Modularity.* The design of control strategies for MARS involves different aspects. Firstly, the local control that allows the maneuvering of the agent. Secondly, the cooperative control that requires collaboration between agents. And finally, a high-level task planning. Thanks to the RP architecture, the validations of one of these techniques do not interfere with the others. Additionally, the transmission of information from one control level to the others as well as between different nodes is implemented in such a way that changing the communication policy (periodic or based on events) is simple and transparent to the user.

A detailed description of the different parts that make up Robotic Park is given in the following sections. Figure 2.4 schematically summarizes the different components that integrate the platform and how they are related to each other. It reflects the role of ROS 2 as a link between all components despite its heterogeneity: physical robots, data storage systems, simulation software, etc. All code developed for the testbed's implementation is hosted in repositories under an organization on GitHub [1].

2.4 Robots in Robotic Park

Robotic Park operates indoors, meaning agents must have architectures that allow them to work together effectively in small spaces. It has been considered that small-sized differential mobile

[1] Github: `https://github.com/Robotic-Park-Lab`

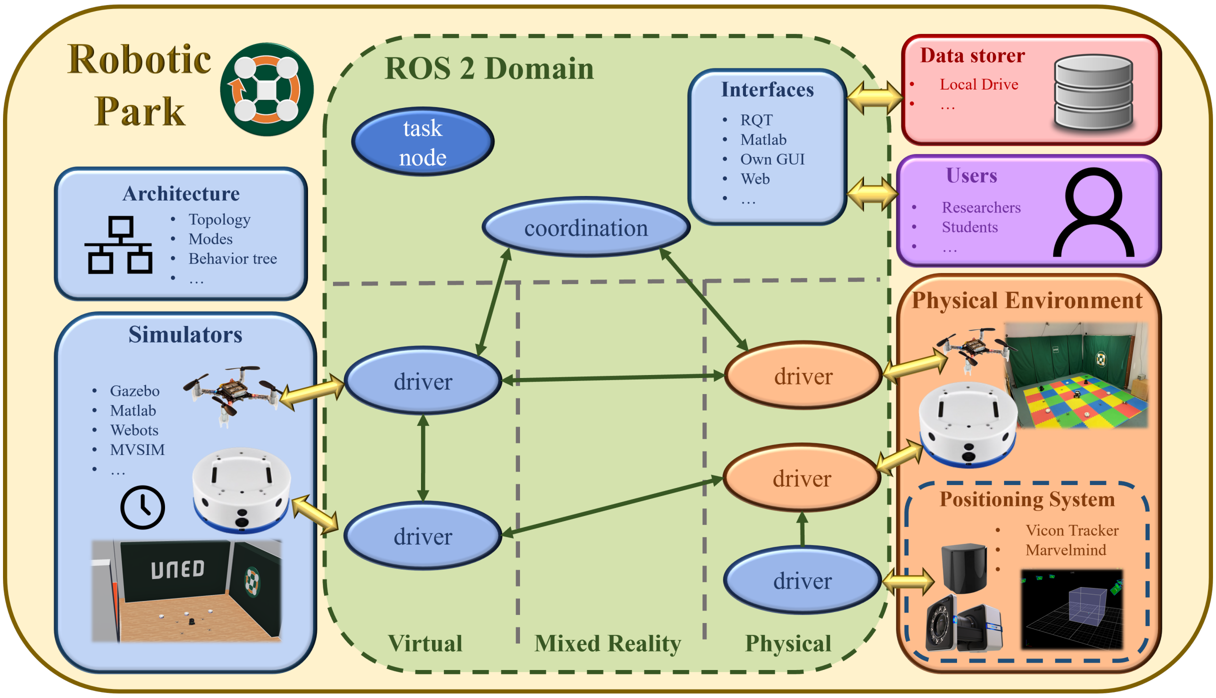
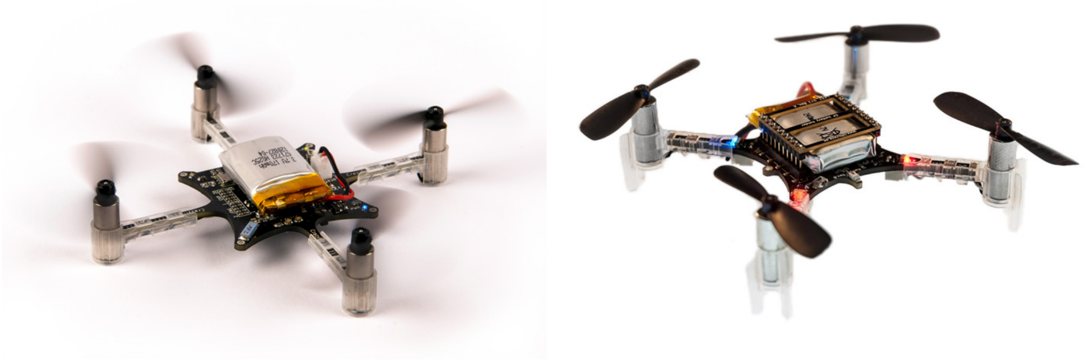


Figure 2.4. Components of Robotic Park.

robots and MAVs are suitable to carry out experiences in RP, as they are the most widespread robots in indoor workspaces for research and teaching. Currently, two types of aerial vehicles (Crazyflie 2.1 and DJI Tello) and two types of differential mobile robots (Turtlebot3 and Khepera IV) are available. The Crazyflie 2.1 and the Turtlebot3 are open-source platforms with official documentation available on their official websites and a partial development of ROS 2 drivers have been developed using their Software Development Kit (SDK). In the following, the main features of the robots' hardware are presented.

2.4.1 Crazyflie 2.1

Crazyflie is a versatile open-source flight development platform from *Bitcraze AB* [124]. In 2009 the Crazyflie project was part of the Swedish consulting company *Epsilon AB*. In 2011, the project grew to launch *Bitcraze AB*. Crazyflie is a four-propeller aerial vehicle (quadcopter) of small size. Two generations of the Crazyflie are available, Crazyflie v 1.0 and Crazyflie v2.X (see Figure 2.5). Crazyflie 2.1 is the latest version. The most significant hardware change between both generations is the position of the rotors, using a "+" configuration in the first, see Figure 2.5a, and a "×" configuration in the second, see Figure 2.5b. At the sensory level, the first generation is based on 6 degrees of freedom (DOF) Inertial Measurement Unit (IMU), MPU-6050 which provides acceleration data and orientation and angular velocity. The second generation, is based on 6 DOF IMU, BMI088, in combination with a high-precision pressure sensor (BMP388).



(a) Crazyflie v1.0, “+”.

(b) Crazyflie v2.X, “×”.

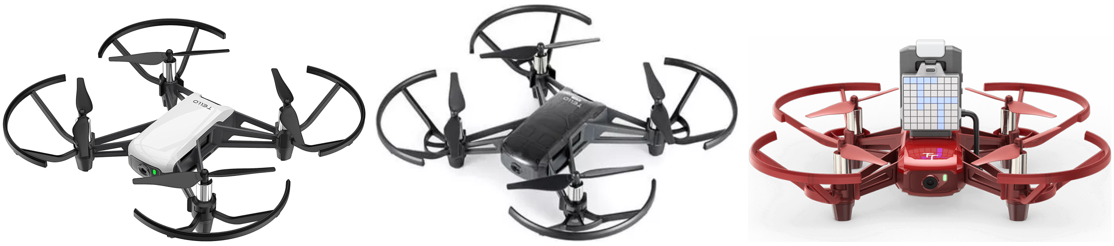
Figure 2.5. 1^{st} and 2^{nd} generations of the quadcopter Crazyflie.

Regarding its physical characteristics, it is a MAV with dimensions $92 \times 92 \times 29$ $[mm]$ and weight 27 $[g]$. Its flight time is about 7 $[min]$. It has a STM32F405 microcontroller (Cortex-M4, 168MHz, 192kb SRAM, 1Mb flash) and communication can be carried out via Bluetooth and radio with nRF51822 (Cortex-M0, 32Mhz, 16kb SRAM, 128kb flash). Its basic configuration incorporates the minimum sensors for its correct operation. To increase its sensory capacity, expansion decks are used to attach to its connection pins. The most relevant are the multi-ranger deck that equips five VL53L1x ToF that is capable of measuring distances up to 4 $[m]$ and the AI deck that increases the computational and integration capabilities of the Crazyflie through the ultra-low power 8+1-core GAP8 RISC-V IoT application processor, a ultra-low-power 320×320 grayscale camera (Himax HM01B0) and an ESP32 module to add WiFi connection (NINA-W102). A prototyping deck to include proprietary sensors is available to allow circuit design by users. It is worth noting that in addition to the software available to control the Crazyflie individually and with open Python-based tools, it has support to operate as MRS through Crazyswarm [125] and its Crazyswarm2 version integrated into ROS 2. All these features make it a robot with great success among researchers.

2.4.2 DJI Tello

The DJI Tello quadcopters are a commercial proposal more focused on entertainment than on research [126]. It is a closed commercial platform with limited programming capacity through an SDK that allows the control of the robot. Due to its great boom in recent years, different versions have been launched, and without leaving the entertainment objective, they have expanded their functionalities for research, such as swarm control (Tello EDU) or programming through the Python language (RoboMaster Tello Talent), see Figure 2.6. Among its main advantages are its robustness in navigation and the quality of its cameras. However, its main drawback is the latency in communication and the impossibility of modifying its firmware.

Regarding its physical characteristics, it is a MAV with dimensions $98 \times 92.5 \times 41$ $[mm]$ and a weight 87 $[g]$. Its flight time ranges from 13 to $30[min]$. The drone equipment includes a rangefinder, a barometer, a LED, a vision system, a 2.4 GHz 802.11n Wi-Fi, and live camera display with 720p resolution.

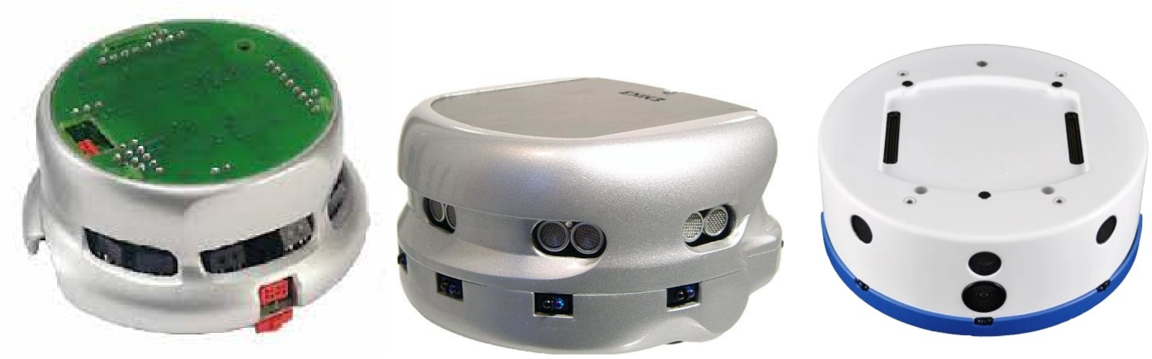


(a) Tello. (b) Tello EDU. (c) RoboMaster Tello Talent.

Figure 2.6. quadcopters DJI Tello.

2.4.3 Khepera IV

This robot is a commercial differential-drive robot developed by *K-Team* and launched in January 2015 [127]. It is the fourth generation of this robot model, see Figure 2.7. It also incorporates an 800 MHz ARM Cortex-A8 processor with C64x fixed-point DSP core and an additional microcontroller for peripheral management. Moreover, it includes a Yocto 1.8 Linux kernel that prevents the installation in the repositories of packages not previously compiled for this system.



(a) Khepera II. (b) Khepera III. (c) Khepera IV.

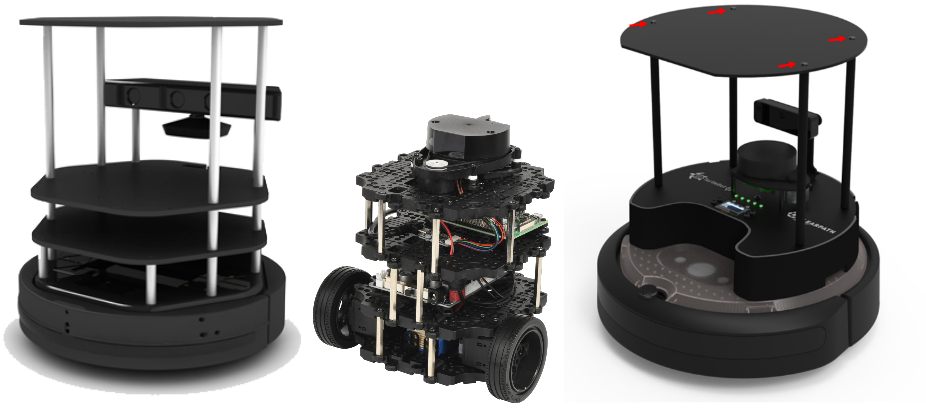
Figure 2.7. Generations of Khepera robots.

Regarding its physical characteristics, it is a cylindrical robot 140 $[mm]$ in diameter and 58 $[mm]$ in height, with a weight of 540 $[g]$ and a maximum capacity of 2000 $[g]$. The battery life for average resource use is 7 $[h]$. In terms of equipment, in addition to the KB-250 expansion bus to include external equipment, it has 8 infrared proximity and ambient light sensors with a range of up to 25 $[cm]$ (TCRT5000, Vishay Telefunken), 4 infrared ground proximity sensors for line tracking and fall prevention applications (TCRT5000, Vishay Telefunken), 3-axis accelerometer and 3-axis gyroscope (LSM330DLC, STMicroelectronics), Built-in microphone ($100-10000$ [Hz] PU0414HR5H-SB, Knowles), 0.7 [W] speaker (400-20 000 [Hz]), Built-in color camera (752×480 pixels, 30FPS, Focal length of 2.1 [mm], Field of view: 131º horizontal, 101º vertical) and 2 Faulhaber 1717 DC brushed motors with incremental encoders (approximately 147 pulses per mm of robot motion). As communication channels, it has 1 USB 2.0 host (500 [mA]), 1 USB 2.0 device, Wi-Fi 802.11 b/g, and Bluetooth 2.0 EDR. These features make it a very useful robot for indoor research.

However, being a 2016 model without significant updates, it does not offer the versatility and compatibility that would be desired in multi-agent environments. For integration into the ROS 2-based laboratory system, we have performed a WiFi communication using User Datagram Protocol (UDP) with a client-server protocol with an intermediary node in one of the laboratory's operational base stations.

2.4.4 Turtlebot3 Burger

The *Turtlebot* series of robot emerged in *Willow Garage* in parallel to other open-source platforms such as ROS and OpenCV. *TurtleBot1* was developed in 2010 by Tully (Platform Manager at Open Robotics) and Melonee (Fetch Robotics CEO) of Willow Garage on iRobot's Roomba-based robot, although commercialization begins in 2011. In 2012, the *TurtleBot2* developed by Yujin Robot appeared based on the iClebo Kobuki research robot, see Figure 2.8a. In 2017, the *TurtleBot3* developed by ROBOTIS' Co [128] emerged with new features to complement functions not included in previous versions and user demands, see Figure 2.8b. In May 2022 *TurtleBot4* developed by *Clearpath* was released with an exterior design more similar to *TurtleBot1*, as shown in Figure 2.8c. The goal of *TurtleBot3* is to drastically reduce the size of the



(a) Tutlebot 2. (b) Tutlebot 3 Burger. (c) Tutlebot 4.

Figure 2.8. Generations of Turtlebot robots.

platform and lower the price without having to sacrifice functionality and quality while offering expandability. As it is an open-source platform, all the code and design are available on the web and the manufacturer opens the possibility of making designs and programming modifications to the developer community. RP has opted for this robot because it has a flexible open-source proposal on which to test your own software and hardware configurations.

It is one of the most used open-source wheeled mobile robot for robotics teaching specifically designed to operate in indoor environments with ROS/ROS 2. Its main advantage over the Khepera IV is its greater configuration flexibility. Its main CPU is a Raspberry Pi 4 with 2Gb of RAM, and this allows it to operate as an independent agent within the ROS 2 network. Its dimensions are $138 \times 178 \times 192$ $[mm]$. Its weight including sensors is 1 $[kg]$ and its maximum payload is 15 $[kg]$. Its sensors in the basic configuration are a 360 Laser Distance Sensor LDS-01 or LDS-02 (Detection distance 0.12 $[m]$ to 3.5 $[m]$) and an IMU with 3 axis gyroscope and accelerometer. The battery life for average resource use is 2.5 $[h]$.

2.5 Robotic Park. System Architecture

RP is a versatile platform that is designed to be flexible and modular throughout its lifetime. The core of its architecture is based on these principles. The platform's architecture is **highly flexible**, enabling it to be easily and seamlessly switched between a centralized or distributed topology at both the hardware and software levels. At the hardware level, the user can set the architecture by selecting the physical computing platform on which each component runs. For example, a centralized hardware architecture is used when everything runs on a single main PC in the lab, as shown in Figure 2.9. However, when the computational load of the experiments needs to be distributed, the user can run different components on different PCs or Raspberry Pi. These units just need to be ROS 2 compatible and connected to the same network. This allows users to run lower levels of control on the fixed infrastructure of the platform, while higher levels such as task resolution or coordination can be run on their own PCs. Figure 2.10 shows an example where the global positioning system runs on the host computer, the individual communication and control of a set of robots on a Raspberry Pi 4, and the graphical interface and coordination control of the MARS on the user computer.

The fundamental idea behind **modularity** is that changing a system component can be done via software without modifying the configuration of the rest of the components. In this way, the user can replace default system components (parameter estimators, controllers, drivers, agents, etc.) with his developments without breaking the rest of the elements as long as he complies with the platform's rules. These rules are established to ensure that all users maintain a standard, that there are no conflicts between components, and that the monitoring and automatic data processing tools are compatible with all experiences.

As part of this modularity, RP has three operation modes: virtual, physical, or hybrid. These

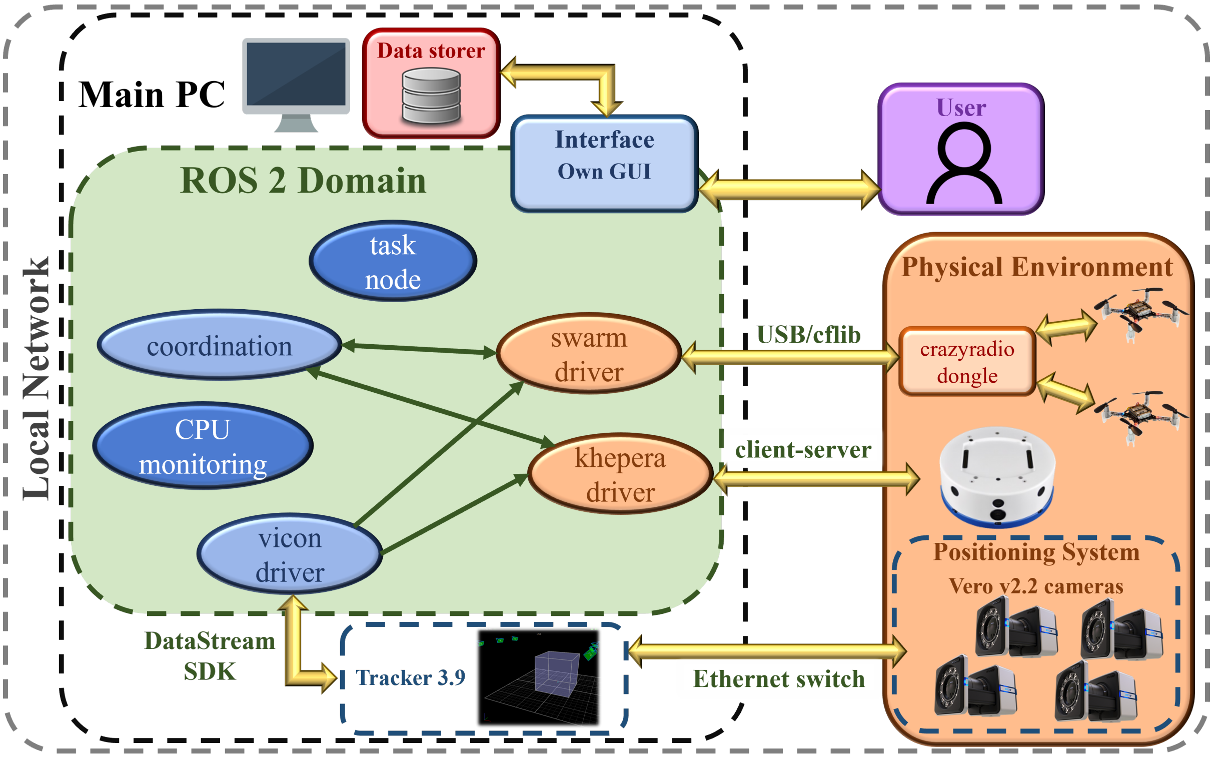


Figure 2.9. Example of centralized architecture in Robotic Park.

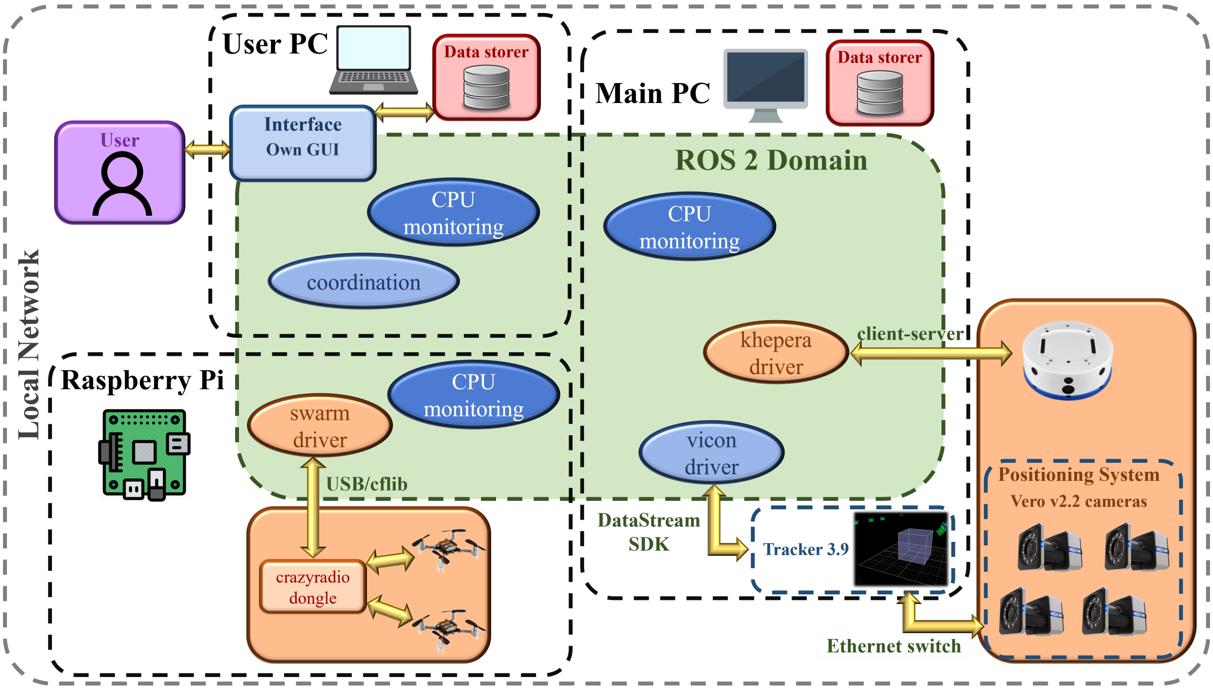


Figure 2.10. Example of distributed architecture in Robotic Park.

three modes are compatible with each other, i.e. developments carried out in simulation do not need additional configurations to perform the same work in the physical environment. This is thanks to the level of abstraction provided by ROS 2 as system middleware. The virtual mode is characterized by the fact that it does not use physical agents. The behaviour of these virtual agents must be generated by digital environments such as simulators. The simulators replicate the physical agents' components defined internally in the firmware. Among them, the most relevant are the internal position controllers and sensors. The system's software architecture is designed for the simultaneous execution of the simulator and the rest of the platform's software modules. The most relevant factor in this mode of operation is the consumption of computational resources by the simulation tool. The main parameter that allows adjusting this term is the simulation step time. If its value is increased, the required resources are reduced but the accuracy of the generated behaviour worsens. To ensure synchronism of all software modules, all share the simulated clock generated by the simulation tool. Figure 2.11 shows an example of a formation experience that only uses a virtual environment. Specifically, a single crazyflie and two kheperas IV are running in Webots. They establish communication between the simulator and ROS 2 driver node of each robot using the "webots_ros2_driver" library of Cybernetics.

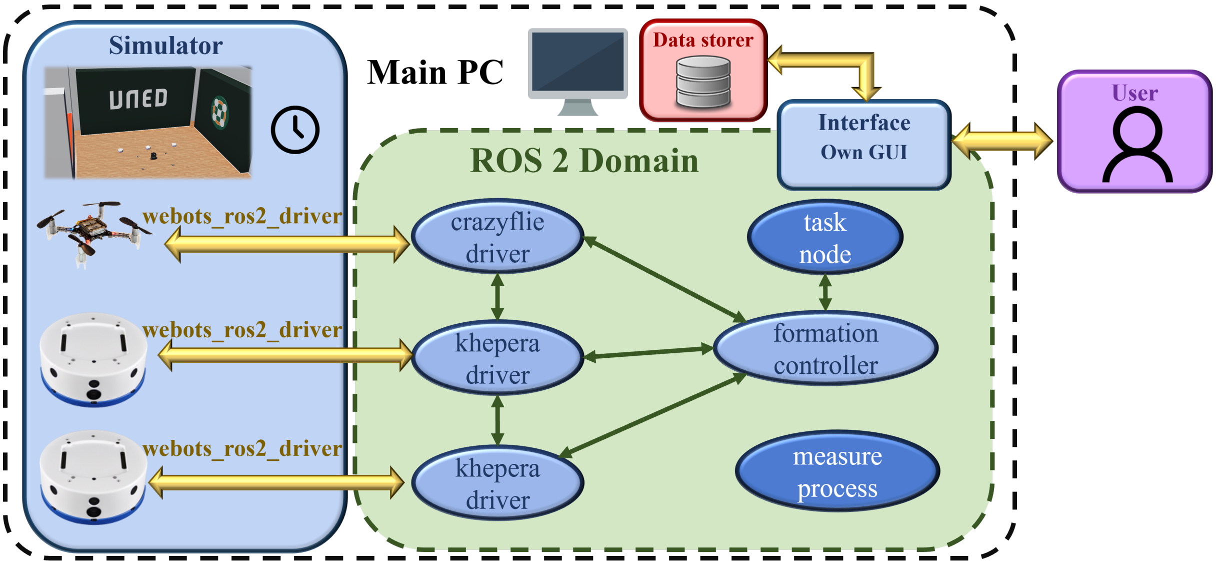


Figure 2.11. Example of virtual operation mode in Robotic Park.

In the case of the physical mode of operation, all agents are physical. This implies the need for specific modules, such as positioning systems, and taking into account the embedded systems of each agent as extensions of the software structure. Among the main challenges to be addressed in this mode are the battery duration of aerial robots (much shorter than mobile robots) and the resetting of the physical setting of the agents. On the one hand, the duration of the robots' energy conditions the maximum duration of the experiences. In RP, this duration is conditioned by the Crazyflie 2.1, therefore experiences can last up to 7 $[min]$. On the other hand, if the experience is successful, in RP the agents can return to their initial positions autonomously and turn off. However, if a failure occurs that avoids the autonomous system reset, an operator intervention is required to reset the system. The examples shown in Figures 2.9 and 2.10 are cases of this operation mode that combine aerial robots, mobile robots, and the Vicon MoCap positioning system.

Finally, the third mode of operation, hybrid, allows virtual and physical agents to be combined in the same experience. Through the middleware abstraction of ROS 2, an agent is not able to distinguish whether its neighbors are virtual or physical. The main challenge of this mode of operation is the clock synchronization between the two types of agents. Physical agents have their internal clock. Virtual agents use the simulator clock. These tools, when faced with saturation in the use of computational resources, slow down this clock, which implies a time lag between the agents in the system. The consequence of this phenomenon is erratic behavior. The main uses of this mode are:

- Scale the number of agents in the system without the constraint of physical space.
- Test the integration of new robots in combination with those already available.
- Integrate additional virtual sensors into the physical robots (cameras, lidars, etc.).
- Real robots on RP can be remotely teleoperated with a virtual backup on the user's computer, ensuring stability in case of connection loss.

Incorporating DTs is crucial for the last two points mentioned above. DTs are virtual agents that can communicate in real-time bidirectionally with physical agents. In the context of RP, Digital Twins are used to provide sensor feedback for the physical agent and represent it in the virtual environment. Sensor feedback from the agent allows for the incorporation of new sensors such as cameras or lidars in the case of the Crazyflie, without computational or energy implications. In addition, it allows behaviors such as obstacle avoidance in front of objects defined only in the virtual environment. On the other hand, representing the physical agent in the virtual environment allows other virtual agents to perceive it. Figure 2.12 shows an example diagram of MARS formation control experience using hybrid mode with virtual, physical, and DTs robots.

The precise definition of the structure of experiments is a fundamental factor in accelerating the realization of experiences by users without programming knowledge. Therefore, in RP a global launch file has been defined that receives as an attribute a configuration file (yaml format). In this file, using a specific dictionary to RP, the user can define the mode of operation of the platform, data logging, CPU monitoring, robots, etc. The user can include the developments carried out in this configuration file (nodes and their parameters and attributes), through the entry “Others” available in this file, reserved for that purpose. Below are the descriptions of the parameters that define all possible experiences.

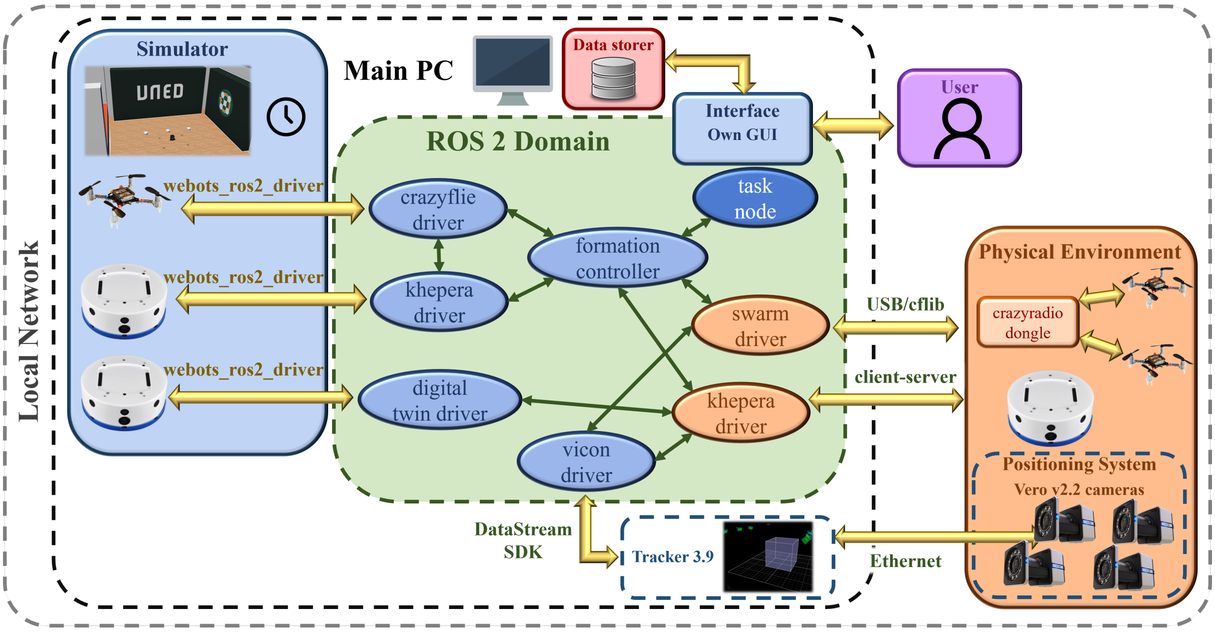


Figure 2.12. Example of hybrid operation mode in Robotic Park.

- **Operation mode**. The user can choose between “physical”, “virtual”, or “hybrid”. In case of choosing between the last two options, the user must select the simulation tool (Gazebo or Webots), if it runs a graphical interface and the name of the world file to be uploaded by the environment.

- **Type of experience**. This parameter determines which nodes should be executed externally to robots. The user can specify between “navigation”, “formation” or “identification”. The user must specify the node or launch defined in RP and its parameters within each option.

- **Control architecture**. The user sets MARS formation control as “centralized”, “distributed_ros” or “distributed_onboard”.

- **CPU monitoring**. If this option is enabled, the user specifies the name of the processes to be monitored and the sampling period.

- **Interface**. Depending on the objective of the experience, the user can choose the interfaces enabled between “rqt”, “rviz2” and “rp_default”. The first two are standard ROS 2 interfaces. The last one is the interface developed in RP. Its goal is to hide all interactions with the code while using the platform.

- **Data logging**. Using this option, the user defines the registration enablement, destination file name, desired topics and a small description. If the name field value is “date”, it sets the dataset’s name according to the date and time of the trial. The description will be recorded in a .txt file on the same path as the data files.

- **Robots**. In this entry, the types of robots used in the MARS are defined by a list of sub-entries. The most relevant parameters that must be indicated for each robot to determine the corresponding nodes are the nature of the robot (physical, virtual or DT), the positioning system (internal or external), its addresses (IP, URI and port) and the link it has with its neighbours (role, type, relationships, controller location, etc).

In RP, the user can log data using *ros2 bag* command line tool. This tool records system topics in .db3 files and a .yaml file that stores their information. It also allows the playback of datasets registered previously. However, using this tool to work with the topics' values is impractical. Therefore, we have developed a Python script that reads all the topics in a record and exports them individually to .csv files.

2.6 Robotic Park. System Communication

The communication system is a crucial component in systems that involve multiple elements working together to achieve a common goal. It is responsible for ensuring that all system components can effectively transmit, receive, and comprehend information shared between them. This is achieved through the establishment of a common communication channel and a set of message standards. In RP, this task is performed and masked from users by the middleware. To develop RP, ROS 2 as middleware has been used due to its characteristics and wide application in robot systems. Figure 2.13 shows the hardware infrastructure implemented in RP, illustrating the flow of information from each component to the stations responsible for publishing the information in ROS 2 domain, as well as the technology employed. Next to each communication channel (arrows) the technology it uses is indicated. Furthermore, in the case of positioning systems, it is shown how the external systems (Vicon and Marvelmind) communicate with the base station that publishes their estimates. Moreover, internal systems such as LPS or Lighthouse transmit information directly to the Crazyflies. The Crazyflies then communicate with the base station to publish the estimate to the network, which is fused with its internal sensors.

In our system, a ROS 2 layer above the hardware layer has been developed to improve the effective exchange of information between different components. Each component has its node within the ROS 2 domain that can generate or require information. The goal of these nodes is to communicate with devices using the provided libraries, Application Programming Interface (API), or SDK and convert useful data into ROS 2 messages using standard types. Modifications have been made to the existing nodes built by other developers to eliminate dependencies during installation and data processing. Table 2.1 describes the main nodes developed that compose the system.

One of the main interests of ROS 2 in MARS is the namespaces' use. This technique allows splitting the ROS 2 domain into subspaces. With its use, it is possible to reuse sets of nodes

Node	Package	Purpose
vicon_client	vicon_receiver	It reads the position of agents using Vicon's MoCap system from Tracker and publishes it.
marvelmind_ros2	marvelmind_ros2	It reads the position of agents using Marvelmind Beacon system from Marvelmind modem and publishes it.
rqt_gui	rqt_gui	Standard interface of ROS 2 for numerical topics display in graphs and publishing topics manually.
rviz2	rviz2	Standard interface of ROS 2 for 3D visualization of robots and their sensors.
measure_process	measure_process_ros2_pkg	It reads and publishes the computational consumption of the processes received as attributes.
swarm_driver	uned_crazyflie_driver	It performs the communication with the crazyflies via a crazyradio. It reads and publishes sensor data and sends the control commands to each drone.
kheperaIV_client_driver	uned_kheperaiv_driver	It communicates with a Khepera IV robot using the client-server protocol. It receives as attributes its IP address and port. It reads and publishes sensor data and transmits the control commands to the robot.
crazyflie_driver	uned_crazyflie_webots	It replicates in Webots' environment the behavior of the node driver for physical crazyflies and its firmware.
khepera_driver	uned_kheperaiv_webots	It replicates in Webots' environment the behavior of the node driver for physical Khepera IV and its firmware.
gazebo_driver	uned_swarm_driver	It replicates in Gazebo's environment the behavior of the node driver for physical crazyflie and its firmware.
gazebo_driver	uned_kheperaiv_driver	It replicates in Gazebo's environment the behavior of the node driver for physical Khepera IV and its firmware.
open_loop_signal	uned_swarm_task	It publishes high-level control commands for the robots by reading trajectory files or generating position or velocity commands for the robots.
formation_controller	uned_swarm_task	It can perform formation control for each robot individually or in a centralized approach for the whole MARS, based on the linked agents and their restrictions.

Table 2.1. Nodes in Robotic Park ROS 2 network.

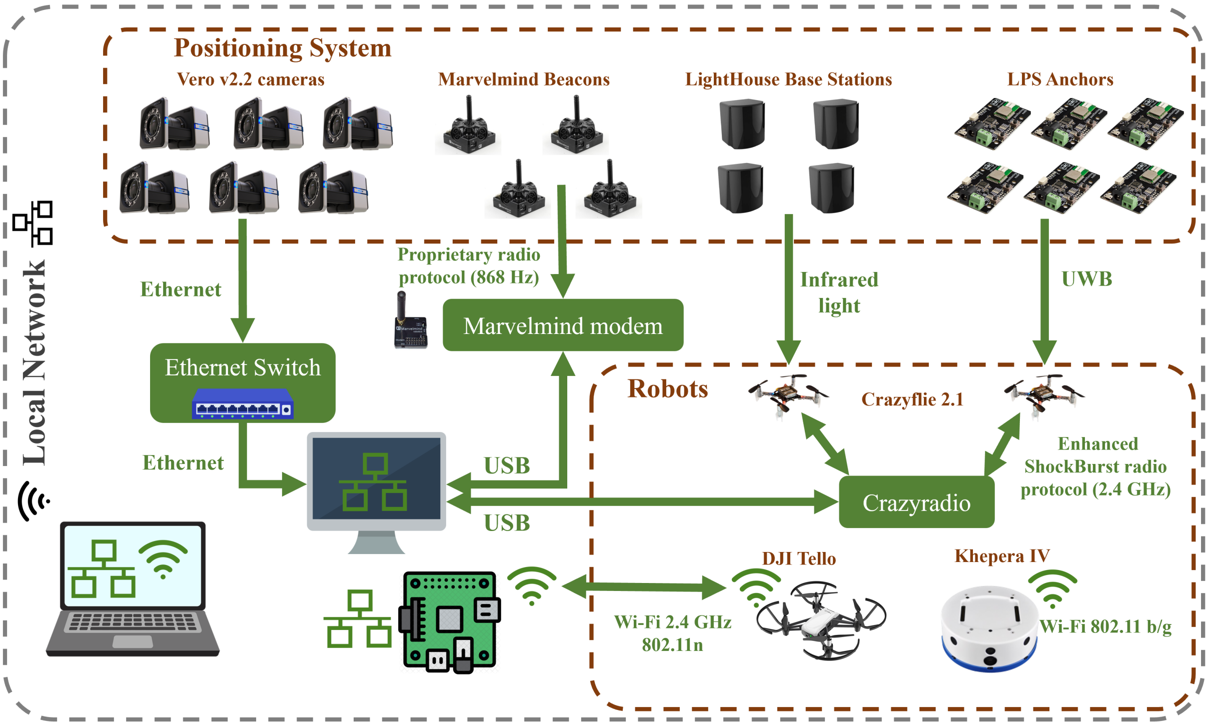


Figure 2.13. Hardware communication in Robotic Park.

and topics with only the addition of a prefix to their IDs. Furthermore, it allows the nodes and topics' clustering in data logging and feature extraction. The namespaces allow RP to exploit the scalability of MARS more efficiently. Figure 2.14 shows the diagram of a basic example of formation control using namespaces to replicate the space of the same type of robot. In this experiment, it is tested the formation control of four aerial robots. Two of the robots are only virtual in Webots. The other two robots are real. One of them uses the Vicon positioning system. The other uses the Lighthouse system. In addition, both use their DTs to obtain sensory information from the virtual environment to be perceived by the virtual robots. Figure 2.14 shows an example of a ROS topics and nodes diagram, which uses namespaces to collect topics and nodes of each robot in the MARS formation experience. This diagram corresponds to the scenario depicted in Figure 2.12, where the robot "khepera01" has a DT ("khepera driver" node). To make it easier to understand, different colors have been used to highlight the namespaces in the diagram. The light orange color is used for real robots, while the light blue color represents virtual robots. Similarly, nodes that communicate with physical robots follow the same rule. The arrows of topics (rectangles) that link nodes indicate the flow of information during the experience.

After analyzing the different options available for **monitoring** the system in real-time by the user, it has been considered that the best tools for their versatility and efficiency are RQT and RVIZ2. Both tools are native to ROS and allow 3D representations and 2D graphs of all the topics of the ROS 2 domain. Also, they will enable us to create in a simple way *ad*

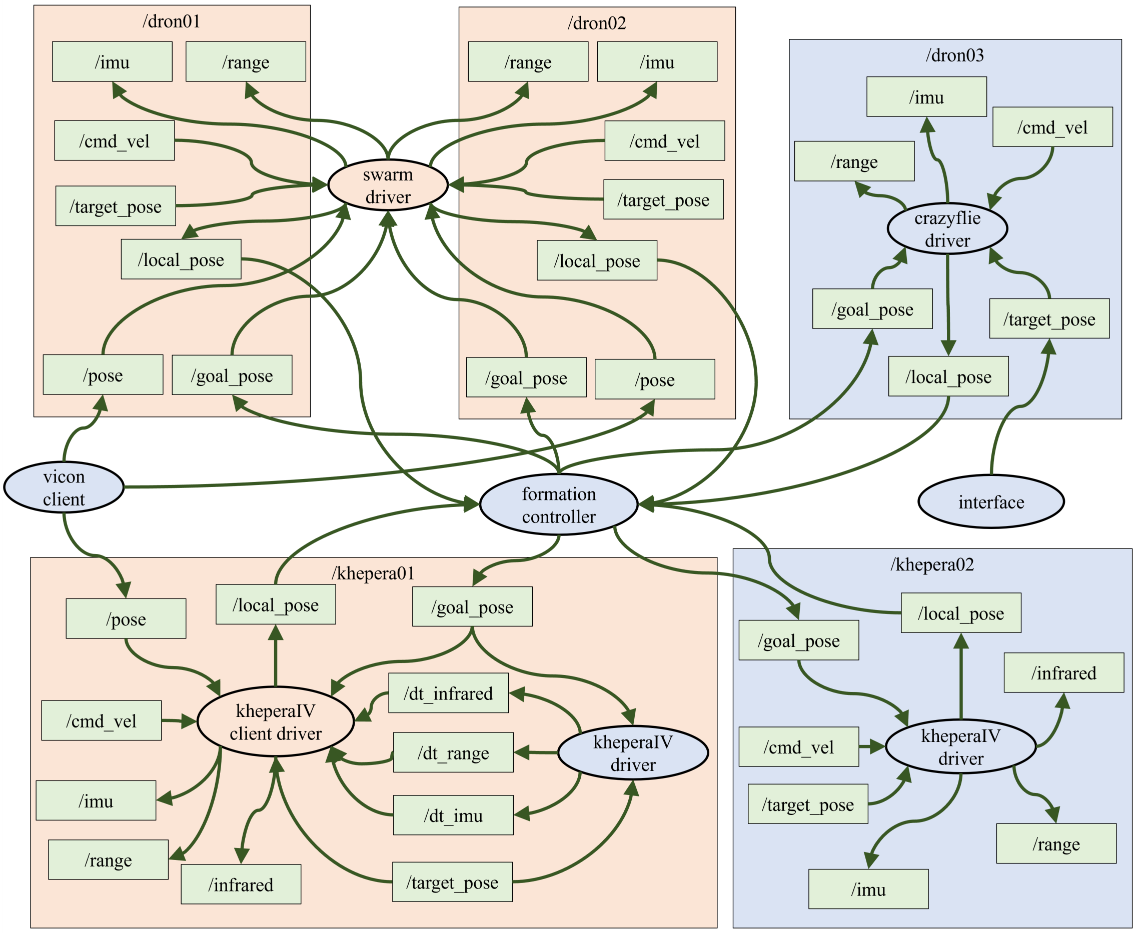


Figure 2.14. Example of a ROS diagram in MARS formation control using Mixed Reality.

hoc “perspectives” that can later be loaded from the configuration file of the experience. As mentioned above, an interface has been developed, which allows the user to generate and launch the configuration files graphically and interactively. It has limited functionalities for real-time monitoring using RQT-based tools and integrating the RVIZ2 terminal into the interface itself.

2.7 Indoor Positioning System (IPS)

As an indoor testbed, Robotic Park focuses on IPS [129, 130]. These positioning systems for robots can be classified as internal or external, depending on their configuration. In internal systems, the robot acquires and processes the information from the sensors, while in external systems the sensing and positioning steps are performed outside the robot.

The most basic internal positioning systems generate positions relative to an initial value defined by the user or consider that the robot starts from the coordinate origin. Typical sensors in these systems are encoders in odometry cases, cameras in optical flow sensors, or IMUs. An absolute internal positioning system uses static beacons and receivers on the robots. Initially, the beacons' positions must be known. The position of the robots can be estimated by measuring the delay of the signals, Time Difference of Arrival (TDoA). Some systems that require more processing power use Simultaneous Localization And Mapping (SLAM), either by scanners/lidars or vision. In this case, if the robot loads a previous map of the environment into its memory or there are predefined points that the robot can detect (i.e. QR patterns), it is possible that the positioning is obtained in absolute coordinates. External systems estimate the absolute position of robots, and the most typical ones are those that use vision or MoCap (i.e. Vicon Tracker or Optitrack).

Combinations of these systems are often used through data fusion algorithms, also called collaborative indoor positioning systems [131]. There are many algorithms available in the literature. The choice of the most appropiate will depend on the system, on the requirements (precision, accuracy, frequency, etc.), and the specific applications. The most common due to its versatility, performance, and implementation is the Kalman filter and its extended version [132].

RP's current systems are described in detail below, including their features, ROS 2 integration, and comparison.

2.7.1 Vicon Positioning System

The first external positioning system available at RP, is a Vicon's MoCap system[2]. Other relevant companies providing MoCap-based solutions include Qualisys[3] and Optitrack[4]. All these systems are based on cameras that emit infrared light and detect the reflection on spherical pearl markers. With the detection from multiple cameras, the software can triangulate the precise position of the marker in the measurement volume. If the user defines the markers' distribution on each robot on the tracker, the software can determine and track the robot's position in real-time. Ensuring each robot has a unique and asymmetrical distribution of pearl markers is crucial when designing a system. Figure 2.15 shows an example of this necessary asymmetry used in three Crazyflies. This aspect is necessary to prevent any estimation failures like instant changes of orientation or swap estimations. When implementing the MARS's controllers, it is important to consider that the cameras have a latency of 3.6 $[ms]$. For this reason, it has been decided to set the software frequency to 100 $[Hz]$ to guarantee a conservative response in the feedback of the position and velocity controllers.

At the hardware level, the Vicon Tracker System consists of six Vero v2.2 cameras, see Figure 2.16a, with Power over Ethernet (PoE) technology. Highlights include a resolution of 2.2 Mega Píxel (MP), a maximum frame rate of 300 $[Hz]$, a maximum power consumption of 12 $[W]$,

[2] Official Vicon web: `https://www.vicon.com/`

[3] Qualisys Track Manager: `https://www.qualisys.com/software/qualisys-track-manager/`

[4] Motive: `https://optitrack.com/software/motive/`

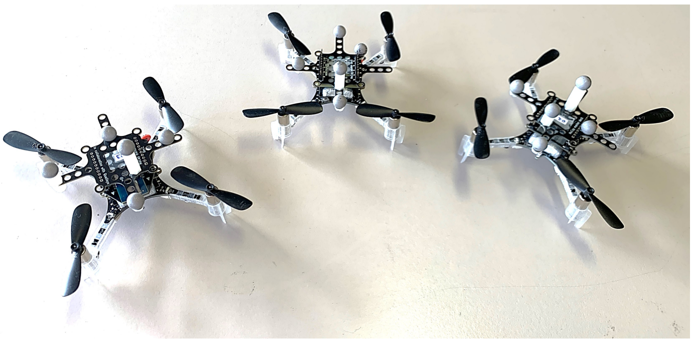
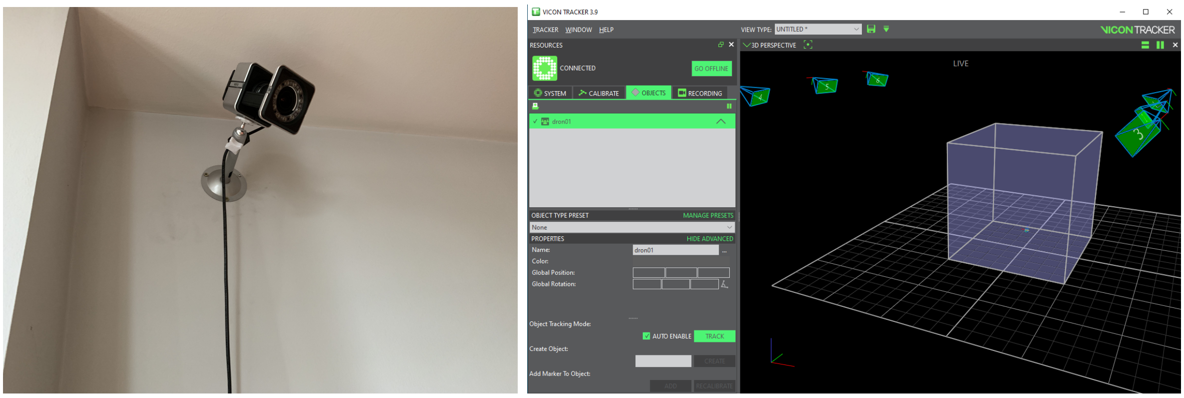


Figure 2.15. Pearl marker distribution for three Crazyflies 2.1.

and a weight of 0.57 $[kg]$. These devices come equipped with an accelerometer that can detect any shifts or vibrations in the cameras, indicating the need for recalibration. The six available cameras are connected to an Ethernet switch that connects them to the PC, where the software processes the data. The Tracker 3.9 software tool is installed on this PC. Figure 2.16b shows the main dashboard. This tool is responsible for estimating the position of a robot with an accuracy of more than 1mm based on the detection of the markers defined by the system. This software has to run on a Windows operating system. For the integration of ROS 2 for communication between MARS agents, Vicon's Python SDK has been used. For this purpose, a ROS 2 node has been developed that reads the instant agent's position from the tracker, converts them into standard messages, and publishes them on the ROS 2 network.



VIEW TYPE: UNTITLED *

✓ dron01

None

Name: dron01

Color:

Global Position:

Global Rotation:

Object Tracking Mode: ☑ AUTO ENABLE

Create Object:

Add Marker To Object:

(a) Cámara Vero v2.2.

(b) Software Tracker 3.9.

Figure 2.16. Vicon's Motion Capture components.

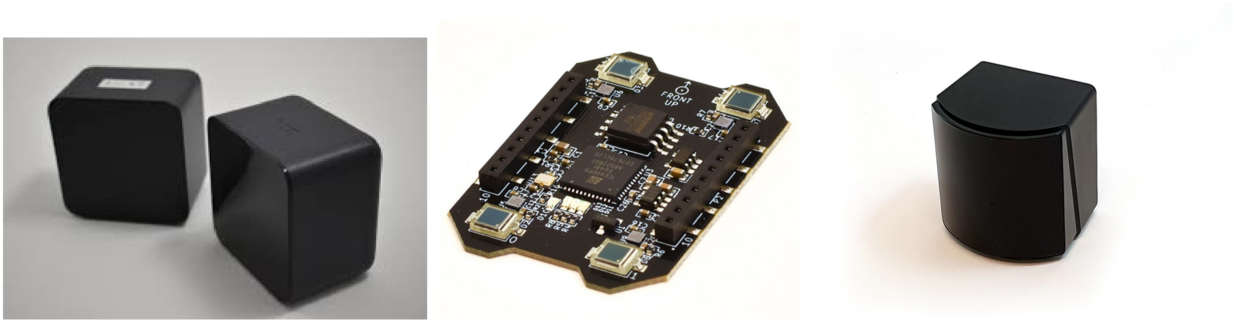
2.7.2 Lighthouse Positioning System

Lighthouse Positioning System is an indoor optical-based positioning system developed by Bitcraze[5]. The system provides tracking accuracy close to that achievable with a MoCap system

[5] Lighthouse: https://www.bitcraze.io/.../positioning/ligthouse-positioning-system/

but at a much lower cost. Its main advantage is the onboard acquisition of the tracked device position. Therefore, it reduces transmission delays and network saturation. The system uses HTC Vive Station 2.0 base stations as optical beacons. The receiver (a photodiode) can estimate its position with a relative accuracy of better than a millimeter and an absolute accuracy of better than a decimetre. This feature means that you can return to a starting point with millimeter accuracy. However, reaching a target point in absolute coordinates may be off by centimeters (less than a decimetre).

Base stations perform laser scanning in the workspace. It is possible to determine the position and orientation of an object relative to a base station using multiple receivers on a single deck, see Figure 2.17b. There are two generations of base stations, and both are compatible: V1 (Figure 2.17a) and V2 (Figure 2.17c). The Lighthouse V1 has two rotating drums, while the Lighthouse has two inclined light planes on its only drum. Lighthouse V1 systems can use up to 2 base stations, while Lighthouse V2 systems are designed for up to 16, allowing for a much more extended working volume. It has a range of 6 $[m]$, an operating frequency of 50 $[Hz]$, a horizontal viewing angle of 150^o, and a vertical viewing angle of 110^o. To accurately determine the position and orientation of each robot, it is essential to have prior knowledge of the position and orientation of the base stations. This process, known as System Geometry, can be obtained automatically by the Crazyflie client and saved in the robot, or it can be kept in a file that can be used to write the same geometry information to multiple robots. It enables multiple robots to operate in the same workspace with distributed architectures.



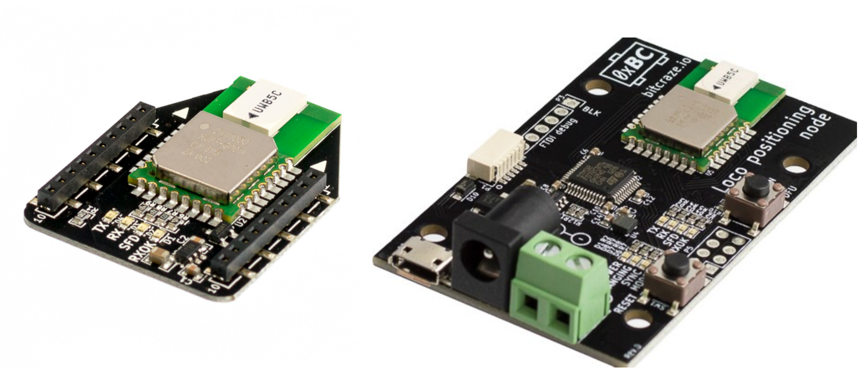
(a) Lighthouse base station V1. (b) Lighthouse Positioning deck. (c) Lighthouse base station V2.

Figure 2.17. Lighthouse Positioning System components.

RP offers a system with four base stations placed in the working volume's upper corners, allowing the space to be maximized and pushed toward the edges. To ensure accurate capture of the ground area, it is recommended to space the two beacons 0.5 $[m]$ apart. However, this spacing will cause the corner opposite the beacon to be outside its nominal range by some distance. The 4-base station configuration provides the most optimal solution to avoid this issue. Being a technology based on infrared emissions, it is incompatible with MoCap systems as the camera's signal interferes with the robots' photodiodes. Active markers must replace the pearl markers, and the cameras must be set to passive mode to allow the combined use of these systems. Other systems based on UWB or ultrasonic technologies are compatible. The developers' detailed analysis of this positioning system can be found in [133].

2.7.3 Loco Positioning System (LPS)

Another positioning system available on RP platform is Loco Positioning System. It is an internal positioning system developed by Bitcraze[6], and it is based on UWB technology. The system consists of multiple anchors (see Figure 2.18b) placed around the work area and tags attached to the agent (see Figure 2.18a). The system can accurately estimate the agent's global position by analyzing the timing of transmissions between the targets and anchors. Its performance is comparable to that of the Lighthouse Positioning System.



(a) Loco Positioning deck/tag.

(b) Anchors.

Figure 2.18. Loco Positioning System components.

This system has three modes of operation that determine the performance and range of the system:

- Two Way Ranging (TWR). When operating in this mode, the robot's tag pings the anchors in a sequence, which enables to measure the distance between itself and each anchor. With this information, it requires a minimum of four anchors to calculate the 3D position of a receiver. It is advisable to have six anchors to improve accuracy and reliability, although the maximum allowable is 8. This last configuration is the most precise mode, and it is effective even when the receiver has left the area enclosed by the anchors.
- TDoA 2. In this mode, the anchor continuously sends synchronization packets. The onboard deck receives these packets and determines the relative distance to three anchors by measuring the difference in arrival time of the packets. This information is used to estimate the 3D position of the robot. The passive operation of the receivers allows for the system's scalability and enables the use of more robots in the arena. For optimal TDoA performance, the recommendation is to use eight anchors placed at the corners of the workspace.

[6] LPS: `https://www.bitcraze.io/documentation/system/positioning/loco-positioning-system/`

- TDoA 3. The main difference with TDoA 2 is that the random transmission schedule replaces the time interval scheme. This improvement allows the addition of more anchors. It will enable the platform to scale to larger spaces or span multiple rooms with no line of sight between all the anchors at the cost of adding some noise to the measurement. It also makes it more robust and can dynamically handle the loss or addition of anchors.

RP, implements this system in a TDoA 3 configuration with eight anchors. This choice is based on the platform's flexibility and to facilitate its scalability. Therefore, if the developers want to expand the workspace to different building rooms, no modifications will be necessary to the basic configuration of the platform. In this way, experimental availability will not be interrupted.

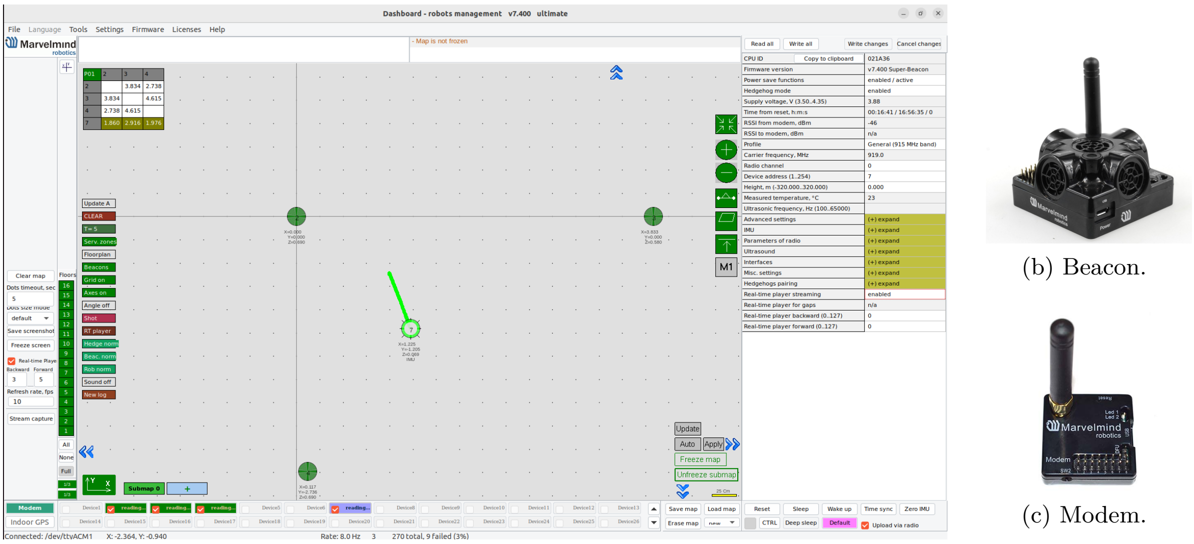
2.7.4 Marvelmind Indoor Positioning System

Marvelmind has an IPS called Marvelmind IPS, Marvelmind Indoor Global Positioning System (GPS), or Marvelmind RTLS. Its main elements are the beacons (see Figure 2.19b), the modem (see Figure 2.19c), and the Marvelmind Dashboard software (see Figure 2.19a). In addition to the ultrasonic transducers, all beacons include a 6D IMU (3D accelerometer + 3D gyroscope). The beacons are interconnected by a radio interface in a license-free band (915/868 $[MHz]$). This system triangulates the positions of the moving beacons with the measure of the delay of an ultrasonic signal (Time Of Flight (ToF)) between a set of beacons. It can improve the accuracy of the UWB systems by a factor of 10 and Bluetooth-based systems by a factor of 100. Typically, this system has an error of ± 2 $[cm]$. The maximum range in industrial settings is 30 $[m]$, but it can potentially reach up to 50 $[m]$ in laboratory environments. There are three different architectures available in the system that are based on the agents and the application field:

- Inverse Architecture (IA). In this setup, each fixed beacon should operate on a unique frequency of 19, 25, 31 & 37 $[kHz]$. The mobile beacons act as receivers. Although there is no limit to the number of mobile beacons used, this configuration is not recommended for drones. The rate of position estimation is 40 $[Hz]$.

- Non-Inverse Architecture (NIA). In this configuration all stationary beacons are receivers and can operate on any frequency. The mobile beacons act as transmitters and can work at any frequency. The position estimation rate depends on the number of moving beacons (n) according to $f = 40$ $[Hz/n]$. This configuration is suitable for use with drones, but the mobile beacon limit is four.

- Multi-Frequency NIA (MF NIA) is an implementation similar to NIA. However, it allows up to eight mobile beacons by operating twice as many frequencies (19, 22, 25, 28, 31, 34, 37 & 45 $[kHz]$).



P01	2	3	4
2		3.834	2.738
3	3.834		4.615
4	2.738	4.615	
7	1.860	2.916	1.976

CPU ID	021A36
Firmware version	v7.400 Super-Beacon
Power save functions	enabled / active
Hedgehog mode	enabled
Supply voltage, V (3.50..4.35)	3.88
Time from reset, h:m:s	00:16:41 / 16:56:35 / 0
RSSI from modem, dBm	-46
RSSI to modem, dBm	n/a
Profile	General (915 MHz band)
Carrier frequency, MHz	919.0
Radio channel	0
Device address (1..254)	7
Height, m (-320.000..320.000)	0.000
Measured temperature, °C	23
Ultrasonic frequency, Hz (100..65000)	
Advanced settings	(+) expand
IMU	(+) expand
Parameters of radio	(+) expand
Ultrasound	(+) expand
Interfaces	(+) expand
Misc. settings	(+) expand
Hedgehogs pairing	(+) expand
Real-time player streaming	enabled
Real-time player for gaps	n/a
Real-time player backward (0..127)	0
Real-time player forward (0..127)	0

Connected: /dev/ttyACM1 X: -2.364, Y: -0.940 Rate: 8.0 Hz 3 270 total, 9 failed (3%)

(a) Marvelmind Dashboard.

(b) Beacon.

(c) Modem.

Figure 2.19. Marvelmind Positioning System components.

RP implements an inverse architecture to maximize the number of mobile robots and maintain the position estimation rate. The fixed beacons have been positioned at a height of 0.5m to prevent any 'shadows' between agents, which is the minimum drone altitude. This system has an API that allows the development of programs in C and Python. In this case, for its integration in ROS 2, RP uses its official repository on GitHub[7].

2.7.5 Vision-based Positioning System

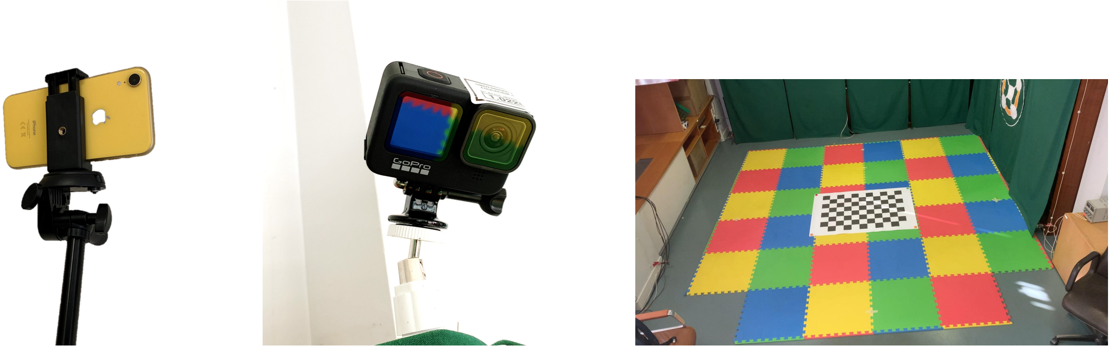
At RP,an easy-to-use positioning system that processes images from RGB cameras has been developed. These systems are commonly used for experimental platforms that use only mobile robots with patterns placed on top. A zenithal camera estimates their position by identifying the tags. However, this approach is not feasible in the case of RP because drones can cause visual occlusions.

Regarding aerial robots, a single camera provides a solution with one degree of freedom. Therefore, a minimum of two cameras are necessary. By pre-calibrating the positions of these two cameras and understanding their location within the workspace, the system can accurately determine the absolute position of any robot by calculating the intersection of the two straight lines detected by both cameras.

This system presents several challenges, e.g. lighting conditions. The accuracy of the system is affected by the precision of the cameras and the calibration of their characteristic parameters. Its advantages include the fact that it does not require any communication or intervention

[7] Marvelmind ROS 2 Upstream: `https://github.com/MarvelmindRobotics/marvelmind_ros2_upstream`

from robots in the system. Currently, its offline operation has been implemented in Matlab, using functions available in OpenCV to ease its subsequent integration into a ROS 2 node for its execution from the launch file. The system comprises two cameras: iPhone XR (see Figure 2.20a) and GoPro HERO 9 Black (see Figure 2.20b). Appendix C details the process carried out for both cameras' calibration. Once the intrinsic parameters were obtained, the cameras were mounted in their respective locations and the extrinsic parameters of each camera were obtained. These parameters generate the homogeneous transformation matrix between the cameras and the reference frame of the workspace. To achieve this, a calibration pattern was placed at the origin of the system coordinates and an image of the final positions of the cameras was captured, as shown in Figure 2.20c. At this point, the system is ready to work.



(a) Cam.1: iPhone XR. (b) Cam.2: GoPro HERO 9 Black. (c) Pattern in Robotic Park arena.

Figure 2.20. Vision-based System cameras.

Figure 2.21 illustrates the steps involved in the system's workflow during experiences. After the system is ready, it determines the reference frame (undistorted). This frame must represent all the static objects in the scene. The first step of the cycle is to read the frames of each camera, which undergoes a correction process to remove any distortion from the camera lens.

Next, the current frames are compared with their references to detect any changes. We apply an erosion to the binary image obtained and extract the centroids from the resulting closed areas. The pairing of these areas in both cameras is done by optimizing the minimum distance between them. This minimum distance should not exceed a certain threshold. If it does, the physical sense determines they are two objects. If a pairing of detections works, it is optimized to the last position value of the available robots. This pairing considers that a robot between two frames has not moved over a certain threshold. If the pairing is done successfully, the algorithm updates the position of the paired robot and repeats this process for all centroids.

To avoid constant track changes regarding the initial reference frame (green arrows), an option to update the reference frame with the last frame read (yellow arrows) is proposed. In this implementation, the system would only detect those objects that move between frames. This technique helps to reduce the detection and pairing of robots that remain immobile for periods (computational efficiency).

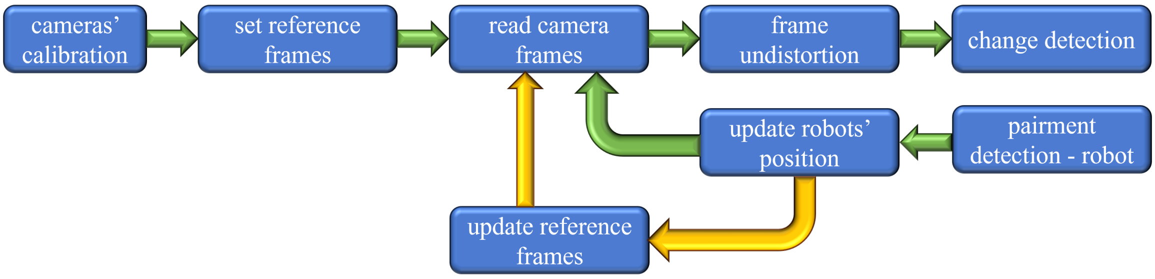
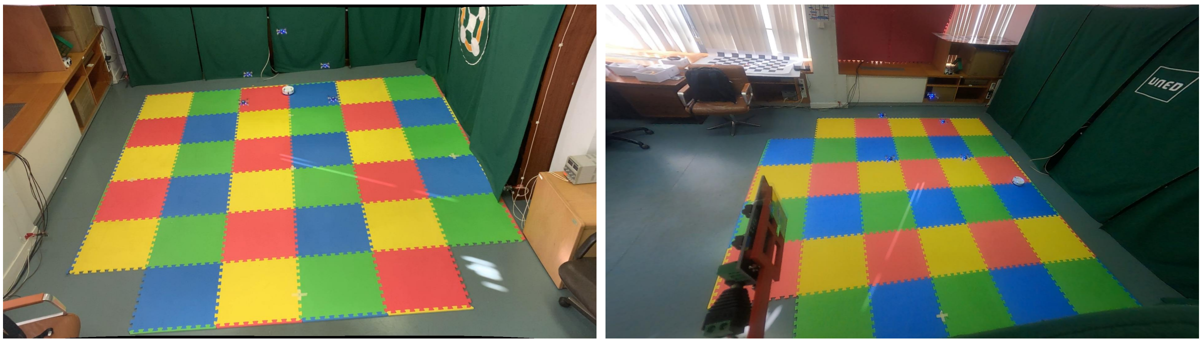


Figure 2.21. Workflow of Vision-based Positioning System.

An experience with five Crazyflie 2.1 has been carried out to validate this system. Figure 2.22 shows the undistorted instant frames of both cameras. Figures 2.23a, 2.23b, and 2.23c show the results obtained on the X, Y, and Z axes. In them, each colour represents a drone. The continuous line represents their ground truth, while the dotted line is the position determined by the developed system. Figure 2.23d shows the 3D results where the red dots represent the ground truth and the blue dots represent the estimation of this positioning system. The results show an average precision of 3 $[cm]$ (maximum distance between the projections of the detections of each camera) and an accuracy of 1.62 ± 2.78 $[cm]$ (difference concerning the signal measured with other positioning systems). Therefore, the proposed algorithm shows enough performance for tracking robots, but the accuracy and rate are not suitable for position control.

2.7.6 Comparison

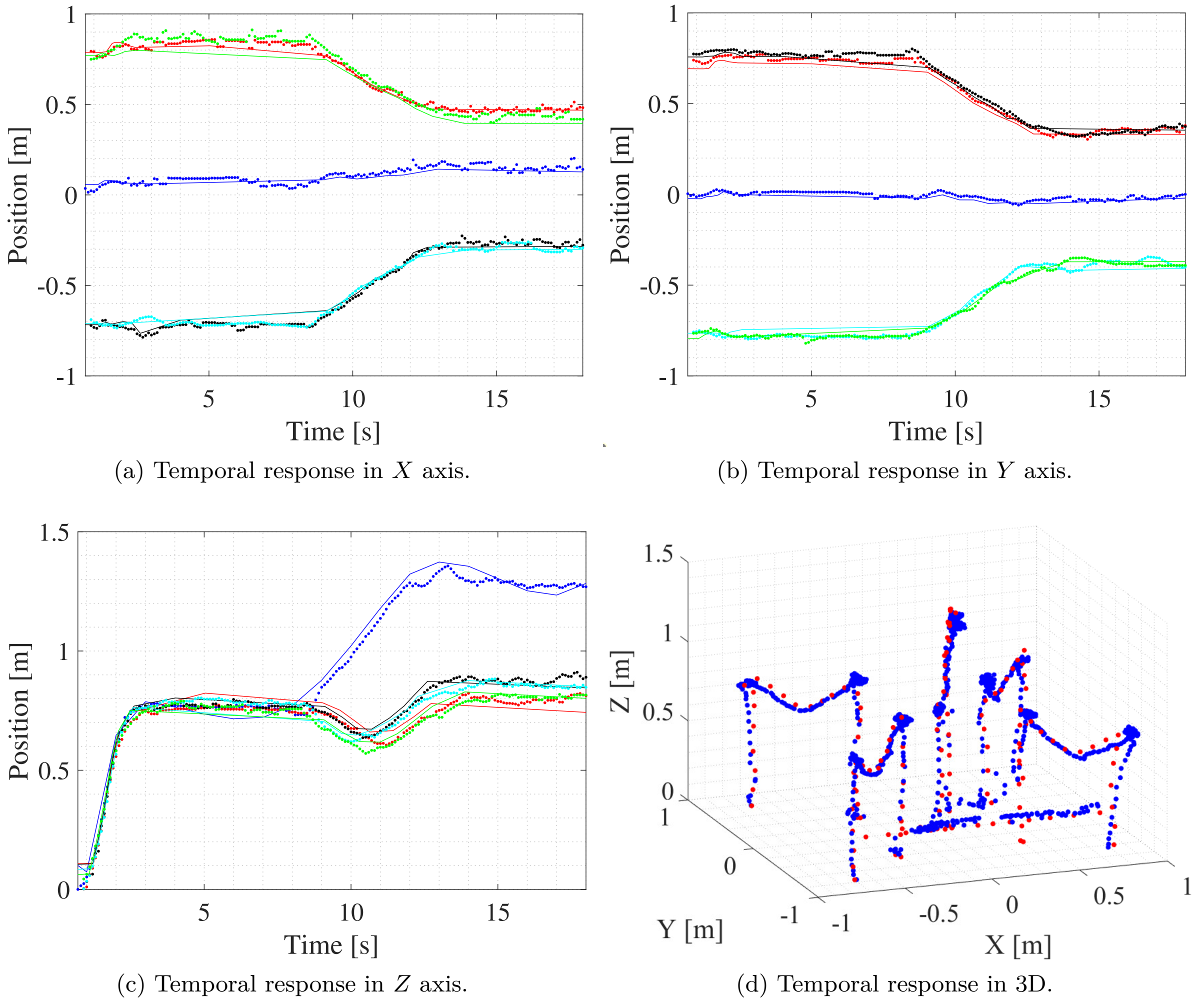
After the overview of the positioning systems available at RP, two experiences are shown in which the difference in results of each system is observable. First, experience evaluate the systems available for the Khepera IV mobile robots. Figure 2.24 shows the result obtained in the X-Y plane during a trajectory along a hexagon of 1 $[m]$ side for two laps. In this case, odometry from the encoders of the robot's wheels (black markers), Vicon's MoCap at 100 $[Hz]$



(a) iPhone frame.

(b) GoPro frame.

Figure 2.22. Vision-based positioning system instant undistort frames.



(a) Temporal response in X axis.

(b) Temporal response in Y axis.

(c) Temporal response in Z axis.

(d) Temporal response in 3D.

Figure 2.23. Temporal response of MARS (first-second-third-fourth-fifth) using Vision-based Positioning System (continuous line: ground truth. dotted line: estimation).

(blue markers) and the Marvelmind GPS with four static beacons (red markers) have been evaluated simultaneously. For position control, odometry has been used, which justifies that the drawn path coincides with a perfect hexagon. The characteristics of the Vicon system allow it to be used as ground truth. Therefore, it is observed that the odometry does not have an exact result in turns since the observed trajectory is outdated with the original but mantains the size of the hexagon. Finally, the marvelmind system demonstrates significant correlation and limited error within the manufacturer's stated thresholds. It is observed that their frequency makes fine and high tracking difficult. However, to control the position of these robots it is sufficient in combination with another system, such as odometry, given their operating speed.

Secondly, we carried out an experience where we evaluated the systems available for Crazyflie 2.1. The small size of the Crazyflie drone and the interference caused by the Vicon camera emitters on the Lighthouse deck make it impossible to use all systems at once. Moreover,

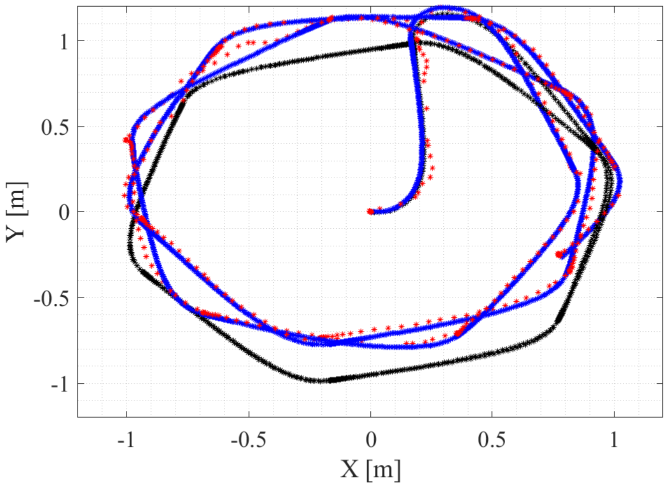
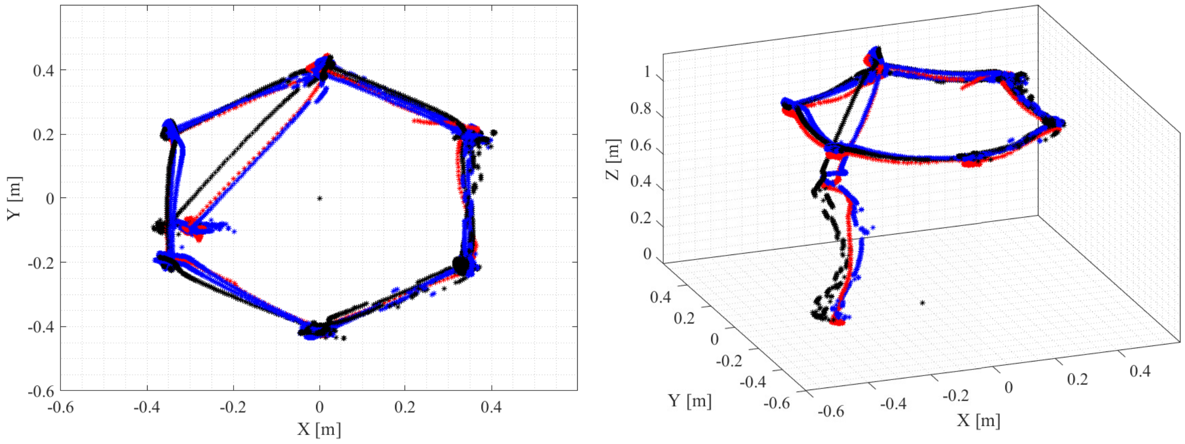


Figure 2.24. Comparison between odometry (black), Vicon's MoCap (blue) and Marvelmind GPS (red) as positioning systems.

because of the firmware configuration, extracting individual measurements from the positioning system is difficult without combining them with data from other sensors like IMU and flow deck. This measure is what the drone uses as feedback in the individual control, which implies that the results will not show the accuracy of the positioning systems but their effect on the drone position control.

Therefore, there have been three identical experiences where a drone performs a trajectory along a hexagon of 0.5 $[m]$ on the side for two laps. Figure 2.25 shows the results obtained of the systems evaluated: Vicon's MoCap at 100 $[Hz]$ (black markers), Lighthouse Positioning System (blue markers) and LPS (red markers). These results reflect that all three systems



(a) Temporal response in XY plane.

(b) Temporal response in 3D.

Figure 2.25. Comparison between Vicon's MoCap (black), Lighthouse Positioning System (blue) and Loco Positioning System (red).

adequately support Crazyflie position control. There is a smooth performance from the Vicon MoCap. It is due to its high accuracy and refresh rate, which causes a smooth signal in feedback. The system addresses challenges in generating asymmetric distributions for many drones and efficiently using communication channels to transmit position. In contrast, the Lighthouse and LPS systems exhibit similar performance with slightly more noise than the first case. However, the controllers effectively compensate for these minor differences, resulting in no significant variations. The main advantage of these systems is that they are not affected by the system's scalability, and the position estimation is performed onboard, leading to savings in the use of communication channels.

2.8 Simulators

2.8.1 Tools overview

As mentioned previously in this thesis, simulators are a type of virtual experimental platform. They allow us to analyze a wide variety of current problems in our society in a repeatable, fast, and completely configurable way [134–136]. One of the benefits of using simulators is that they do not require physical space and can be accessed anytime and anywhere. That makes them ideal for displaying demos at conferences and exhibitions. However, it is challenging to discover new problems based solely on system analytical models that are observable in real systems [137]. Many simulation tools focus on specific cases to provide accuracy and high performance in their answers. An example is [138], which uses the Unreal Engine 4 environment for designing a precise underwater robotics tool. In [139], a specific review of tools focused on autonomous vehicle simulators is available. Due to the possibility of automating simulations and configuring their parameters, it is especially useful in initial tasks of Reinforcement Learning systems [140]. More focused on MAS, a recent review of swarm robotics, both at the hardware level and its main simulators, is available at [141]. These tools are not only useful for research. Their increasing realism and accuracy have also made them popular for professional training. Adding the AR and VR techniques has further improved user engagement. A notable example of their application in recent years is in medicine and surgeon training [142, 143].

When introducing a new experimental platform to the community, it is essential to use technologies that are widely accepted within the industry. It creates a positive impression and encourages people to use the platform. In this thesis, an analysis of the main tools that impact work with MRS in ROS 2 has been carried out following the open-source philosophy to achieve this objective. Below is a brief review of the tools analyzed.

- *Matlab/Simulink.* It is one of the most widespread programming and numerical calculation platforms in the world for teaching and research. Its programming language is based on C and has many resources in the form of toolboxes that make it a very versatile and easy-to-learn tool with a widespread support community behind it. Since its 2019b version, *ROS Toolbox* has been available, which includes all the necessary functionalities to be able to communicate within any network where ROS or ROS 2 is being used.

- *Gazebo* [102]. It is one of the most widely used open-source simulators currently in use [144]. It is developed by *Open Robotics*, the same organization in charge of the development of ROS and ROS 2, so it has a native integration, favoring its joint use. Its extensive library of sensors and joints, and its fine reproduction of the dynamic properties of the models make it the simulator par excellence in mobile robotics and one of the top three in manipulation and aerial robotics. Specialized tools dominate the marine robotics sector.

- *Gazebo Ignition.* This is a component of *Ignition Robotics*, a set of libraries designed to fast develop simulation and robot applications. It is the project proposed as the successor of Gazebo, including improvements in simulation time and some dynamics such as friction, achieving smoother and more realistic results. Upon succesion, Gazebo will be renamed *Gazebo Classic*, and the term "Ignition" will disappear from this version to preserve terminology.

- *Webots* [103]. It is a professional-grade, open-source tool developed by Cyberbotics Ltd. that can simulate robots and multi-agent systems across different platforms. The development environment comes equipped with a variety of useful tools such as modeling, programming, and simulation. It is considered the first alternative to Gazebo and is widely used in various sectors like industry, education, and research.

- *CoppeliaSim* (old *V-REP* [145]). It is a multiplatform commercial software tool. It allows working with the most common programming languages (C/C++, Java, Python, etc.) and has APIs to integrate with ROS. It allows the export of models in the most used formats (URDF, COLLADA, DXF, OBJ, STL, glTF, etc). Its learning curve is much faster than other tools.

- *Pybullet* [146]. This software is a Python module for robot simulation due to the advantage of the properties of this language for machine learning tasks. PyBullet can be easily used with TensorFlow and OpenAI Gym. It is used in labs such as Google Brain [147], Stanford AI Lab [148], or OpenAI.

At RP, simulation tools play a crucial role. As commented in section 1.3, they provide students with a safe and instructive environment to develop their skills before they work in a physical environment. Researchers can use these tools to familiarize themselves with the platform and validate their developments before spending time and resources on the physical system. Simulation tools are also essential to create experiences that blend virtual and real environments, expanding the platform's capabilities to Mixed Reality (MR) and Digital Twin (DT). Figure 2.26 shows a general diagram of the elements that a simulator must have to integrate into Robotic Park.

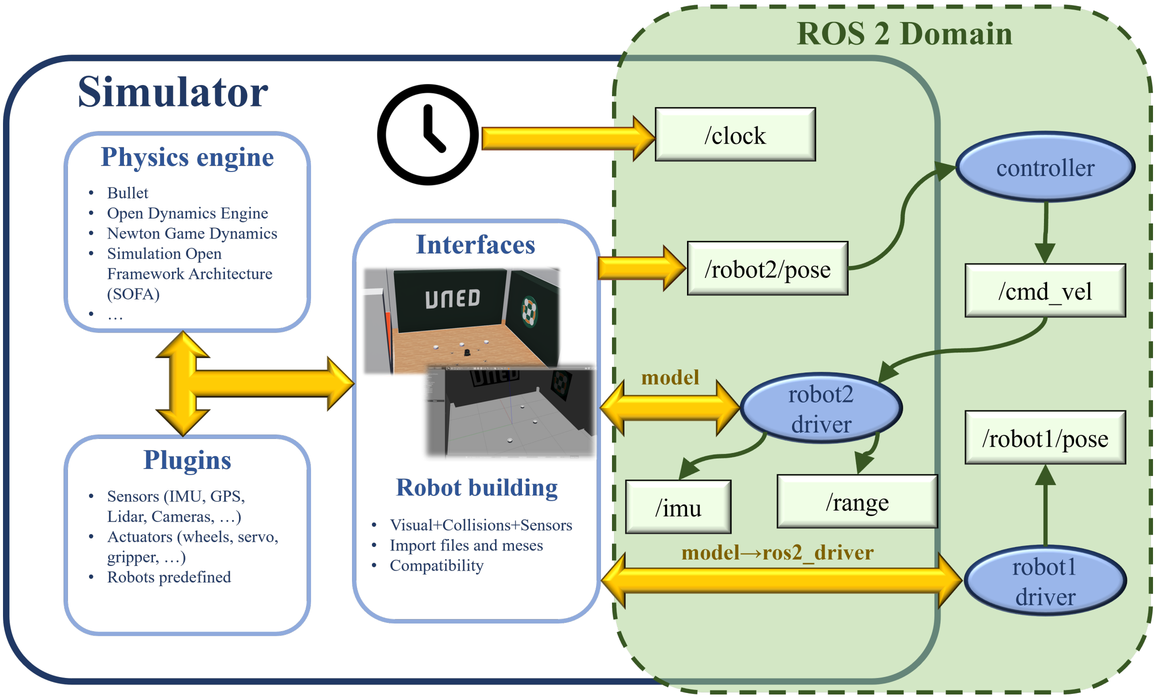


Figure 2.26. Basic diagram of ROS 2 simulator.

On the one hand, one of the key features of each simulator is its physics engine. It determines the power and realism of simulations. On the other hand, the libraries and plugins available are a critical factor that determines the degree of “plug and play” of the tool. For inexperienced users, such as students, the learning and start-up curve must quickly focus on their development. There are two ways to integrate robots in ROS 2 through a node. On the one hand, the node can be configured from the robot model using plugins (as in the Gazebo case). In this case, the node runs inside the simulator, and its parameters can only be configured from the plugin attributes. On the other hand, if access to the virtual robot is done through a set of libraries, the user can design and execute the driver node from outside the simulator (as in the case of Webots). The three simulation tools in RP are Matlab/Simulink, Gazebo and Webots. The main reasons they have been selected are their widespread use in the community and their compatibility with ROS 2. Below is detailed coverage of its use and integration into RP.

2.8.2 Matlab

The first tool developed in this work for modeling and simulation is based on Matlab and Simulink. One of the main benefits of this tool is that it can be used on multiple platforms, and it offers versatility by allowing users to build models using block diagrams. Hence, users can carry out simulations regardless of their operating system or programming knowledge. However, this tool has some disadvantages. For instance, it is a paid tool, and there can be serious compatibility issues when using newer versions of Simulink toolboxes with files from older versions. It means that, to maintain compatibility, models must be developed using older versions of the tool, which may not allow users to exploit the latest developments. The oldest version of the tool that is currently compatible with ROS 2 is 2019b.

In RP, the tool is primarily used to introduce students to the control architecture of quadcopters and differential robots. A comprehensive model and corresponding control architectures for both types of robots have been developed to this end. Scaling up these models can be difficult due to variable nomenclature and resource consumption. For these reasons, these use cases are focused on one agent. Figure 2.27 shows an overview of the development of the Crazyflie 2.1 script.

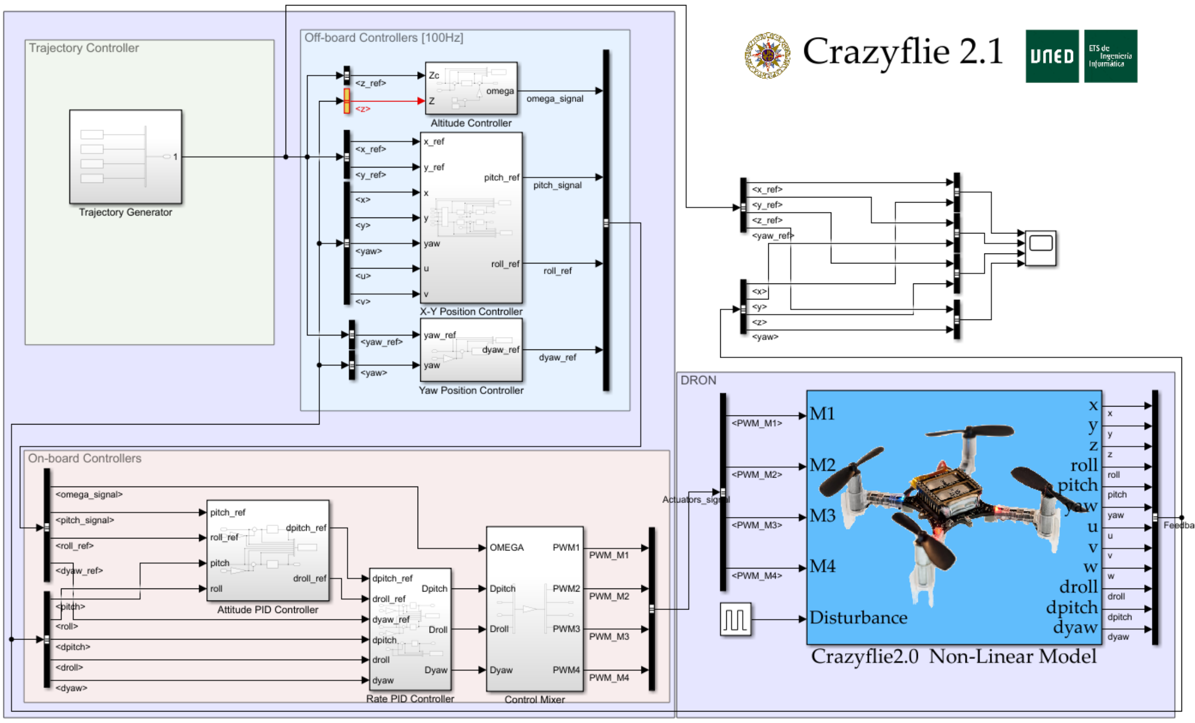


Figure 2.27. Model and controllers of Crazyflie 2.X in Simulink.

This file integrates the generated path (green area), the position and speed control (light blue area), the orientation and attitude controller (orange area), and the physical model of the robot (blue area). This architecture is a replica of the architecture that exists in the robot firmware. Therefore, this modular implementation has been made to simplify the change of control architectures. The physical model implemented is Non-Linear to have maximum realism at this level, see Figure 2.28. This model provides an option to introduce disturbances by adding a vector of forces directly into the model equations after calculating the forces introduced by each quadcopter rotor. The disturbance signal can emulate the presence of air currents or other agents flying nearby, allowing to simulation of various scenarios. The modeling of this robot is covered in-depth in section 4.2. The controllers currently available are continuous, discrete, event-based PIDs with fixed thresholds and event-based PIDs with adaptive thresholds.

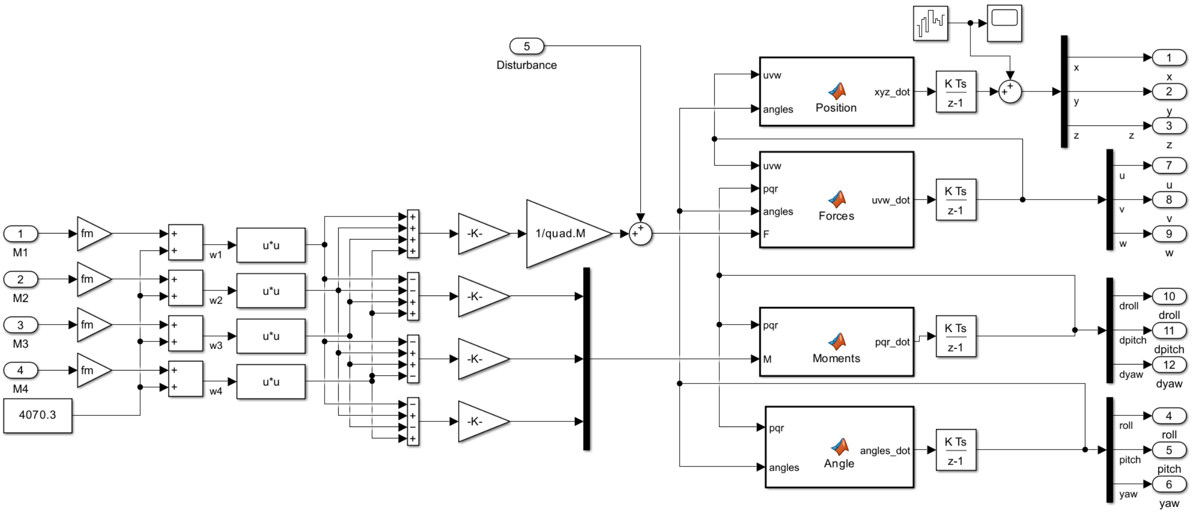


Figure 2.28. Physical model of Crazyflie 2.X in Simulink.

In the case of the differential robots, a structure similar to that described for the quadcopters has been developed. Figure 2.29 shows the simulation script in Simulink.

The control structure is much simpler since it only requires a position control composed of two velocity and rotation controllers. Figure 2.30 shows the implementation of the model described in section 3.2. Dynamic robot and motors models have been incorporated to improve accuracy compared to the classic kinematic model.

Once the user gets used to the robots and has overcome their architecture, integration with ROS 2 is done through the toolbox "ROS Toolbox" components. This toolbox allows communication directly with the rest of the nodes and topics in the network by creating a default node for the entire script. Figure 2.31 shows an example of its use.

In this case, the control architecture is implemented for Crazyflie 2.1, but instead of simulating the dynamic of the robot, it communicates with the real robot. To do this, it publishes the control signal of each rotor using a topic of the type "Float64MultiArray" called "/cmd_motor"

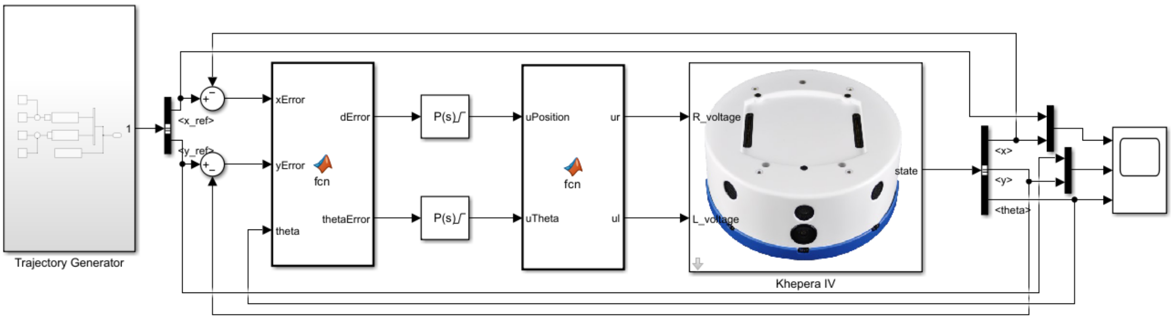


Figure 2.29. Model and controllers of Khepera IV in Simulink.

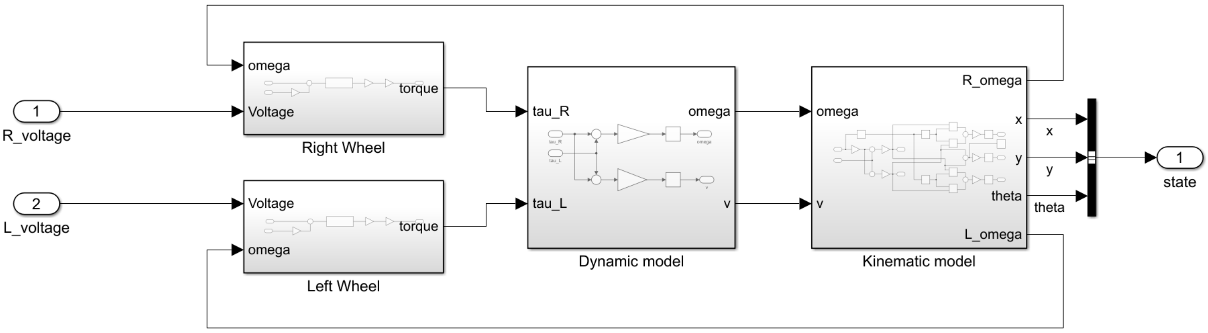


Figure 2.30. Physical model of Khepera IV in Simulink.

using the "Publish" block. For feedback to the controllers, the script subscribed to the topics "pose" and "twist", which are the position and velocity of the robot. Other useful blocks, such as "Read Data", allow the reading of raw recordings. Using standard message types in these cases is important since using custom messages is not very successful in Matlab/Simulink and uses old versions of Python and Visual Studio libraries. This system is suitable for small-scale testing and for those users who do not have programming knowledge in ROS 2.

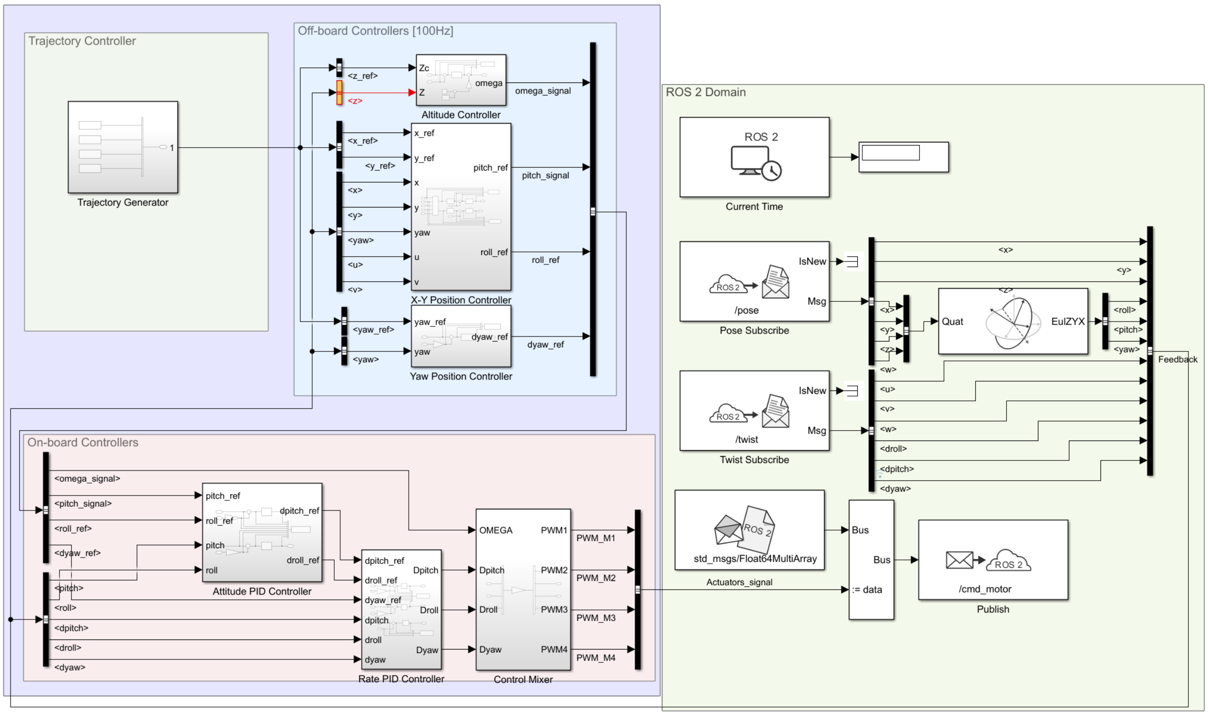


Figure 2.31. Example of using the Simulink script with ROS 2.

2.8.3 Gazebo

As already mentioned, Gazebo is the most widespread simulation tool in robotics. The current version will be the last of this tool, continuing in Gazebo Ignition. The first ROS 2 developments in Robotic Park have used this simulator for its integration into the community. However, the unavailability of plugins in ROS 2 to simulate drones without dependencies on third-party packages such as PX4 has limited its use. This fact has led to more frequent use of other platforms where the robots available in Robotic Park are officially and natively available in the most recent versions. However, it remains an useful tool for differential robot experiences due to its ease of use and scalability from ROS 2. In the following, the current use cases of Gazebo and the contributions made for each robot are detailed.

For the **Crazyflie**, it has been necessary to update the plugin from CrazyS project [149] (currently in ROS Melodic) to ROS Noetic. That project is an adaption to Crazyflie from RotorS project [150]. The libraries' dependencies have also been updated to the most recent version of Python, improving the execution of swarms, and a communication system for using DT with real robots has been developed. As it is shown in Figure 2.32,

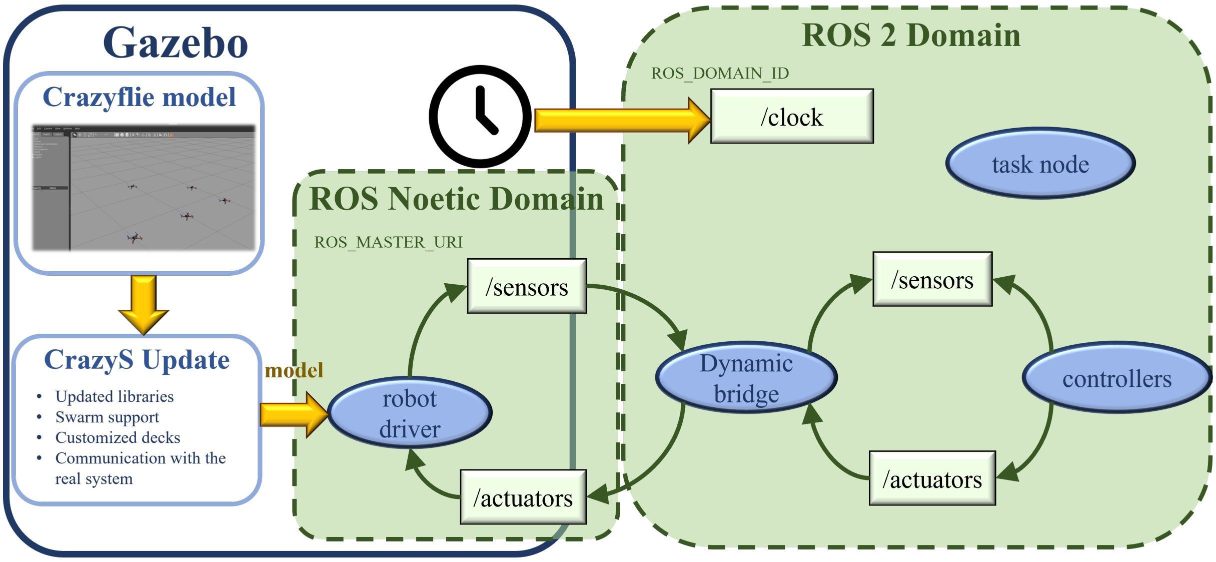
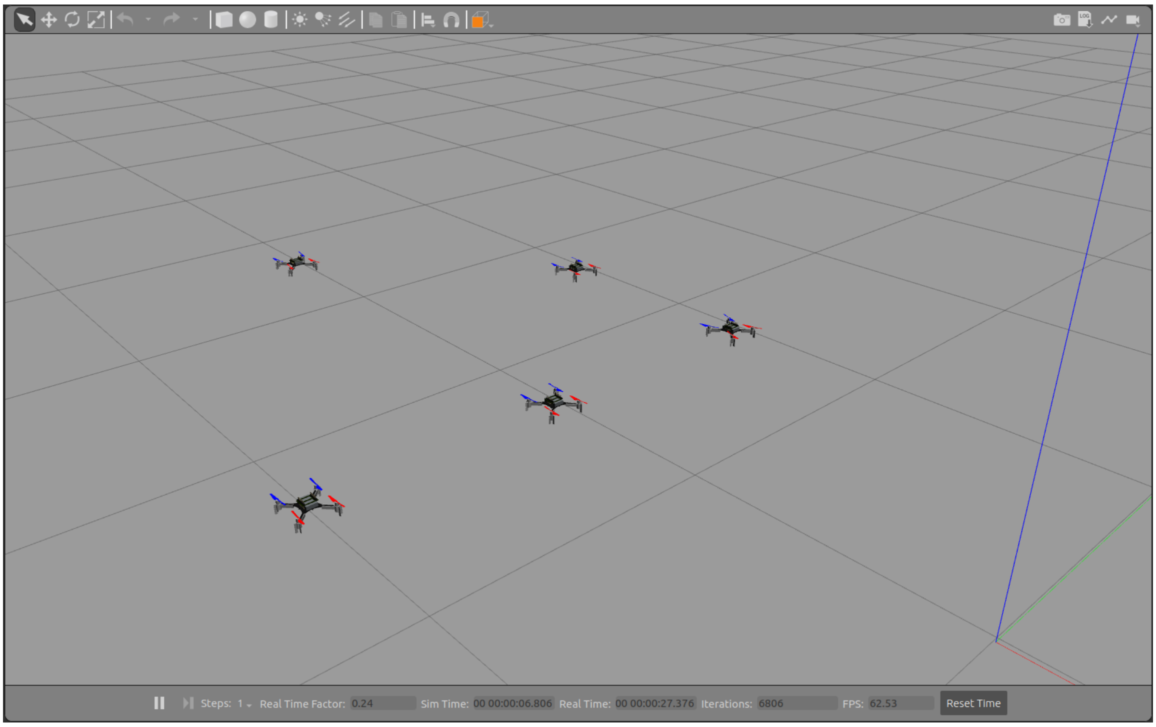


Figure 2.32. CrazyS project communication with ROS 2 domain.

Crazyflie runs on a ROS Noetic and uses a bridge for communication with the rest of the network in ROS 2. This bridge was designed for users who are migrating between systems or systems with components that cannot be updated. Figure 2.33 shows the result of running this implementation for the case of five agents. This implementation has certain factors that must be taken into account for proper operation. The most important is the ratio between real-time and simulated time. A swarm run, with all sensors active, can slow down the system. Therefore, it is important to use the "use_sim_time" parameter on the active nodes. In this way, all nodes will use the simulated clock for their operating frequency. On more recent implementations or more powerful computers, this ratio is more robust and allows for real-time-like execution. For this reason, adapting the system to other tools that offer equal results using fewer resources is necessary.



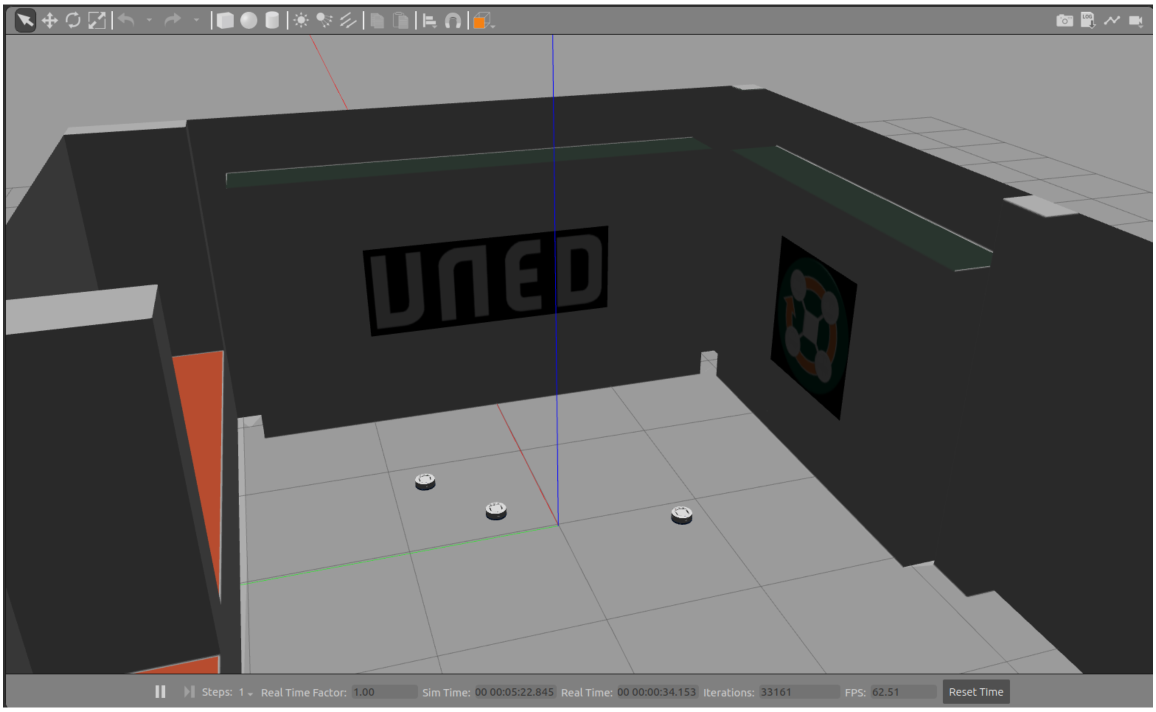
Steps: 1 Real Time Factor: 0.24 Sim Time: 00 00:00:06.806 Real Time: 00 00:00:27.376 Iterations: 6806 FPS: 62.53

Figure 2.33. Swarm of five Crazyflies 2.1 in Gazebo.

Khepera IV robots have traditionally used *CoppeliaSim* software (old *V-REP*), so to integrate them into RP and ROS 2, its complete model has been developed in SDF and URDF format with standard plugins and a driver node that includes the communication with the real robot. In this case, the necessary plugins for the simulation of the differential dynamic robots and their infrared, ultrasound, and camera sensors are supported by ROS 2. The main problem with Gazebo is the limitation of the number of sensors using the PC GPU. This issue translates into a maximum of four robots with all functional sensors. To overcome this limitation, it is important to ensure that only the necessary sensors for each experience are properly configured. Figure 2.34 shows a case where three virtual Khepera IV robots are combined with physical aerial robots to maintain a formation based on positions relative to each other [151].

2.8.4 Webots

As an alternative to the difficulties encountered in Gazebo, the Webots platform has been chosen as appropiarte to carry out MARS experiences. Despite having a much smaller impact on the developer community, in recent years, a great effort has been made to improve its positioning, adding many robots to its official libraries and developing the necessary plugins for its direct



Steps: 1 Real Time Factor: 1.00 Sim Time: 00 00:05:22.845 Real Time: 00 00:00:34.153 Iterations: 33161 FPS: 62.51

Figure 2.34. MRS of three Khepera IV in Gazebo.

compatibility with ROS 2. In addition, it has a user interface and a system that allows the use of programs developed in Python. These are very useful features to encourage experimentation in research teams with heterogeneous areas of knowledge. In this way, researchers who want to carry out tests on the platform's robots can previously validate them in simulation without needing specific knowledge of ROS 2.

In RP this simulator is used as the primary tool since it currently contains the models of all operating robots. The main contributions made in this simulator have been related to the ROS 2 node that connects the simulator with the rest of the network. It has been necessary to develop specific drivers for each robot with the same features as their firmware. Likewise, the necessary functions have been developed to establish real-time communication with real robots, allowing the carry-out experiences of MR to use virtual robots as DTs of the physical system. This simulator has proven to be an efficient tool in Mixed Reality experiments with greater efficiency and computational robustness compared to Gazebo [97]. Figure 2.35 shows the Webots simulation environment during a formation control experience applied to a MARS with Crazyflies 2.1 and Kheperas IV. Figure 2.36 shows the perspective of the RVIZ2 tool for the same experiment, where it is possible to visualize the distance specifications between robots (green lines), the trajectories followed by each robot (blue path), their current position (red arrows) and their desired positions (orange arrows).

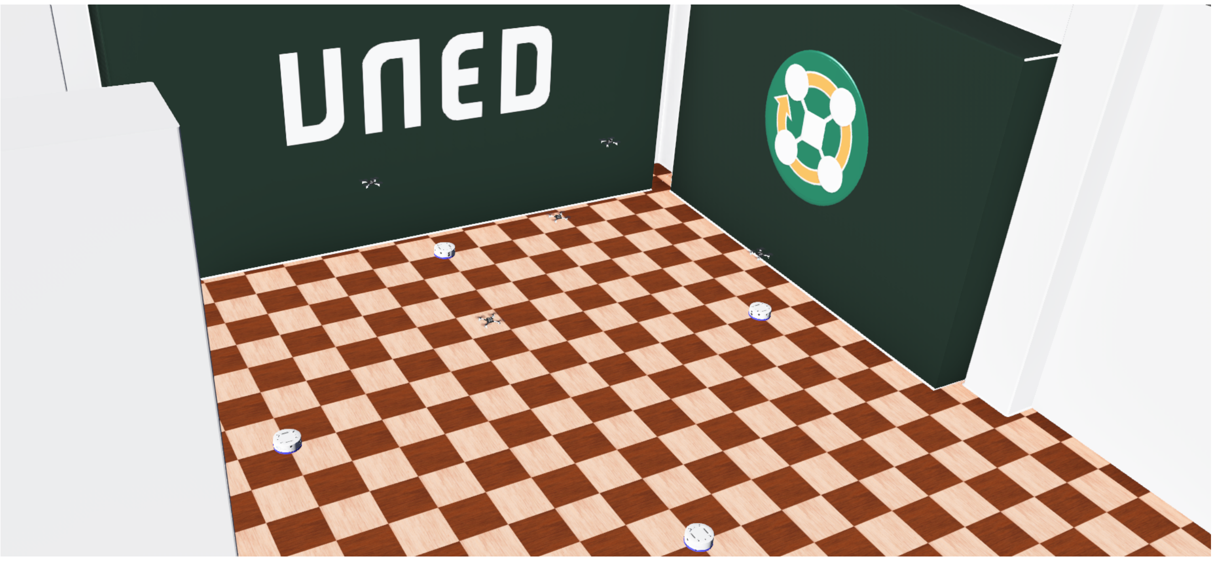
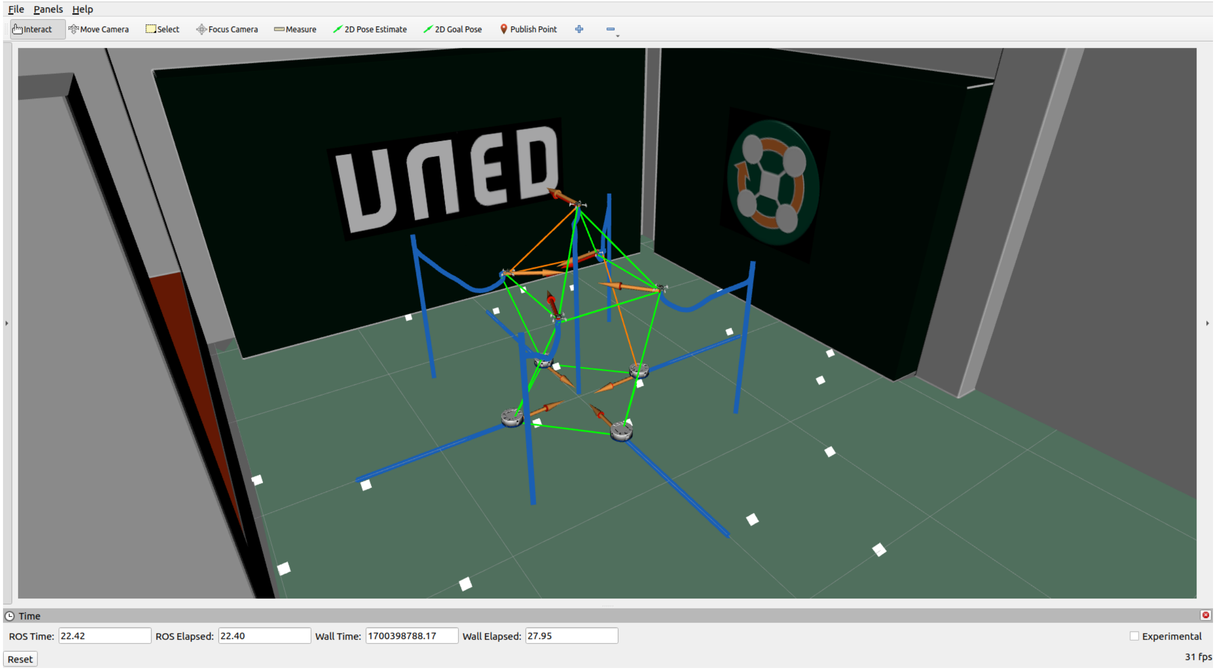


Figure 2.35. Robotic Park virtual environment at Webots in MARS experience.



ROS Time: 22.42 ROS Elapsed: 22.40 Wall Time: 1700398788.17 Wall Elapsed: 27.95 □ Experimental

Figure 2.36. RVIZ2 visualization in a MARS experience.

2.9 Conclusions

This chapter presents the heterogeneous experimental indoor platform for control and robotics Robotic Park to support MARS experiences. The hardware and software infrastructure of the system has been analyzed in detail, including the different types of robots (aerial, mobile), positioning systems (MoCap, UWB, Infrared, based on Vision), and communication architectures (centralized, distributed, or hybrid). The system is integrated in ROS 2 that interconnects all the elements. Several simulators have been developed and are integrated into the platform. The main highlights of the platform are its modularity and flexibility. Moreover, the system has been proven to offer a high degree of equipment and a wide range of control and robotics experiences. However, the goal is to establish the *Kaizen* philosophy in the platform development. This implies a commitment to continuous improvement while the platform is active. These improvements involve maintaining Robotic Park as an experimental platform at the forefront of MARS research.

To achieve this, some lines of future work concerning both hardware and software are proposed. On the one hand, including new robots with different locomotion systems could be an interesting topic that will open new research lines and the application of new cooperative tasks. On the other hand, at the software level, providing more autonomous navigation to the robots by making use of SLAM techniques under limited computation capabilities will be addressed. Additionally, studying some security aspects such as confidentiality and integrity of the experimental infrastructure when, for instance, the platform is operated remotely, will be considered.

Chapter 3

Differential Drive Mobile Robots

3.1 Introduction

Most of the available mobile robots in experimental platforms are wheel-based structures because of their efficiency and simple mechanical implementation. Particularly, the Differential Drive Wheeled Mobile Robot (DDWMR) is widely used.

A DDWMR is a mobile robot with movement usually based on two (four or six in some cases) separately driven wheels mounted on a common axis placed on either side of the robot body. They are non-holonomic robots, allowing only to move back and front yet not sideways. This is the main difference with omnidirectional robots which can drive in any direction, as the omni drive enables the robot to change direction instantly without having to turn the wheels. In this way, omnidirectional robots are beneficial when maneuvering in narrow spaces, which is not the case in our experimental platform. In addition, robots with differential drive are a good choice for thrust power as the wheels have individual motors with controlled speeds in each one. All these characteristics of the DDWMRs, along with their simpler motion programming and the fact that they are more easily controllable than other robots, make them appropriate for experiences in Robotic Park (RP).

Regarding modeling for DDWMRs different works are found in the literature [152]. However, most of them make only use of the robot's kinematic model when designing their motion controllers. Thus, the mathematical model used in RP's simulators consists of the kinematic model, the dynamic model which includes the mass and moment of inertia for each part of the robot and the DC motors. This complete model has been implemented in the Matlab simulator to allow evaluating changes in dynamics in possible experiences in which the physical characteristics of the robot change, for example, during a test with loads. In the case Gazebo and Webots tools, these modifications are solved by the physical engines. A complete knowledge of the robot model is also relevant to perform advanced tasks with Digital Twin (DT) and Digital Shadow (DS), such as preventive detection of hardware failures by detecting anomalous consumption in the actuators. In this thesis, as experiences are primarily focused on formation control and do not have access to the hardware drivers of the engines, the isolated kinematic model will be good enough to design the control structure.

Concerning the control, the number of control algorithms that have been used in the robot is equally extensive. Position control is usually performed using PID controllers due to its good performance and simplicity [153] Despite the good results obtained with this controller, more complex control techniques such as Linear-Quadratic Regulator (LQR) [154], nonlinear PID [155], robust control [156,157], event-based control [158], predictive control [159], intelligent control through neural networks [160], genetic algorithms [161], fuzzy logic [162], or combinations of them [163] have been applied. Such a variety of control algorithms allows the designer to make comparisons and determine the most appropriate architecture for each implementation.

Thus, this chapter mainly focuses on the modeling and local controllers implemented in Robotic Park for the DDWMR. Section 3.2 presents a detailed analysis of their modeling, first presenting the theoretical models (kinematic and dynamic) of the robots and then the external and internal parameterized representation of the robots. In Section 3.3, the process of identifying the theoretical parameters of the robots is carried out based on experimental data from real robots. Section 3.4 presents the proposed control strategies for the position and orientation control of the robot. Section 3.5 presents the experimental validation of the operating robots in Robotic Park. Finally, a summary of the main results obtained is presented.

3.2 Theoretical model

3.2.1 Kinematic Model

The kinematic model determines the motion of the robot from its geometrical characteristics and constraints. The main parameters of these robots are shown in Figure 3.1, where (lw) and (rw) are the left and right wheels respectively separated a distance d, Q represents the center of gravity, v_Q is the linear velocity, and $\dot{\theta}$ the wheels' angular velocities. The YX coordinate system is a fixed global reference system. (x_Q, y_Q) and φ represent the position and orientation respectively in the global coordinate system. The $Y_R X_R$ coordinate system is a local coordinate system of the robot with origin at Q. The following hypotheses are established for the modeling of the system:

- The wheels rotate on the surface without sliding effects.
- The axis of rotation of the robot is perpendicular to the X-Y plane.
- The center of gravity is located at point Q (see Figure 3.1).

Assuming the absence of wheel slip, the velocities of each wheel are the following:

$$\begin{pmatrix} v_r \\ v_l \end{pmatrix} = r \cdot \begin{pmatrix} \dot{\theta}_r \\ \dot{\theta}_l \end{pmatrix} \tag{3.1}$$

where r is the wheel's radius. From these values, the forward velocity, v_Q, and the angular velocity, $\dot{\varphi}$ of the center of gravity of the robot are determined according to (3.2).

$$\begin{pmatrix} v_Q \\ \dot{\varphi} \end{pmatrix} = \begin{pmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{d} & -\frac{r}{d} \end{pmatrix} \begin{pmatrix} \dot{\theta}_r \\ \dot{\theta}_l \end{pmatrix} \tag{3.2}$$

The global velocities of the robot in the X-Y plane and the angular velocity is calculated as follows:

$$\begin{pmatrix} \dot{x}_Q \\ \dot{y}_Q \\ \dot{\varphi} \end{pmatrix} = \begin{pmatrix} \cos\varphi & 0 \\ \sin\varphi & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v_Q \\ \dot{\varphi} \end{pmatrix} = \begin{pmatrix} \frac{r \cdot \cos\varphi}{2} & \frac{r \cdot \cos\varphi}{2} \\ \frac{r \cdot \sin\varphi}{2} & \frac{r \cdot \sin\varphi}{2} \\ \frac{r}{d} & -\frac{r}{d} \end{pmatrix} \begin{pmatrix} \dot{\theta}_r \\ \dot{\theta}_l \end{pmatrix} \tag{3.3}$$

These equations are known as the **Forward Kinematics Model for Differential Drive Robots** and are expressed in simplified form as

$$\dot{\mathbf{p}} = \mathbf{J}\dot{\mathbf{q}} \tag{3.4}$$

where $\dot{\mathbf{p}} = \begin{pmatrix} \dot{x}_Q \\ \dot{y}_Q \\ \dot{\varphi} \end{pmatrix}$, $\dot{\mathbf{q}} = \begin{pmatrix} \dot{\theta}_r \\ \dot{\theta}_l \end{pmatrix}$ and $\mathbf{J}$ is the Jacobian matrix of differential drive robot.

Since the Jacobian matrix is not invertible, to obtain $\dot{\mathbf{q}}$ in the forward kinematics model for differential drive robots, the pseudoinverse (Moore-Penrose) matrix is used, which establishes its value according to the following expression:

$$\mathbf{J}^{+} = \left(\mathbf{J}^T\mathbf{J}\right)^{-1}\mathbf{J}^T \tag{3.5}$$

where $\mathbf{J}^{+}$ represents the pseudoinverse matrix. Another way to calculate $\mathbf{J}^{+}$ is from the equations that link the linear velocity of each wheel to the velocity of the center of gravity, resulting,

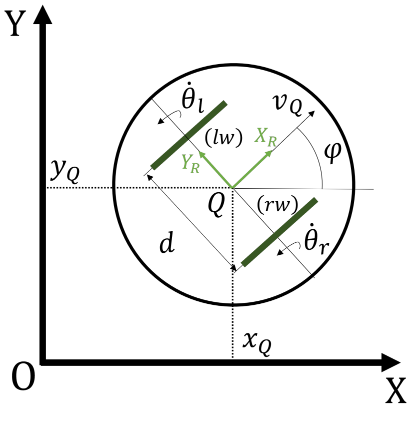


Figure 3.1. Diagram of a generic differential drive robot.

$$
\begin{aligned}
r\dot{\theta}_r &= \dot{x}_Q \cos(\varphi) + \dot{y}_Q \sin(\varphi) + \tfrac{d}{2}\dot{\varphi} \\
r\dot{\theta}_l &= \dot{x}_Q \cos(\varphi) + \dot{y}_Q \sin(\varphi) - \tfrac{d}{2}\dot{\varphi}
\end{aligned} \tag{3.6}
$$

Thus, the **Inverse Kinematics Model for Differential Drive Robots** method is obtained as

$$
\dot{\mathbf{q}} = \mathbf{J}^{+}\dot{\mathbf{p}} \rightarrow \begin{pmatrix} \dot{\theta}_r \\ \dot{\theta}_l \end{pmatrix} = \frac{1}{r}\begin{pmatrix} \cos(\varphi) & \sin(\varphi) & \frac{d}{2} \\ \cos(\varphi) & \sin(\varphi) & -\frac{d}{2} \end{pmatrix}\begin{pmatrix} \dot{x}_Q \\ \dot{y}_Q \\ \dot{\varphi} \end{pmatrix} \tag{3.7}
$$

And the instantaneous radius of curvature of the robot can be calculated as

$$
R = \frac{v_Q}{\dot{\varphi}} = \frac{d}{2}\frac{v_r + v_l}{v_r - v_l} \tag{3.8}
$$

3.2.2 Dynamic Model

Most applications involving differential drive mobile robots perform the control on the kinematic model, as it provides good results using the linear and angular velocities of the robot as control signals. However, if more accurate results are required, it is necessary to include the dynamic model of the system. The dynamic model studies the effect of the forces to which the robot is subjected in its motion. The model obtained is of interest for the analysis of both longitudinal and lateral sliding. Dynamic modeling makes use the Newton-Euler method. The translational motion of the robot is given by

$$
m \cdot \dot{v_Q} = F \tag{3.9}
$$

where F is the force applied at the center of gravity Q, v_Q is the linear velocity, and m is the mass of the robot. The rotation of the robot is defined by the expression

$$
I \cdot \dot{\omega}_Q = M \tag{3.10}
$$

where M is the angular momentum with respect to the center of gravity, ω_Q is the angular velocity, and I is the moment of inertia of the robot. The forces and moments generated by the wheels are related to each other according to the following expression

$$
\begin{aligned}
F &= F_r + F_l \\
\tau_{r,l} &= r \cdot F_{r,l}
\end{aligned} \tag{3.11}
$$

where $F_{r,l}$ and $\tau_{r,l}$ represent the forces and moments generated in each wheel by the action of the motors attached to it. Linking the forces and moments to the geometrical components of the robot we obtain the following expressions

$$
F = \frac{1}{r}(\tau_r + \tau_l) \tag{3.12}
$$

$$M = (F_r + F_l) \cdot d = \frac{d}{r}(\tau_r - \tau_l) \tag{3.13}$$

Finally, by replacing these F and M values in the equations of motion the **Dynamic Model of the Differential Drive Robot** is obtained as follows:

$$\begin{pmatrix} \dot{v} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} \frac{1}{m \cdot r} & \frac{1}{m \cdot r} \\ \frac{d}{I \cdot r} & \frac{-d}{I \cdot r} \end{pmatrix} \begin{pmatrix} \tau_r \\ \tau_l \end{pmatrix} \tag{3.14}$$

3.2.3 Direct current motor

The lowest level of the differential drive robot model studies the dynamics of the DC motors. These systems have as input signal the control voltage and their output is the torque that is generated in the motor. In a DC motor, the torque is produced by the existence of an electric current because there is no balance between its electromotive forces. The mechanical equation of the motor is given by

$$J_m \ddot{\theta} = \tau - \dot{\theta} \tag{3.15}$$

where J_m is the motor inertia, θ is the motor shaft rotation angle, and τ is the rotor mechanical torque. The mechanical torque has a proportional relationship with the motor armature current i, the characteristic torque constant of the motor k_t, and the gear ratio N according to

$$\tau = i \cdot N \cdot k_t \tag{3.16}$$

Therefore,

$$J_m \ddot{\theta} = N \cdot k_t \cdot i - \dot{\theta} \tag{3.17}$$

Applying Kirchoff's law to the motor in Figure 3.2 gives the differential equation of the system

$$R \cdot i + L \cdot \dot{i} = v - e_c = v - k_b \cdot \dot{\theta} \tag{3.18}$$

where v is the motor control voltage, i is the armature current, R is its characteristic resistance,

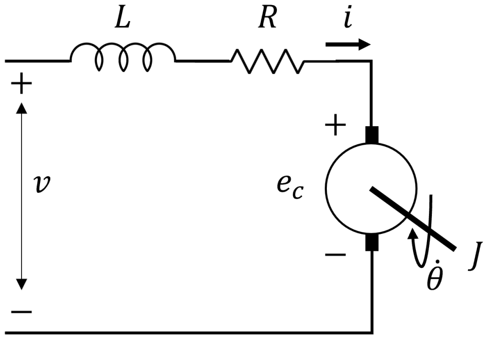


Figure 3.2. Electrical diagram of DC motor.

L is the rotor inductance, and e_c is the counter-electromotive force produced in the motor by its instantaneous rotational speed. This last force depends on the rotational speed and a characteristic constant of the motor, k_b.

3.2.4 State space model

Once the individual models of the different parts that make up the system from the control signal received by the actuators to the overall response of the system position have been obtained, the global model based on the internal description is developed. To this end, the *state vector* of the DDWMR, is defined as

$$\mathbf{x} = \begin{bmatrix} x_Q & y_Q & \varphi & v_Q & \omega_Q & \dot{\theta}_r & i_r & \dot{\theta}_l & i_l \end{bmatrix}^T \tag{3.19}$$

and the signals received by the actuators are selected as control signals as follows

$$\mathbf{u} = \begin{bmatrix} v_l & v_r \end{bmatrix}^T \tag{3.20}$$

where v_r is the input voltage of the right motor and v_l is the input voltage of the left motor.

In this way, the direct kinematic model governing the dynamics of the system is nonlinear and can be represented as

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \\ \mathbf{y}(t) &= \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t)) \end{aligned} \tag{3.21}$$

The Taylor serie linearizes the system's response as (3.22) shows by deleting second and higher-order terms.

$$f(\mathbf{x}, \mathbf{u}) \approx f(\mathbf{x}_{eq}, \mathbf{u}_{eq}) + \left(\left. \frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} \right|_{(\mathbf{x}_{eq}, \mathbf{u}_{eq})} (\mathbf{x} - \mathbf{x}_{eq}) + \left. \frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} \right|_{(\mathbf{x}_{eq}, \mathbf{u}_{eq})} (\mathbf{u} - \mathbf{u}_{eq}) \right) \tag{3.22}$$

The linearization the equations of the forward kinematics model is shown below, where the rotational speeds of each motor in a steady state, $\left(\dot{\theta}_{r,eq}, \dot{\theta}_{l,eq}\right)$, are considered zero.

$$\begin{aligned} \dot{x}_Q =& \frac{r}{2} \cos\varphi_{eq} \left(\dot{\theta}_{r,eq} + \dot{\theta}_{l,eq} - \frac{\sin\varphi_{eq}}{\cos\varphi_{eq}} \left(\dot{\theta}_{r,eq} + \dot{\theta}_{l,eq} \right) (\varphi - \varphi_{eq}) + \dot{\theta}_r - \dot{\theta}_{r,eq} + \dot{\theta}_l - \dot{\theta}_{l,eq} \right) \\ \dot{x}_Q =& \frac{r}{2} \cos\varphi_{eq} \dot{\theta}_r + \frac{r}{2} \cos\varphi_{eq} \dot{\theta}_l \end{aligned} \tag{3.23}$$

$$\begin{aligned} \dot{y}_Q =& \frac{r}{2} \sin\varphi_{eq} \left(\dot{\theta}_{r,eq} + \dot{\theta}_{l,eq} + \frac{\cos\varphi_{eq}}{\sin\varphi_{eq}} \left(\dot{\theta}_{r,eq} + \dot{\theta}_{l,eq} \right) (\varphi - \varphi_{eq}) + \dot{\theta}_r - \dot{\theta}_{r,eq} + \dot{\theta}_l - \dot{\theta}_{l,eq} \right) \\ \dot{y}_Q =& \frac{r}{2} \sin\varphi_{eq} \dot{\theta}_r + \frac{r}{2} \sin\varphi_{eq} \dot{\theta}_l \end{aligned} \tag{3.24}$$

$$\dot{\varphi} = \frac{r}{d}\dot{\theta}_r - \frac{r}{d}\dot{\theta}_l \tag{3.25}$$

After the linearization of this system, its state space model can be represented as follows:

$$\begin{aligned} \dot{\mathbf{x}}\left(t\right) &= \mathbf{A}\mathbf{x}\left(t\right) + \mathbf{B}\mathbf{u}\left(t\right) \\ \mathbf{y}\left(t\right) &= \mathbf{C}\mathbf{x}\left(t\right) + \mathbf{D}\mathbf{u}\left(t\right) \end{aligned} \tag{3.26}$$

where $\mathbf{x}\left(t\right) : \mathbb{R}^n$ is the state vector, $\mathbf{y}\left(t\right) : \mathbb{R}^m$ is the output vector, $\mathbf{u}\left(t\right) : \mathbb{R}^r$ is the input vector, $\mathbf{A} : \mathbb{R}^{n\times n}$ is the state matrix, $\mathbf{B} : \mathbb{R}^{n\times r}$ is the input matrix, $\mathbf{C} : \mathbb{R}^{m\times n}$ is the output matrix, and $\mathbf{D} : \mathbb{R}^{m\times r}$ is the direct transmission matrix. Replacing (3.3), (3.14), (3.17), and (3.18), the global model is built as follows

$$\begin{aligned} \begin{pmatrix} \dot{x}_Q \\ \dot{y}_Q \\ \dot{\varphi} \\ \dot{v}_Q \\ \dot{\omega}_Q \\ \ddot{\theta}_r \\ \dot{i}_r \\ \ddot{\theta}_l \\ \dot{i}_l \end{pmatrix} &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & \frac{r}{2}\cos\varphi_{eq} & 0 & \frac{r}{2}\cos\varphi_{eq} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{r}{2}\sin\varphi_{eq} & 0 & \frac{r}{2}\sin\varphi_{eq} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{r}{d} & 0 & \frac{r}{d} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{Nk_t}{mr} & 0 & \frac{Nk_t}{mr} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{Nk_t d}{Ir} & 0 & -\frac{Nk_t d}{Ir} \\ 0 & 0 & 0 & 0 & 0 & -\frac{1}{J_m} & \frac{Nk_t}{J_m} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{k_b}{L} & -\frac{R}{L} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{J_m} & \frac{Nk_t}{J_m} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{k_b}{L} & -\frac{R}{L} \end{pmatrix} \begin{pmatrix} x_Q \\ y_Q \\ \varphi \\ v_Q \\ \omega_Q \\ \dot{\theta}_r \\ i_r \\ \dot{\theta}_l \\ i_l \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{L} & 0 \\ 0 & 0 \\ 0 & \frac{1}{L} \end{pmatrix} \begin{pmatrix} v_r \\ v_l \end{pmatrix} \\ \begin{pmatrix} x_{Q,k} \\ y_{Q,k} \\ \varphi_k \end{pmatrix} &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_Q \\ y_Q \\ \varphi \\ v_Q \\ \omega_Q \\ \dot{\theta}_r \\ i_r \\ \dot{\theta}_l \\ i_l \end{pmatrix} \end{aligned} \tag{3.27}$$

The model described in (3.27) defines the complete behavior of the system around an operating point. However, in most cases, when controlling real robots, the control of DC motors is done through the firmware, hiding this stage of the process through an API that only receives as setpoints the linear and angular target speeds. As already mentioned, this means that in practice we only work with the approximation of the kinematic model taking the mentioned velocities as control signals, (3.28) and (3.29). This approximation is suitable because the system's dynamic and the motor are sufficiently fast concerning the kinematic model. Furthermore, in many commercial applications, the physical parameters of the motors are not available to protect design patents.

$$\begin{pmatrix} \dot{x}_Q \\ \dot{y}_Q \\ \dot{\varphi} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_Q \\ y_Q \\ \varphi \end{pmatrix} + \begin{pmatrix} \frac{r \cdot \cos\varphi_{eq}}{2} & \frac{r \cdot \cos\varphi_{eq}}{2} \\ \frac{r \cdot \sin\varphi_{eq}}{2} & \frac{r \cdot \sin\varphi_{eq}}{2} \\ \frac{r}{d} & -\frac{r}{d} \end{pmatrix} \begin{pmatrix} \dot{\theta}_r \\ \dot{\theta}_l \end{pmatrix} \tag{3.28}$$

$$\begin{pmatrix} x_Q \\ y_Q \\ \varphi \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_Q \\ y_Q \\ \varphi \end{pmatrix} \tag{3.29}$$

Once the internal representation has been defined in non-linear and linear form, the external representation of the simplified model of the system is extracted using transfer functions, what result as follows:

$$G(s) = C[sI - A]^{-1}B + D \tag{3.30}$$

$$\begin{bmatrix} G_{x_Q}(s) \\ G_{y_Q}(s) \\ G_{\varphi}(s) \end{bmatrix} = \begin{bmatrix} \frac{r \cdot \cos\varphi_{eq}}{2s} & \frac{r \cdot \cos\varphi_{eq}}{2s} \\ \frac{r \cdot \sin\varphi_{eq}}{2s} & \frac{r \cdot \sin\varphi_{eq}}{2s} \\ \frac{r}{ds} & -\frac{r}{ds} \end{bmatrix} \tag{3.31}$$

3.2.5 Motion Model

In order to obtain the robot's motion model it is considered the YX inertial reference frame and the $Y_R X_R$ reference frame located at the center of gravity of the robot Q, $\dot{\mathbf{p}} = (\dot{x}_Q, \dot{y}_Q, \dot{\varphi}_Q)^T$ defines the velocities concerning the inertial frame and $\dot{\mathbf{p}}_l = (\dot{x}_l, \dot{y}_l, \dot{\varphi}_l)^T$ the velocities of the robot in the local frame. The relationship between the two frames is defined by the homogeneous transformation matrix $\mathbf{R}(\varphi)$ given by (3.32).

$$\dot{\mathbf{p}}_l = \mathbf{R}(\varphi)\dot{\mathbf{p}} = \begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} \dot{\mathbf{p}} \tag{3.32}$$

As there is no lateral displacement in the robot, there is a non-holonomic constrain given by (3.33).

$$\dot{x}_Q \sin\varphi - \dot{y}_Q \cos\varphi = 0 \tag{3.33}$$

If the robot goes from an initial position $A\,(x_A, y_A, \varphi_A)$ to a target position $B\,(x_B, y_B, \varphi_B)$, see Figure 3.3, and the parameters considered to govern the motion of the robot are the angular velocities of the motor wheels, $\dot{\theta}_{r,l}$, the dynamic equations of motion can be written in polar form as:

$$\rho = \sqrt{(y_B - y_A)^2 + (x_B - x_A)^2} \tag{3.34}$$

$$\alpha = \operatorname{atan2}(y_B - y_A, x_B - x_A) - \varphi_A \tag{3.35}$$

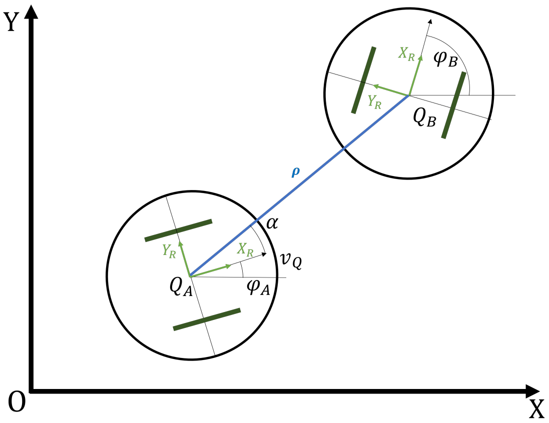


Figure 3.3. Diagram of the motion of a differential drive robot.

where ρ is the distance between the goal point and the initial position and α is the error in the orientation.

These equations imply that

$$\begin{aligned} y_B - y_A &= \rho \sin(\alpha + \varphi_A) \\ x_B - x_A &= \rho \cos(\alpha + \varphi_A) \end{aligned} \tag{3.36}$$

Therefore, the equations modeling the dynamic behavior of the robot's movement can be expressed as follows

$$\begin{gathered} \dot{\rho} = \tfrac{1}{2}\tfrac{-2\dot{y}_A(y_B-y_A)-2\dot{x}_A(x_B-x_A)}{\rho} = -\tfrac{v\sin(\varphi)\rho\sin(\alpha+\varphi_A)+v\cos(\varphi)\rho\cos(\alpha+\varphi_A)}{\rho} \\ \dot{\rho} = -v\cos(\alpha) = -\tfrac{r}{2}\left(\dot{\theta}_r + \dot{\theta}_l\right)\cos(\alpha) \end{gathered} \tag{3.37}$$

$$\begin{gathered} \dot{\alpha} = \tfrac{-(x_B-x_A)\dot{y}+(y_B-y_A)\dot{x}}{(y_B-y_A)^2+(x_B-x_A)^2} - \dot{\varphi}_A = \tfrac{-\rho\cos(\alpha+\varphi_A)v\sin(\varphi)+\rho\sin(\alpha+\varphi_A)v\cos(\varphi)}{\rho^2} - \omega \\ \dot{\alpha} = \tfrac{r}{2}\left(\dot{\theta}_r + \dot{\theta}_l\right)\tfrac{\sin(\alpha)}{\rho} - \tfrac{r}{d}\left(\dot{\theta}_r - \dot{\theta}_l\right) \end{gathered} \tag{3.38}$$

3.3 Parameters identification

Once the model of the robot has been determined, a set of experimental tests is carried out to verify the correspondence between the theoretical and empirical parameters. To estimate the parameters of the motion model, i.e. the radius of the wheels and the distance between them, an experience, shown in Figure 3.4, is performed. The robot moves along the path shown in the figure, starting at the red dot and ending at the green dot. First, the speed profiles from the odometry and the Vicon positioning system are extracted. Figure 3.5a shows the linear velocity profile of the robot measured with the Vicon system. Figure 3.5b shows the angular speed of the robot according to the Vicon system and the angular velocity of both wheels.

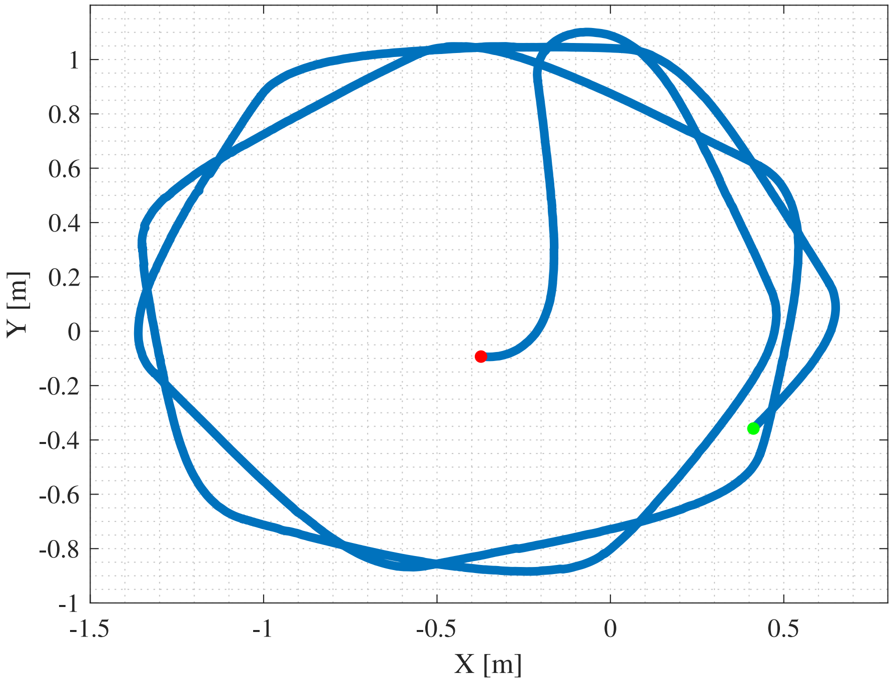
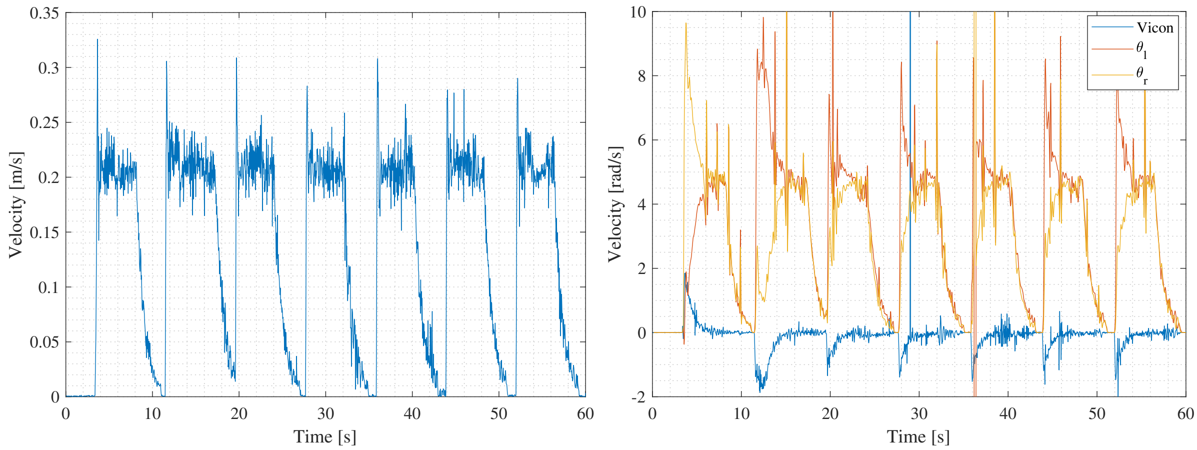


Figure 3.4. Trajectory followed by DDWMR for its characteristic parameters estimation.



(a) Linear.

(b) Angular.

Figure 3.5. Velocity profiles for identification experiment.

The radius of the wheels, r, is determined by (3.37) using the linear velocity of the robot, $\dot{\rho}$, and the angular velocity of each wheel, $\dot{\theta}_{r,l}$. Similarly, the distance between wheels d is calculated by (3.38) using the angular velocity of the robot, $\dot{\alpha}$, the turning speed of each wheel, $\dot{\theta}_{r,l}$, and the radius previously calculated. Figure 3.6 shows the results obtained. It is observed that the estimates ($r = 0.027$ $[m]$ and $d = 0.1019$ $[m]$) are close to the ground truth ($r = 0.021$ $[m]$ and $d = 0.105$ $[m]$) at the times when the speed of each wheel is higher. The reason is that, in

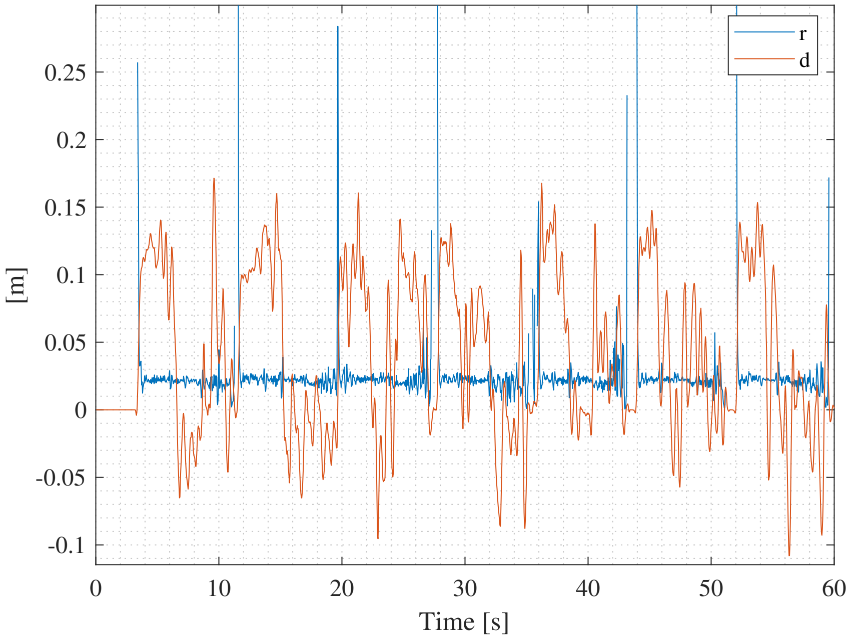


Figure 3.6. Wheels' radious ,r, and distance between them, d.

those moments, the signal-to-noise ratio is higher and, therefore, the results are more accurate. When the velocities are minimal, the signal-to-noise ratio increases, affecting the value of the estimate. Thus, the estimated final values have been extracted from the median values of the records during the 4 $[s]$ in which the robot's speed approaches 0.2 $[m/s]$.

In addition, a new test to determine the transfer function in a closed loop of linear and angular velocity has been carried out. In this case, an amplitude and random period signal for the robot's speed commands have been generated. Figure 3.7 shows the path described in the XY plane in blue, starting at the red dot and ending at the green dot.In this case, an amplitude and random period signal for the robot's speed commands have been generated. The recursive least square technique for a first-order model to estimate both models have been used. This technique has been implemented in a ROS 2 node that allows developing online identification techniques.

Figure 3.8 shows the results obtained in the case of linear velocity. Figure 3.8a shows the temporal response to the input signal and the estimation of the response at each instant. Figure 3.8b shows the evolution of the characteristic parameters of a first-order function in discrete time. The results obtained in discrete time are $a = 0.3882$ and $b = 0.6162$. In the Laplace domain, these results involve a stationary gain of $k = 1.0071$ and a time constant $\tau = 0.0217$ $[s]$.

The same process is repeated in the case of angular velocity, see Figure 3.9. Figure 3.9a shows the temporal evolution of the response for the input signal and the estimation of the response at each instant. Figure 3.9b shows the evolution of the characteristic parameters of the first-order function in discrete time. The results obtained in discrete time are $a = 0.3600$

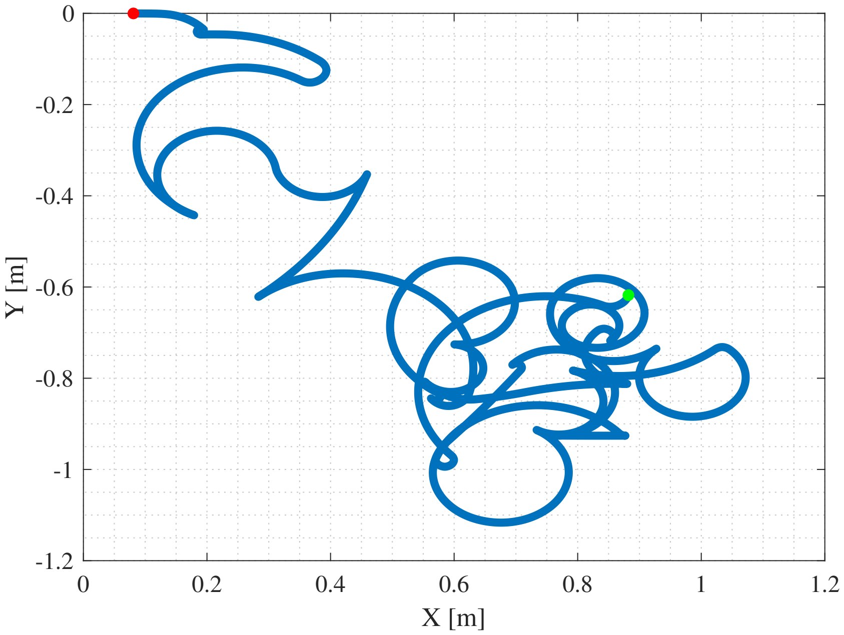
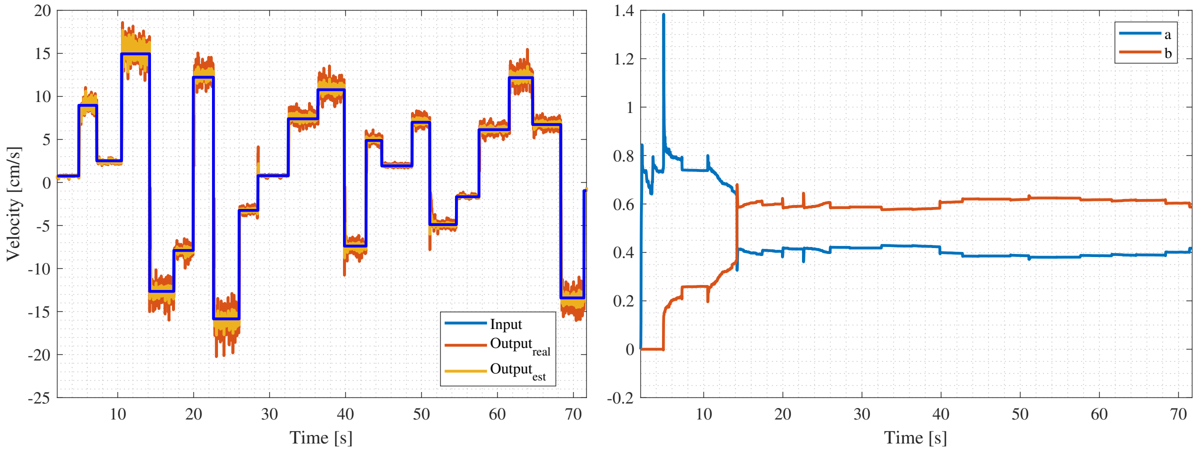


Figure 3.7. Trajectory followed by DDWMR for identification process.

and $b = 0.192$. In the Laplace domain, these results involve a stationary gain of $k = 0.9675$ and a time constant $\tau = 0.0201$ $[s]$.

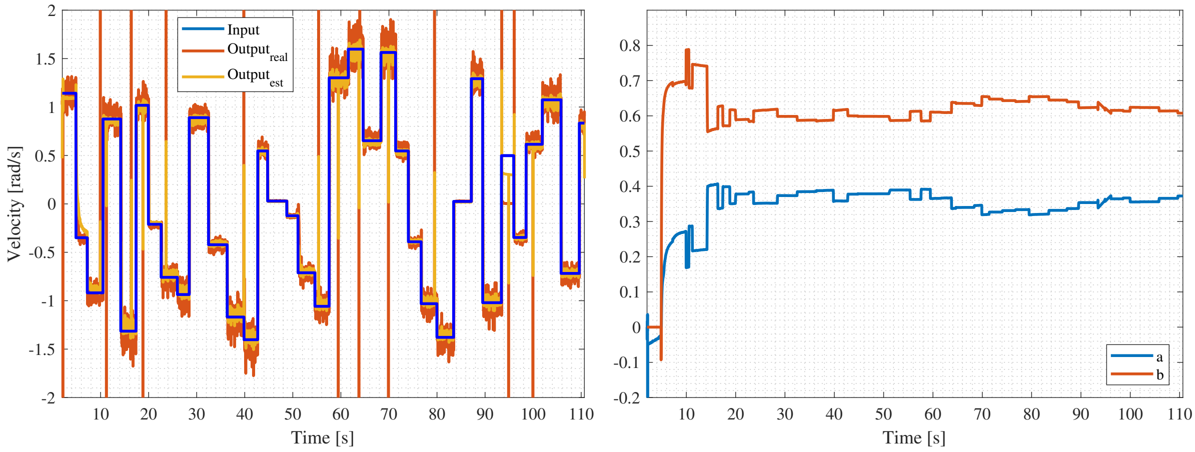
According to the results obtained and considering that the sampling time is $T = 0.02$ $[s]$, the kinematic model approximation is good enough when the robot is in movement without load.



(a) Temporal response and output estimation.

(b) Parameter estimation.

Figure 3.8. Linear velocity model identification.



(a) Temporal response and output estimation.

(b) Parameter estimation.

Figure 3.9. Angular velocity model identification.

3.4 Control architecture

A typical motion control architecture of these robots to implement (3.37) and (3.38) is made up of a single level with two controllers: a position controller, responsible of the translation control, and an orientation controller, as shown in Figure 3.10. The feedback signal of this control level is the position of the robot, p_i, and its control signal is the linear, v_i, and angular, ω_i, velocity commands. It is necessary a first function to transform the pose value from global to local frame. The outputs of this function are the distance error, $e_{distance}$, and the yaw angle error, E_{yaw}. Obstacle avoidance is also implemented at this level. All robots on the platform have obstacle avoidance implemented by artificial potentials [164].

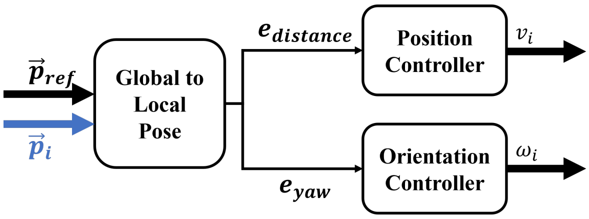


Figure 3.10. Diagram of the typical control architecture of a differential drive robot.

If the motion control problem is considered as a regulation problem for a Multiple-Input Multiple-Output (MIMO) system, the transfer functions governing the behavior of the ρ and α in (3.37) and (3.38) can be obtained using the robot velocities as control signals. Taking into account (3.30), these functions are defined by

$$\begin{bmatrix} G_\rho(s) \\ G_\alpha(s) \end{bmatrix} = \begin{bmatrix} \frac{-\cos\alpha}{s} & 0 \\ \frac{\sin(\alpha)}{\rho s} & -\frac{1}{s} \end{bmatrix} \tag{3.39}$$

Controller design

As it is an integral type process (see (3.39)), the best controller is one of proportional type due to its simplicity, ensuring zero error against step signals and a closed-loop first-order response. The resulting control law is as follows

$$\begin{aligned} v &= K_{\rho}\rho\cos(\alpha) \\ \omega &= K_{\rho}\sin(\alpha)\cos(\alpha) + K_{\alpha}\alpha \end{aligned} \tag{3.40}$$

where K_{ρ} is the gain of control over the forward velocity, and K_{α} is the gain of control over the angular velocity concerning the error of α.

In [165], improvements to this control law (IPC) are proposed that establish a maximum speed of motion when the Euclidean distance error is above a threshold and an integral component in the angular velocity control that overcomes limits of the previous law when the target position is at the rear of the robot. If the target position is 180° to the robot's current orientation, and the robot cannot move backward or does not have sensors to avoid collisions in that direction, the robot may be "locked". The law proposed, in this case, responds to

$$\begin{aligned} v &= \min\{K_1 dp(\alpha), v_{\max}\} \\ \omega(t) &= K_p\sin(\alpha(t)) + K_i\int_0^t \alpha(s)ds \end{aligned} \tag{3.41}$$

where

$$p(x) = \left(\frac{\pi - |x|}{\pi}\right) \tag{3.42}$$

and the parameters K_1,K_p, and $K_i > 0$ are chosen to satisfy the following constraints

$$0 < K_1 < K_p \tag{3.43}$$

$$K_p + \sqrt{K_i}\pi < \omega_{\max} \tag{3.44}$$

The detailed analysis demonstrating the asymptotic stability of the process and its corresponding validation in simulation can be found in [166]. Given the improvement in robustness that this implementation allows against the base controller, this controller is chosen to be implemented in Robotic Park's real robots.

3.5 Experimental evaluation and discussion

Once the theoretical and experimental models and the control architecture of DDWMR have been described, in this section the experimental validation on real robots is carried out. To this end, a scenario is proposed in which the robot follows a pseudo-random trajectory that generates random target points within the platform arena. The IPC control strategy [165] will be compared with a classical PID controller to evaluate its effectiveness. Figure 3.11 shows the obtained trajectories with both controllers. Figure 3.12 shows the path-tracking errors over time, where the better performance obtained with the IPC controller can be observed. The exponential dynamics of the PID response causes a faster response when the error is greater. However, when the error is small, the response slows down significantly compared to the IPC. Therefore, to integrate these robots into RP a suitable low-level control implementation is IPC.

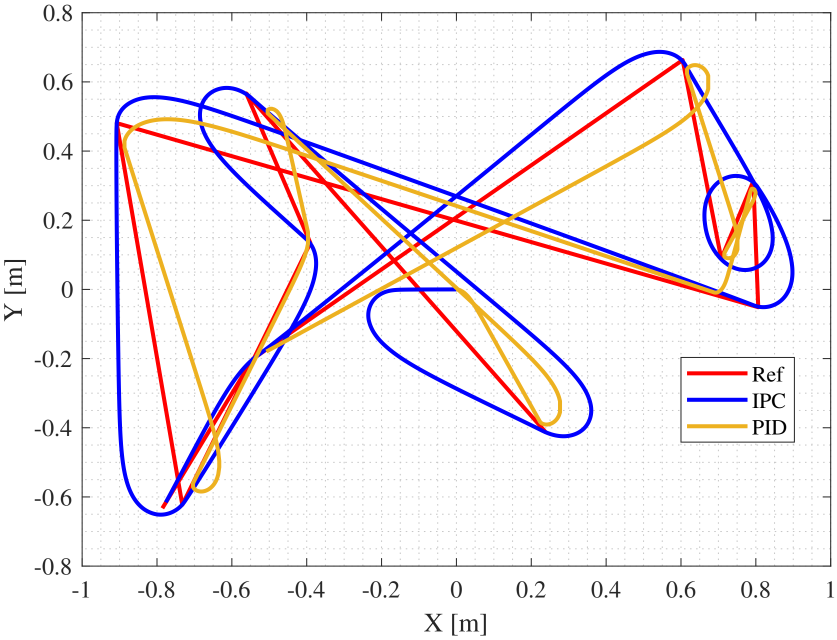


Figure 3.11. Trajectory followed by Khepera IV with different controllers.

3.6 Conclusions

This chapter describes the complete mathematical model of the differential drive mobile robot, the ground robots included in RP. The kinematic and the dynamic models have been presented, both the theoretical models and the external and internal parameterized representation. The parameters of the motion model have been validated from the real robots' experimental data. The inclusion of this model in RP ensures that it is prepared to carry out experiences in a wide spectrum of tasks.

A typical control architecture has been presented and a low-level controller for DDWMR has been experimentally evaluated. This evaluation has been carried out on real robots showing a proper integration of DDWMR in RP.

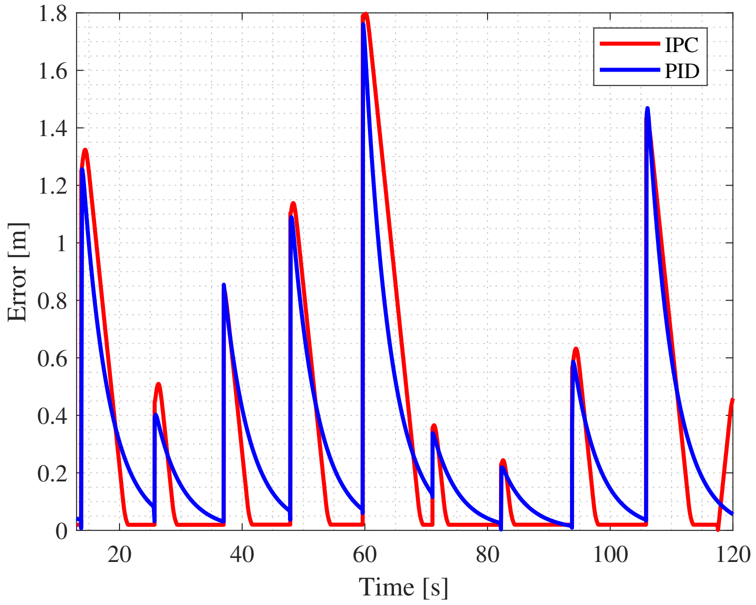


Figure 3.12. Trajectory tracking errors.

Chapter 4

Aerial Robots

4.1 Introduction

A common type of robot used in experimental platforms, both indoor and outdoor, are the aerial robot. These systems, unlike terrestrial mobile robots, are a challenge in the implementation of their control loops as they have faster and less stable dynamics. The most commonly accepted way to classify these robots is by their number of propellers. The most frequently used are those with four propellers, *quadcopters*, for small and medium-sized, and the ones with six and eight propellers, *hexacopters* and *octocopters*, respectively, used for big size systems. This chapter analyzes the case of quadcopters because they are the ones that will be used in the experimental work developed in this thesis.

A wide variety of controllers evaluated on these systems can be found in the literature [167, 168]. From the point of view of its dynamic modeling, a quadcopter is essentially a nonlinear multivariable system, which increases the interest in evaluating control strategies and combinations to obtain optimal systems. As usual, the controller that has been more extensively used and evaluated is the the PID controller [169–171]. Despite the generally good performance of this type of controller, the nonlinearity of the system opens the door to performance improvements with the use of more advanced control strategies. A good example of this is the use of genetic algorithms or fuzzy logic for the adaptive tuning of PID controllers for system dynamics [172, 173]. Other controllers that can be found in the literature include robust control [174], sliding control mode [175], event-based control [176], and neural networks controllers [177].

The control architecture follows a hierarchical structure. The lower level provides the position and orientation control of the quadcopter, whereas the upper level is responsible for controlling the trajectory followed by the quadcopter. The parameters that define the tuning of the controller are the closed-loop dynamics of the bottom layer, and there, a great variety of controllers can be found. For instance, there exist in the literature works with the most basic ones such as the PID controller, with acceptable performance [178, 179], but also one can find more complex controllers which use different techniques, such as sliding mode control [180], genetic algorithms [181], or neural networks [182]. For the specific case of planned navigation, one of the most widely used controllers is the Model Predictive Control (MPC), where the adjustment of the observation window is a key factor for the trajectory optimization and the obstacle avoidance problem [183–185].

Throughout this chapter, a detailed analysis of the case of the quadcopter will be carried out. In section 4.2 the model of this type of robot is studied. Since it is a nonlinear model, it is linearized to obtain the transfer functions that govern the robot's behavior around the equilibrium. Section 4.3 develops some of the most frequently used control strategies for the different levels of quadcopters. The analysis is focused on the Crazyflie 2.X drone. In section 4.4 the validation of the calculated models with experimental data will be performed. Finally, it will be shown experimental results carried out on the Crazyfly 2.1. The chapter ends with some brief conclusions that summarize the current status of the work.

4.2 Theoretical model

The physical modeling of quadcopters is widely studied in the literature [186–188]. Most of this models come from the academic world [189–191]. Quadcopters are underactuated systems since they have six degrees of freedom and only four actuators. For fully actuated systems it is necessary to work with hexarotors or specific modifications of the quadcopters as presented in [192–194]. In this section the modeling will be done in a parameterized way for an "X" configuration. This configuration is the most common for robots that incorporate sensors, such as cameras.

Dynamic model

To obtain the dynamic model of the system, first of all, three initial hypotheses are assumed about the robot that simplifies the model while maintaining a good approximation of reality.

- It behaves as a rigid body without deformations, i.e. the relative distance between all its points is constant. This implies that the arms supporting the rotors do not suffer bending deformations due to the forces to which it is subjected.
- It is symmetrical, both in mass and in rotor properties.
- The mass is constant over time.

To locate the drone reference frame, the extrinsic Tait-Bryan [195] representation is used. This implementation is not an efficient solution when real-time implementations are desired. In such cases, a model based on the Newton-Euler equations with quaternions is a better choice. In this situation, the transformation from the inertial frame, XYZ (see Figure 4.1) to the drone frame is determined by three successive rotations: yaw (ψ) → pitch (θ) → roll (ϕ). The rotations on the X and Z axes are taken clockwise while the Y rotation is taken counterclockwise [189]. The resulting rotation matrix is given in (4.2). Moreover, since it is an orthonormal matrix, equation (4.3) is satisfied.

$$\mathbf{R}_x(-\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix} \quad \mathbf{R}_y(\theta) = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix}$$
$$\mathbf{R}_z(-\psi) = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.1}$$

$$\mathbf{R}_o^b = \begin{pmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{pmatrix} \tag{4.2}$$

$$\left(\mathbf{R}_o^b\right)^{-1} = \left(\mathbf{R}_o^b\right)^T = \mathbf{R}_b^o \tag{4.3}$$

The modeling uses an internal representation based on state space. The state vector is composed of four components associated with the 6 degrees of freedom of the drone and their respective velocities, $\mathbf{x}\,[\mathbf{p}, \boldsymbol{\Phi}, \mathbf{v}, \boldsymbol{\omega}] \in \mathbb{R}^{12}$ where $\mathbf{p} \in \mathbb{R}^3\,[m]$ denotes the center of mass in the inertial frame, $\Phi \in \mathbb{R}^3\,[rad]$ denotes the extrinsic ZYX Tait-Bryan angles of the rigid-body rotation in the inertial frame, $\mathbf{v} \in \mathbb{R}^3\,[m/s]$ denotes the linear velocities of the center of mass in the inertial frame, and $\boldsymbol{\omega} \in \mathbb{R}^3\,[rad/s]$ denotes the angular velocity in the local frame to the inertial frame. Furthermore, $\boldsymbol{\Omega} \in \mathbb{R}^4\,[rad/s]$ represents the rotational velocities of the four rotors of the system.

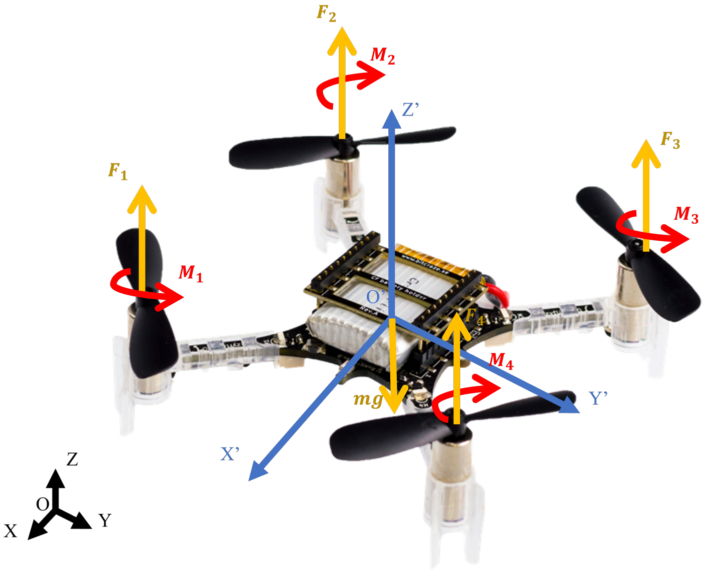


Figure 4.1. Forces and moments diagram in the Crazyflie 2.X.

Force Equations

To determine the equations related to the linear parameters of the drone, Newton's second law is applied, determining the derivative of the velocity according to the Coriolis equation [196]

$$\Sigma F = m \cdot \dot{\mathbf{v}} = m \cdot \left(\dot{\mathbf{v}}^b + \boldsymbol{\omega} \times \mathbf{v}^b\right) \tag{4.4}$$

where F is the global force of the drone, m is the mass, and $\dot{\mathbf{v}}^b$ is the velocity of the center of mass in the local frame. If it is considered that in the stationary state (the drone is oriented parallel to the ground) both roll and pitch are zero, the force balance allows to isolate the derivative of the linear velocity of the drone:

$$\begin{bmatrix} 0 \\ 0 \\ F_z \end{bmatrix} - R_o^b \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} = m \cdot \left(\dot{\mathbf{v}}^b + \boldsymbol{\omega} \times \mathbf{v}^b\right) \rightarrow \dot{\mathbf{v}}^b = \begin{bmatrix} 0 \\ 0 \\ \frac{F_z}{m} \end{bmatrix} - R_o^b \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} - \boldsymbol{\omega} \times \mathbf{v}^b \tag{4.5}$$

To determine the derivatives of the state spaces referring to the position in the inertial frame, the linear velocities of the drone in the local frame are projected using the corresponding transformation matrix (4.2).

$$\dot{\mathbf{p}} = R_b^o \cdot \dot{\mathbf{v}}^b \tag{4.6}$$

The following equation is used to determine the force generated by the drone rotors:

$$F_i^b = \begin{bmatrix} 0 \\ 0 \\ T_i \end{bmatrix} \rightarrow T_i = C_T \Omega_i^2 \rightarrow \Sigma F_i^b = \begin{bmatrix} 0 \\ 0 \\ C_T \left(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2\right) \end{bmatrix} \tag{4.7}$$

where $F_i^b\,[N]$ is the local force generated by rotor i, $T_i\,[N]$ is the vertical thrust, $\Omega_i\,[rad/s]$ is the rotational speed of rotor i, and $C_T\,[N/rpm^2]$ is the thrust coefficient determined by (4.8).

$$C_T = k_T \rho \frac{(2r)^4}{3600} \tag{4.8}$$

where $\rho\,[Kg/m^3]$ is the density of air, $r\,[m]$ and k_T are the rotors' radius and thrust coefficient.

Momentum Equations

To determine the elements associated with the angular components of the system states, momentum balance is used. Applying the summation of moments equivalent to the angular momentum of the drone using the equivalence of the Coriolis equation, it is obtained (4.9).

$$\Sigma M^o = {}^o\dot{h} \rightarrow \Sigma M^o = {}^b\dot{h} + \boldsymbol{\omega} \times h \tag{4.9}$$

where h represents the angular momentum of the center of mass in the global frame. Applied to the local frame, the equations of the quantity of motion are more easily calculated, as explained in [197, 198],

$$\Sigma M^b = J\ {}^b\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times J\boldsymbol{\omega} \tag{4.10}$$

where $\mathbf{J}$ represents the inertia matrix of the drone. Assuming the assumption of drone symmetry, the estimation of the inertia matrix is simplified according to (4.11) and (4.12).

$$\mathbf{J} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \rightarrow \mathbf{J} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{4.11}$$

$$(\mathbf{J})^{-1} = \frac{adj\begin{pmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{pmatrix}}{\begin{vmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{vmatrix}} = \frac{\begin{pmatrix} I_{yy}I_{zz} & 0 & 0 \\ 0 & I_{xx}I_{zz} & 0 \\ 0 & 0 & I_{xx}I_{yy} \end{pmatrix}}{I_{xx}I_{yy}I_{zz}} = \begin{pmatrix} \frac{1}{I_{xx}} & 0 & 0 \\ 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & \frac{1}{I_{zz}} \end{pmatrix} \tag{4.12}$$

Therefore, it can be estimated the term ${}^b\dot{\boldsymbol{\omega}}$ as follows:

$${}^b\dot{\boldsymbol{\omega}} = (J)^{-1}\left(\begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} - \boldsymbol{\omega} \times J\boldsymbol{\omega}\right) \tag{4.13}$$

The last equation of state determines the relationship between the $\boldsymbol{\omega}$ vector and the derivative of $\boldsymbol{\Phi}$.

$$\boldsymbol{\omega} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \dot{\boldsymbol{\Phi}} \rightarrow \dot{\boldsymbol{\Phi}} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \boldsymbol{\omega} \tag{4.14}$$

The procedure to determine the correlation between the velocity of rotation of the propellers and the resulting momentum is the following:

$$M = \Sigma P_i \times F_i + \Sigma\tau_i \tag{4.15}$$

where P_i represents the distance from each motor to the center of gravity, F_i is the force generated by each rotor, and τ_i represents the momentum induced in the drone by each rotor. This moment is a consequence of Newton's third law applied to the propeller rotation. By using an "X" configuration, the position of each rotor is defined by

$$P_1 = \begin{bmatrix} \frac{d}{\sqrt{2}} \\ -\frac{d}{\sqrt{2}} \\ 0 \end{bmatrix}; P_2 = \begin{bmatrix} -\frac{d}{\sqrt{2}} \\ -\frac{d}{\sqrt{2}} \\ 0 \end{bmatrix}; P_3 = \begin{bmatrix} -\frac{d}{\sqrt{2}} \\ \frac{d}{\sqrt{2}} \\ 0 \end{bmatrix}; P_4 = \begin{bmatrix} \frac{d}{\sqrt{2}} \\ \frac{d}{\sqrt{2}} \\ 0 \end{bmatrix} \tag{4.16}$$

where d represents the distance from each rotor to the center of gravity of the drone.

Knowing the force generated by each motor from (4.7), the moment product of that force can be determined as follows:

$$\begin{aligned} P_1 \times F_1 &= \begin{bmatrix} -\left(C_T\Omega_1^2\right) d/\sqrt{2} \\ -\left(C_T\Omega_1^2\right) d/\sqrt{2} \\ 0 \end{bmatrix} & P_2 \times F_2 &= \begin{bmatrix} -\left(C_T\Omega_2^2\right) d/\sqrt{2} \\ \left(C_T\Omega_2^2\right) d/\sqrt{2} \\ 0 \end{bmatrix} \\ P_3 \times F_3 &= \begin{bmatrix} \left(C_T\Omega_3^2\right) d/\sqrt{2} \\ \left(C_T\Omega_3^2\right) d/\sqrt{2} \\ 0 \end{bmatrix} & P_4 \times F_4 &= \begin{bmatrix} \left(C_T\Omega_4^2\right) d/\sqrt{2} \\ -\left(C_T\Omega_4^2\right) d/\sqrt{2} \\ 0 \end{bmatrix} \end{aligned} \tag{4.17}$$

Applying conservation of angular momentum:

$$\Sigma\tau_i^b = \begin{bmatrix} 0 \\ 0 \\ C_D\left(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2\right) \end{bmatrix} \tag{4.18}$$

where C_D represents the torque coefficient [199], which depends on the dimensionless coefficient k_D, the air density ρ_o, and the rotor radius r according to the following expression

$$C_D = k_D\rho\left(2r\right)^5/3600 \tag{4.19}$$

Therefore, in the calculation of the moments, the result would be:

$$M^b = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} dC_T/\sqrt{2}\left(-\Omega_1^2 - \Omega_2^2 + \Omega_3^2 + \Omega_4^2\right) \\ dC_T/\sqrt{2}\left(-\Omega_1^2 + \Omega_2^2 + \Omega_3^2 - \Omega_4^2\right) \\ C_D\left(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2\right) \end{bmatrix} \tag{4.20}$$

In the total quantity of motion equation, there are certain terms including angular accelerations that have been omitted, as they tend to be small compared to the other terms in the equation. The gyroscope moments have also been omitted because the moment of inertia of each motor tends to be small, so their contribution to the total momentum is also negligible [200, 201].

Linear state space model

The development of a nonlinear model of the system is important for the construction of highly accurate simulators or predictors. However, sometimes a simplified model around the operating point where the system works is more valuable. To obtain it, the use of a Taylor series linearization process of the system is a good option. In this way, it is possible to obtain the characteristic matrices A, B, C, and D of the model in the state space as follows:

$$\begin{aligned}\Delta\dot{x} &= A\Delta X + B\Delta U\\ \Delta y &= C\Delta X + D\Delta U\end{aligned}\tag{4.21}$$

At the equilibrium point around which the system is linearized, condition $\dot{X}_e = 0$ is satisfied. In this point, the velocity components and the pitch and roll angles are zero. It means that the drone remains suspended in the air. Therefore the state vector can be expressed as follows:

$$X_e = [x_e \;\; y_e \;\; z_e \;\; 0 \;\; 0 \;\; \psi_e \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0]^T \tag{4.22}$$

Similarly, in the ideal case, at the equilibrium point, the rotational speed of the rotors must be the same, Ω_e. This value is determined by applying the balance of forces on the Z-axis:

$$C_T(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) = mg \rightarrow \Omega_e = \sqrt{\frac{mg}{4C_T}} \tag{4.23}$$

Applying the first term of the Taylor series to (4.5), (4.7), (4.13), and (4.20) it is obtained the new equations governing the linear behavior of the system:

$$\begin{cases} \Delta F_x = m\Delta\dot{u} - mg\Delta\theta & \Delta F_y = m\Delta\dot{v} - mg\Delta\phi & \Delta F_z = m\Delta\dot{w} \\ \Delta M_x = I_{xx}\Delta\dot{p} & \Delta M_y = I_{yy}\Delta\dot{q} & \Delta M_z = I_{zz}\Delta\dot{r} \end{cases}\tag{4.24}$$

$$\begin{bmatrix}\Delta F^b\\ \Delta M^b\end{bmatrix} = \begin{bmatrix}\Delta F_x\\ \Delta F_y\\ \Delta F_z\\ \Delta M_x\\ \Delta M_y\\ \Delta M_z\end{bmatrix} = \begin{bmatrix} 0\\ 0\\ 2C_T\Omega_e(\Delta\Omega_1 + \Delta\Omega_2 + \Delta\Omega_3 + \Delta\Omega_4)\\ \sqrt{2}dC_T\Omega_e(-\Delta\Omega_1 - \Delta\Omega_2 + \Delta\Omega_3 + \Delta\Omega_4)\\ \sqrt{2}dC_T\Omega_e(-\Delta\Omega_1 + \Delta\Omega_2 + \Delta\Omega_3 - \Delta\Omega_4)\\ 2C_D\Omega_e(-\Delta\Omega_1 + \Delta\Omega_2 - \Delta\Omega_3 + \Delta\Omega_4)\end{bmatrix}\tag{4.25}$$

In the stationary flight position, the fixed frame of the body coincides with the inertial frame, which implies the equalities shown in the following equation:

$$\begin{cases} \Delta\dot{x} = \Delta u & \Delta\dot{\phi} = \Delta p\\ \Delta\dot{y} = \Delta y & \Delta\dot{\theta} = \Delta q\\ \Delta\dot{z} = \Delta z & \Delta\dot{\psi} = \Delta r \end{cases}\tag{4.26}$$

Therefore, the linear representation of the system using the state space representation has the following state vectors, control, and characteristic matrices:

$$\Delta X = [\Delta x \;\; \Delta y \;\; \Delta z \;\; \Delta\psi \;\; \Delta\theta \;\; \Delta\phi \;\; \Delta u \;\; \Delta v \;\; \Delta w \;\; \Delta r \;\; \Delta q \;\; \Delta p]^T \tag{4.27}$$

$$\Delta U = [\Delta\Omega_1 \;\; \Delta\Omega_2 \;\; \Delta\Omega_3 \;\; \Delta\Omega_4]^T \tag{4.28}$$

$$\Delta Y = [\Delta x \;\; \Delta y \;\; \Delta z \;\; \Delta\psi]^T \tag{4.29}$$

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & g & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{2C_T}{m} & \frac{2C_T}{m} & \frac{2C_T}{m} & \frac{2C_T}{m} \\ -\frac{2C_D}{I_zz} & \frac{2C_D}{I_zz} & -\frac{2C_D}{I_zz} & \frac{2C_D}{I_zz} \\ -\frac{\sqrt{2}dC_T}{I_{yy}} & \frac{\sqrt{2}dC_T}{I_{yy}} & \frac{\sqrt{2}dC_T}{I_{yy}} & -\frac{\sqrt{2}dC_T}{I_{yy}} \\ -\frac{\sqrt{2}dC_T}{I_{xx}} & -\frac{\sqrt{2}dC_T}{I_{xx}} & \frac{\sqrt{2}dC_T}{I_{xx}} & \frac{\sqrt{2}dC_T}{I_{xx}} \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.30}$$

Once the internal representation has been defined (both nonlinear and linear), it can be calculated the external representation of the system using transfer functions as follows:

$$G(s) = \frac{Y(s)}{U(s)} = \Omega_e \begin{bmatrix} -\frac{\sqrt{2}dC_T g}{I_{yy}s^4} & \frac{\sqrt{2}dC_T g}{I_{yy}s^4} & \frac{\sqrt{2}dC_T g}{I_{yy}s^4} & -\frac{\sqrt{2}dC_T g}{I_{yy}s^4} \\ \frac{\sqrt{2}dC_T g}{I_{xx}s^4} & \frac{\sqrt{2}dC_T g}{I_{xx}s^4} & -\frac{\sqrt{2}dC_T g}{I_{xx}s^4} & -\frac{\sqrt{2}dC_T g}{I_{xx}s^4} \\ \frac{2C_T}{ms^2} & \frac{2C_T}{ms^2} & \frac{2C_T}{ms^2} & \frac{2C_T}{ms^2} \\ -\frac{2C_D}{I_{zz}s^2} & \frac{2C_D}{I_{zz}s^2} & -\frac{2C_D}{I_{zz}s^2} & \frac{2C_D}{I_{zz}s^2} \\ -\frac{\sqrt{2}dC_T}{I_{yy}s^2} & \frac{\sqrt{2}dC_T}{I_{yy}s^2} & \frac{\sqrt{2}dC_T}{I_{yy}s^2} & -\frac{\sqrt{2}dC_T}{I_{yy}s^2} \\ -\frac{\sqrt{2}dC_T}{I_{xx}s^2} & -\frac{\sqrt{2}dC_T}{I_{xx}s^2} & \frac{\sqrt{2}dC_T}{I_{xx}s^2} & \frac{\sqrt{2}dC_T}{I_{xx}s^2} \\ -\frac{\sqrt{2}dC_T g}{I_{yy}s^3} & \frac{\sqrt{2}dC_T g}{I_{yy}s^3} & \frac{\sqrt{2}dC_T g}{I_{yy}s^3} & -\frac{\sqrt{2}dC_T g}{I_{yy}s^3} \\ \frac{\sqrt{2}dC_T g}{I_{xx}s^3} & \frac{\sqrt{2}dC_T g}{I_{xx}s^3} & -\frac{\sqrt{2}dC_T g}{I_{xx}s^3} & -\frac{\sqrt{2}dC_T g}{I_{xx}s^3} \\ \frac{2C_T}{ms} & \frac{2C_T}{ms} & \frac{2C_T}{ms} & \frac{2C_T}{ms} \\ -\frac{2C_D}{I_{zz}s} & \frac{2C_D}{I_{zz}s} & -\frac{2C_D}{I_{zz}s} & \frac{2C_D}{I_{zz}s} \\ -\frac{\sqrt{2}dC_T}{I_{yy}s} & \frac{\sqrt{2}dC_T}{I_{yy}s} & \frac{\sqrt{2}dC_T}{I_{yy}s} & -\frac{\sqrt{2}dC_T}{I_{yy}s} \\ -\frac{\sqrt{2}dC_T}{I_{xx}s} & -\frac{\sqrt{2}dC_T}{I_{xx}s} & \frac{\sqrt{2}dC_T}{I_{xx}s} & \frac{\sqrt{2}dC_T}{I_{xx}s} \end{bmatrix} \tag{4.31}$$

The results obtained are equivalent to other works that can be found in the literature [202], thus verifying the process followed both in modeling and linearization. It is important to note that in this representation the pure integrative nature of the system and the cascading relationship between the position and velocity parameters is more clearly seen. A direct implication of this phenomenon in the control and stability of these systems is that the open-loop instability is verified.

4.3 Multivariable control architecture

As described in the previous section, the quadcopter model is a MIMO system. The control architecture of this type of agent usually has two levels made up of a cascade system each one, as shown in Figure 4.2. The upper level is in charge of controlling the position and speed of the robot. This level usually runs at 100 Hz. Its input signal is the target position and its outputs are the thrust and the pitch and roll angles. The next level is the stability control and it is defined by the Attitude and Rate controllers. It requires a higher operating frequency, 500 Hz, and its output signals are the commands that each robot's rotors must receive. In the following it is shown some control architectures applicable to this type of multivariable system to decouple their control levels and conventional PID-based implementations.

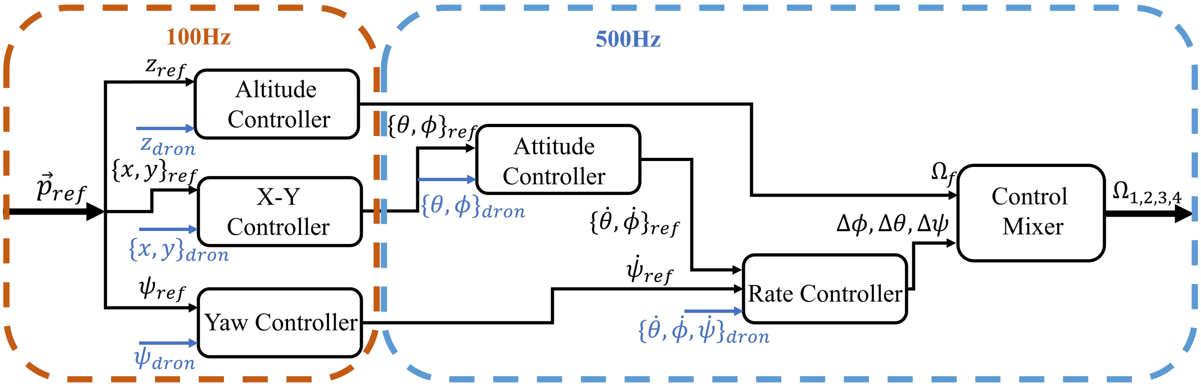


Figure 4.2. Block diagram of the quadcopters' control architecture.

4.3.1 Preliminary considerations

With the current selection of input and output variables, it is complex to determine the variable pairings to determine the control loops. In the majority of the developed projects with quadcopters, a previous stage is added to the process called "power distribution". This distribution stage establishes a correlation between the rotational speeds of the motors and four new input signals. It simplifies the matching of the multivariable system. The manipulated variables of the quadcopter become the thrust (Ω_f) and the($\Delta\theta$), roll ($\Delta\phi$), and yaw ($\Delta\psi$) angle increments. The correlation between these new variables and the rotational speeds of the different rotors (Ω_i) is not the same for all implementations. In the proposed structure, the angular variation is distributed among the four rotors. If the increases in the quadcopter angles were not distributed in this ratio of ±0.5 among the four motors, any angles' variation would lead to increases or decreases in the thrust signal and it would have to be compensated by the control loop of the thrust signal because only two rotors were actuated with unity gain. In the same way, the resulting variable switching matrix is designed for an "X" configuration. For a "+" configuration, the association of motors with the different angles varies. The following equation shows the $D(s)$ matrix that establishes the relationship between the newly manipulated variables and the rotational speeds of each rotor:

$$\begin{bmatrix}\Omega_1\\ \Omega_2\\ \Omega_3\\ \Omega_4\end{bmatrix} = D(s)\begin{bmatrix}\Omega_f\\ \Delta\theta\\ \Delta\phi\\ \Delta\psi\end{bmatrix} = \begin{bmatrix}1 & -0.5 & -0.5 & -1\\ 1 & 0.5 & -0.5 & 1\\ 1 & 0.5 & 0.5 & -1\\ 1 & -0.5 & 0.5 & 1\end{bmatrix}\begin{bmatrix}\Omega_f\\ \Delta\theta\\ \Delta\phi\\ \Delta\psi\end{bmatrix} \tag{4.32}$$

With this change, the transfer function matrix of the MIMO system that determines the behavior of a generic quadcopter as follows

$$G'(s) = G(s)\cdot D(s) = \Omega_e \begin{bmatrix} 0 & \frac{2\sqrt{2}dC_Tg}{I_{yy}s^4} & 0 & 0\\ 0 & 0 & -\frac{2\sqrt{2}dC_Tg}{I_{xx}s^4} & 0\\ \frac{8C_T}{ms^2} & 0 & 0 & 0\\ 0 & 0 & 0 & \frac{8C_D}{I_{zz}s^2}\\ 0 & \frac{2\sqrt{2}dC_T}{I_{yy}s^2} & 0 & 0\\ 0 & 0 & \frac{2\sqrt{2}dC_T}{I_{xx}s^2} & 0\\ 0 & \frac{2\sqrt{2}dC_Tg}{I_{yy}s^3} & 0 & 0\\ 0 & 0 & -\frac{2\sqrt{2}dC_Tg}{I_{xx}s^3} & 0\\ \frac{8C_T}{ms} & 0 & 0 & 0\\ 0 & 0 & 0 & \frac{8C_D}{I_{zz}s}\\ 0 & \frac{2\sqrt{2}dC_T}{I_{yy}s} & 0 & 0\\ 0 & 0 & \frac{2\sqrt{2}dC_T}{I_{xx}s} & 0 \end{bmatrix} \tag{4.33}$$

It is observed that the new process structure is a fully decoupled system. So, from (4.33), it can be concluded that the power distribution stage involves a perfect decoupling in the multivariable system comprising the operation of a quadcopter. This decoupling network involves a form of compensation known as **direct decoupling** since the new control signals go to the direct process inputs through the $D(s)$ terms. In this case, the transfer function matrix that enables decoupling is composed of static gains but achieves decoupling in both steady and transient regimes.

4.3.2 Centralized control by decoupling

As shown in Figure 4.2, the complete system is composed of 6 loops, each of which is a cascade system containing the position and velocity of each parameter. This would imply that the design of a complete control system requires the design of 12 controllers simultaneously. Given the order of the processes, it was decided to narrow down the control problem for the study of this type of controller. For this purpose, the proposed implementation for the controller of this first level includes the elevation speed control and the angles *roll, pitch*, and *yaw* speed control, also called "Rate Controller". If the state vector is defined as $\begin{bmatrix}\dot{z} & \dot{\theta} & \dot{\phi} & \dot{\psi}\end{bmatrix}^T$, the transfer function matrices of these processes are defined as follows:

$$G(s) = \Omega_e \begin{pmatrix} \frac{2C_T}{ms} & \frac{2C_T}{ms} & \frac{2C_T}{ms} & \frac{2C_T}{ms} \\ -\frac{\sqrt{2}dC_T}{I_{yy}s} & \frac{\sqrt{2}dC_T}{I_{yy}s} & \frac{\sqrt{2}dC_T}{I_{yy}s} & -\frac{\sqrt{2}dC_T}{I_{yy}s} \\ -\frac{\sqrt{2}dC_T}{I_{xx}s} & -\frac{\sqrt{2}dC_T}{I_{xx}s} & \frac{\sqrt{2}dC_T}{I_{xx}s} & \frac{\sqrt{2}dC_T}{I_{xx}s} \\ -\frac{2C_D}{I_{zz}s} & \frac{2C_D}{I_{zz}s} & -\frac{2C_D}{I_{zz}s} & \frac{2C_D}{I_{zz}s} \end{pmatrix} \tag{4.34}$$

The techniques used to design a centralized decoupling controller are similar to those used to design decoupling networks [203]. Therefore, the designed network (4.32) will be used.

Design by explicit decoupling

The equivalent multivariable controller for direct decoupling is expressed as follows:

$$K(s) = D(s)K_d(s) = D(s)\begin{pmatrix} k_1 & 0 & 0 & 0 \\ 0 & k_2 & 0 & 0 \\ 0 & 0 & k_3 & 0 \\ 0 & 0 & 0 & k_4 \end{pmatrix} = \begin{pmatrix} k_1 & -0.5k_2 & -0.5k_3 & -k_4 \\ k_1 & 0.5k_2 & -0.5k_3 & k_4 \\ k_1 & 0.5k_2 & 0.5k_3 & -k_4 \\ k_1 & -0.5k_2 & 0.5k_3 & k_4 \end{pmatrix} \tag{4.35}$$

where k_1, k_2, k_3, and k_4 are controllers that are designed using single-variable techniques according to the apparent processes resulting from direct decoupling.

First, since the four processes to be controlled are first-order systems with a pole at the origin, the design of the controller is carried out generically. In this case, to maintain the first-order closed loop and establish a unity gain, a proportional controller will be used:

$$G_{CL}(s) = \frac{C(s)G(s)}{1 + C(s)G(s)} = \frac{K_p \cdot \frac{k}{s}}{1 + K_p \cdot \frac{k}{s}} = \frac{1}{\frac{1}{K_p k}s + 1} \rightarrow \tau_{CL} = \frac{1}{K_p k} \rightarrow K_p = \frac{1}{\tau_{CL} k} \tag{4.36}$$

With this design condition, valid for the four controllers to be implemented, the next action is to establish the desired time constant for the closed-loop systems. In the case of the stability control, $\dot{\theta}$ and $\dot{\phi}$, it is considered appropriate to set its time constant to $0.05[s]$. This criterion is adopted to avoid excessively high values in the proportional term that may cause undesired behaviors in the system dynamics. In the case of $\dot{\psi}$ control, the most frequently used solution is to adopt a common value. According to this criterion, the time constant τ_{CL} for this loop is set at $0.015[s]$. Finally, for the speed control, following the same criterion, the time constant is set to $0.04[s]$. As a result, the control parameters for the UAV Crazyflie are shown in Table 4.1.

Loop	$\tau_{CL}[s]$	K_p
$\dot{z}$	0.04	27.38
$\dot{\theta}$	0.05	55.55
$\dot{\phi}$	0.05	529.10
$\dot{\psi}$	0.015	116.22

Table 4.1. Parameters for a centralized controller for explicit decoupling.

Therefore, the obtained centralized controller by explicit decoupling is defined as follows:

$$K(s) = D(s)K_d(s) = \begin{pmatrix} 27.38 & -277.7778 & -264.5503 & -116.22 \\ 27.38 & 277.7778 & -264.5503 & 116.22 \\ 27.38 & 277.7778 & 264.5503 & -116.22 \\ 27.38 & -277.7778 & 264.5503 & 116.22 \end{pmatrix} \tag{4.37}$$

The results of this implementation will be reflected after the development of the implicit method to perform a joint comparison of all the decoupling control data.

Design by implicit decoupling

In the case of centralized control by implicit decoupling, the general expression for the calculation of the corresponding controller is defined in (37) of [203]:

$$K(s) = \frac{adj(G(s))}{det(G(s))}L(s) \tag{4.38}$$

Considering again the practical case of the Crazyflie, the resulting matrix from the coefficient between the adjoint and the determinant of $G(s)$ is given by

$$\frac{adj(G(s))}{det(G(s))} = \begin{pmatrix} 1.0955s & -13.8889s & -13.2275s & -1.7434s \\ 1.0955s & 13.8889s & -13.2275s & 1.7434s \\ 1.0955s & 13.8889s & 13.2275s & -1.7434s \\ 1.0955s & -13.8889s & 13.2275s & 1.7434s \end{pmatrix} \tag{4.39}$$

The next step is to determine the four open-loop transfer functions to be imposed on the system. Their definition must take into account that the controllers are achievable and the closed-loop specifications are attainable. Normally, for the response to be stable in the closed loop, following references and rejecting disturbances, an integrator is added to the open-loop transfer functions. In this case, this is necessary because the process includes a pure integrator that fulfills this condition. Since the system is a minimum phase and has no delay, no relative stability specifications can be used for the controller. The same design criteria will be used as in the case of the explicit controller. The closed-loop transfer function $G_{CL}(s)$ is defined as follows:

$$G_{CL}(s) = \frac{C(s)G(s)}{1+C(s)G(s)} = \frac{\frac{k_i}{s}}{1+\frac{k_i}{s}} = \frac{1}{\frac{1}{k_i}s+1} \rightarrow \tau_{CL} = \frac{1}{k_i} \rightarrow l_i(s) = \frac{k_i}{s} \tag{4.40}$$

$$K(s) = \frac{adj(G(s))}{det(G(s))}\begin{pmatrix} \frac{k_1}{s} & 0 & 0 & 0 \\ 0 & \frac{k_2}{s} & 0 & 0 \\ 0 & 0 & \frac{k_3}{s} & 0 \\ 0 & 0 & 0 & \frac{k_4}{s} \end{pmatrix} = \begin{pmatrix} \frac{1.0955}{\tau_{CL,1}} & -\frac{13.8889}{\tau_{CL,2}} & -\frac{13.2275}{\tau_{CL,3}} & -\frac{1.7434}{\tau_{CL,4}} \\ \frac{1.0955}{\tau_{CL,1}} & \frac{13.8889}{\tau_{CL,2}} & -\frac{13.2275}{\tau_{CL,3}} & \frac{1.7434}{\tau_{CL,4}} \\ \frac{1.0955}{\tau_{CL,1}} & \frac{13.8889}{\tau_{CL,2}} & \frac{13.2275}{\tau_{CL,3}} & -\frac{1.7434}{\tau_{CL,4}} \\ \frac{1.0955}{\tau_{CL,1}} & -\frac{13.8889}{\tau_{CL,2}} & \frac{13.2275}{\tau_{CL,3}} & \frac{1.7434}{\tau_{CL,4}} \end{pmatrix} \tag{4.41}$$

To make a comparison between controllers, the same temporal specifications are established for the closed loops as in the explicit case. Therefore, the resulting control matrix is given by:

$$K(s) = \frac{adj(G(s))}{det(G(s))}L(s) = \begin{pmatrix} 27.38 & -277.78 & -264.55 & -116.22 \\ 27.38 & 277.78 & -264.55 & 116.22 \\ 27.38 & 277.78 & 264.55 & -116.22 \\ 27.38 & -277.78 & 264.55 & 116.22 \end{pmatrix} \tag{4.42}$$

It is important to note that the numerical results obtained by both procedures are identical despite having followed two different calculation procedures. This is logical since the same time conditions have been imposed to verify the validity of the obtained results. Table 4.2 compares the performance of the designed controllers with a conventional Proportional control with direct decoupling [186]. The parameters studied to analyze the performance of the controllers are IAE, ITAE, and Root Mean Square Error (RMSE). The results of Table 4.2 show how the designed decouplings, both explicitly and implicitly, improve the quality of the response, being more significant in the implicit case. In contrast, the $\dot{\psi}$ case does not improve the PID implementation. However, this worsening is compensated by the improvement of the proposed implementations with respect to *pitch* and *roll* velocity. In the case of Z-motion, the variation is much less noticeable but still, there exists a slight improvement over the reference case.

4.3.3 PID Controller

The most frequent implementation due to its simplicity and robustness is the use of PID controllers in each of the controllers in Figure 4.2 with direct decoupling (4.32). The mathematical

Loop	z			$\dot{\theta}$			$\dot{\phi}$			$\dot{\psi}$		
Case	PID	Expli.	Impli.	PID	Expli.	Impli.	PID	Expli.	Impli.	PID	Expli.	Impli.
IAE	1.01	1.00	0.99	20.07	7.04	5.41	22.08	8.54	7.03	1.16	3.10	9.14
ITAE	6.79	5.25	5.07	350.1	110.9	81.19	307.5	125.5	105.9	18.45	50.58	154.7
RMSE	0.11	0.11	0.11	2.69	1.25	0.95	2.78	1.28	1.01	0.09	0.15	0.48

Table 4.2. Analytical results for decoupling control.

development that establishes the closed-loop behavior for the position control of the quadcopter Crazyflie is presented next. The same selection of controller type as implemented by the developer in the official firmware will be maintained. However, it will be parameterized to visualize in the final result the effect introduced by each controller.

Rate Controller

At this level, the velocity control of the angles associated with the three degrees of freedom linked to the orientation is implemented. The reference signals are angular velocity setpoints $[rad/s]$ and the control signals are $\Delta\theta$, $\Delta\phi$, and $\Delta\psi$. The three loops involved have the same structure, the processes are pure integrators and the controllers used are PID controllers in their parallel algorithm. The subindexs $\dot{\theta}$, $\dot{\phi}$, y $\dot{\psi}$ are used to identify the loops associated with each angle. For the theoretical development of the closed loop controller, (4.45) and (4.46), the generic form (4.43) for the process and (4.44) for the controller will be considered.

$$G_{\dot{\theta},\dot{\phi},\dot{\psi}}\left(s\right) = \frac{k_{\dot{\theta},\dot{\phi},\dot{\psi}}}{s} \tag{4.43}$$

$$C_{\dot{\theta},\dot{\phi},\dot{\psi}}\left(s\right) = K_{P,\dot{\theta},\dot{\phi},\dot{\psi}} + K_{I,\dot{\theta},\dot{\phi},\dot{\psi}}\frac{1}{s} + K_{D,\dot{\theta},\dot{\phi},\dot{\psi}}s \tag{4.44}$$

$$G_{CL,\dot{\theta},\dot{\phi},\dot{\psi}}\left(s\right) = \frac{\frac{k}{s}\left(K_P + K_I\frac{1}{s} + K_D s\right)}{1 + \frac{k}{s}\left(K_P + K_I\frac{1}{s} + K_D s\right)} = \frac{kK_D s^2 + kK_P s + kK_I}{\left(1 + kK_D\right)s^2 + kK_P s + kK_I} \tag{4.45}$$

$$G_{CL,\dot{\theta},\dot{\phi},\dot{\psi}}\left(s\right) = \frac{\frac{kK_D}{1+kK_D}s^2 + \frac{kK_P}{1+kK_D}s + \frac{kK_I}{1+kK_D}}{s^2 + \frac{kK_P}{1+kK_D}s + \frac{kK_I}{1+kK_D}} = \frac{\frac{kK_D}{1+kK_D}\left(s^2 + \frac{K_P}{K_D}s + \frac{K_I}{K_D}\right)}{s^2 + \frac{kK_P}{1+kK_D}s + \frac{kK_I}{1+kK_D}} \tag{4.46}$$

As it can be seen in (4.47), a second-order response with two poles and two zeros is obtained. To work with lower-order models in the higher levels of control, it is possible to adopt order reduction solutions or filters in the reference that eliminate the zeros and one pole of the system.

$$G_{CL,\dot{\theta},\dot{\phi},\dot{\psi}}\left(s\right) = \frac{k_r\left(s^2 + b_{r,1}s + b_{r,2}\right)}{s^2 + a_{r,1}s + a_{r,2}} \begin{cases} k_r = & \frac{k_{\dot{\theta},\dot{\phi},\dot{\psi}}K_{D,\dot{\theta},\dot{\phi},\dot{\psi}}}{1+k_{\dot{\theta},\dot{\phi},\dot{\psi}}K_{D,\dot{\theta},\dot{\phi},\dot{\psi}}} \\ b_{r,1} = & \frac{K_{P,\dot{\theta},\dot{\phi},\dot{\psi}}}{K_{D,\dot{\theta},\dot{\phi},\dot{\psi}}} \\ b_{r,2} = & \frac{K_{I,\dot{\theta},\dot{\phi},\dot{\psi}}}{K_{D,\dot{\theta},\dot{\phi},\dot{\psi}}} \\ a_{a,1} = & \frac{k_{\dot{\theta},\dot{\phi},\dot{\psi}}K_{P,\dot{\theta},\dot{\phi},\dot{\psi}}}{1+k_{\dot{\theta},\dot{\phi},\dot{\psi}}K_{D,\dot{\theta},\dot{\phi},\dot{\psi}}} \\ a_{a,2} = & \frac{k_{\dot{\theta},\dot{\phi},\dot{\psi}}K_{I,\dot{\theta},\dot{\phi},\dot{\psi}}}{1+k_{\dot{\theta},\dot{\phi},\dot{\psi}}K_{D,\dot{\theta},\dot{\phi},\dot{\psi}}} \end{cases} \tag{4.47}$$

Attitude Controller and Yaw Controller

The next level up is responsible for position control of the angles associated with the three degrees of freedom linked to the orientation. The reference signals are setpoints of the orientation angles $[rad]$ and the control signals are $\dot{\theta}$, $\dot{\phi}$, and $\dot{\psi}$. The three loops involved have the same structure, a series combination of (4.47) and pure integrators. The controllers used are Proportional-Integral (PI) controllers in their parallel algorithm. The subindexs θ, ϕ, y ψ are used to identify the loops associated with each angle. For the theoretical development of the closed loop controller, (4.50) and (4.51), the generic form (4.48) for the process and (4.49) for the controller will be considered.

$$G_{\theta,\phi,\psi}\left(s\right) = \frac{k_r\left(s^2 + b_{r,1}s + b_{r,2}\right)}{s^3 + a_{r,1}s^2 + a_{r,2}s} \tag{4.48}$$

$$C_{\theta,\phi,\psi}\left(s\right) = K_{P,\theta,\phi,\psi} + K_{I,\theta,\phi,\psi}\frac{1}{s} = \frac{K_{P,\theta,\phi,\psi}s + K_{I,\theta,\phi,\psi}}{s} \tag{4.49}$$

$$G_{CL,\theta,\phi,\psi}\left(s\right) = \frac{k\left(K_P s + K_I\right)\left(b_0 s^2 + b_1 s + b_2\right)}{s^4 + a_1 s^3 + a_2 s^2 + k\left(K_P s + K_I\right)\left(b_0 s^2 + b_1 s + b_2\right)} \tag{4.50}$$

$$G_{CL,\theta,\phi,\psi}\left(s\right) = \frac{k_a\left(s^3 + b_{a,1}s^2 + b_{a,2}s + b_{a,3}\right)}{s^4 + a_{a,1}s^3 + a_{a,2}s^2 + a_{a,3}s + a_{a,4}} \begin{cases} k_a = & k_r K_{P,\theta,\phi,\psi} \\ b_{a,1} = & \frac{k_r K_{P,\theta,\phi,\psi} b_{r,1} + k_r K_{I,\theta,\phi,\psi}}{k_r K_{P,\theta,\phi,\psi}} \\ b_{a,2} = & \frac{k_r K_{P,\theta,\phi,\psi} b_{r,2} + k_r K_{I,\theta,\phi,\psi} b_{r,1}}{k_r K_{P,\theta,\phi,\psi}} \\ b_{a,3} = & \frac{k_r K_{I,\theta,\phi,\psi} b_{r,2}}{k_r K_{P,\theta,\phi,\psi}} \\ a_{a,1} = & k_r K_{P,\theta,\phi,\psi} + a_{r,1} \\ a_{a,2} = & k_r K_P b_1 + k_r K_I b_0 + a_{r,2} \\ a_{a,3} = & k_r K_P b_2 + k_r K_I b_{r,1} \\ a_{a,4} = & k_r K_I \end{cases} \tag{4.51}$$

As can be seen in (4.51), a high-order model with four poles and three zeros is obtained. At this point, the entire drone orientation control loop, ψ, is covered. The θ angle control loop will continue to be linked to the Y-axis motion since its variation translates into a linear acceleration in the Y-axis. Similarly, the ψ angle control loop will continue to be linked to the X-axis motion.

$X - Y$ Controller

The $X - Y$ plane motion control level consists of a cascade control where the inner level controls the linear velocity and the outer level controls the position in the X-Y plane. The output signals of this system are the reference angles θ and ϕ generated by the X and Y loops respectively received by the Attitude Controller.

Velocity Controller The two loops involved in this level of control have the same structure, a series combination of (4.51) and pure integrators with gain $g = 9.81\ [m/s^2]$. The controllers used are PI controllers in their parallel algorithm. For the theoretical development of the closed-loop controller, (4.54) and (4.55), the generic form (4.52) for the process and (4.53) for the controller will be considered. The parameterized closed-loop response is shown in (4.56).

$$G_{\dot{x},\dot{y}}\left(s\right) = \frac{gk_a\left(b_{a,0}s^3 + b_{a,1}s^2 + b_{a,2}s + b_{a,3}\right)}{s^5 + a_{a,1}s^4 + a_{a,2}s^3 + a_{a,3}s^2 + a_{a,4}s} \tag{4.52}$$

$$C_{\dot{x},\dot{y}}\left(s\right) = K_{P,\dot{x},\dot{y}} + K_{I,\dot{x},\dot{y}}\frac{1}{s} = \frac{K_{P,\dot{x},\dot{y}}s + K_{I,\dot{x},\dot{y}}}{s} \tag{4.53}$$

$$G_{CL,\dot{x},\dot{y}}\left(s\right) = \frac{k\left(K_Ps + K_I\right)\left(b_0s^3 + b_1s^2 + b_2s + b_3\right)}{s^6 + a_1s^5 + a_2s^4 + a_3s^3 + a_4s^2 + k\left(K_Ps + K_I\right)\left(b_0s^3 + b_1s^2 + b_2s + b_3\right)} \tag{4.54}$$

$$G_{CL,\dot{x},\dot{y}}\left(s\right) = \frac{k_{\dot{x}\dot{y}}\left(s^4 + b_{\dot{x}\dot{y},1}s^3 + b_{\dot{x}\dot{y},2}s^2 + b_{\dot{x}\dot{y},3}s + b_{\dot{x}\dot{y},4}\right)}{s^6 + a_{\dot{x}\dot{y},1}s^5 + a_{\dot{x}\dot{y},2}s^4 + a_{\dot{x}\dot{y},3}s^3 + a_{\dot{x}\dot{y},4}s^2 + a_{\dot{x}\dot{y},5}s + a_{\dot{x}\dot{y},6}} \tag{4.55}$$

$$G_{CL,\dot{x},\dot{y}}\left(s\right) = \begin{cases} k_{\dot{x}\dot{y}} = & k_agK_{P,\dot{x},\dot{y}}b_{a,0} \\ b_{\dot{x}\dot{y},1} = & \frac{K_{P,\dot{x},\dot{y}}b_{a,1}+K_{I,\dot{x},\dot{y}}b_{a,0}}{K_{P,\dot{x},\dot{y}}b_{a,0}} \\ b_{\dot{x}\dot{y},2} = & \frac{K_{P,\dot{x},\dot{y}}b_{a,2}+K_{I,\dot{x},\dot{y}}b_{a,1}}{K_{P,\dot{x},\dot{y}}b_{a,0}} \\ b_{\dot{x}\dot{y},3} = & \frac{K_{P,\dot{x},\dot{y}}b_{a,3}+K_{I,\dot{x},\dot{y}}b_{a,2}}{K_{P,\dot{x},\dot{y}}b_{a,0}} \\ b_{\dot{x}\dot{y},4} = & \frac{K_{I,\dot{x},\dot{y}}b_3}{K_{P,\dot{x},\dot{y}}b_{a,0}} \\ a_{\dot{x}\dot{y},1} = & a_{a,1} \\ a_{\dot{x}\dot{y},2} = & a_{a,2} + k_agK_{P,\dot{x},\dot{y}}b_{a,0} \\ a_{\dot{x}\dot{y},3} = & a_{a,3} + k_agK_{P,\dot{x},\dot{y}}b_{a,1} + k_agK_{I,\dot{x},\dot{y}}b_{a,0} \\ a_{\dot{x}\dot{y},4} = & a_{a,4} + k_agK_{P,\dot{x},\dot{y}}b_{a,2} + k_agK_{I,\dot{x},\dot{y}}b_{a,1} \\ a_{\dot{x}\dot{y},5} = & k_agK_{P,\dot{x},\dot{y}}b_{a,3} + k_agK_{I,\dot{x},\dot{y}}b_{a,2} \\ a_{\dot{x}\dot{y},6} = & k_agK_{I,\dot{x},\dot{y}}b_{a,3} \end{cases} \tag{4.56}$$

Position Controller The two loops involved in this level of control have the same structure, a series combination of (4.55) and pure integrators. The controllers used are *P* controllers. For the theoretical development of the closed loop controller, (4.59), the generic form (4.57) for the process and (4.58) for the controller will be considered. The parameterized closed-loop response is shown in (4.60).

$$G_{x,y}\left(s\right) = \frac{k_{\dot{x}\dot{y}}\left(s^4 + b_{\dot{x}\dot{y},1}s^3 + b_{\dot{x}\dot{y},2}s^2 + b_{\dot{x}\dot{y},3}s + b_{\dot{x}\dot{y},4}\right)}{s^7 + a_{\dot{x}\dot{y},1}s^6 + a_{\dot{x}\dot{y},2}s^5 + a_{\dot{x}\dot{y},3}s^4 + a_{\dot{x}\dot{y},4}s^3 + a_{\dot{x}\dot{y},4}s^2 + a_{\dot{x}\dot{y},4}s} \tag{4.57}$$

$$C_{x,y}\left(s\right) = K_{P,x,y} \tag{4.58}$$

$$G_{CL,x,y}\left(s\right)=\frac{k_{xy}\left(s^4+b_{xy,1}s^3+b_{xy,2}s^2+b_{xy,3}s+b_{xy,4}\right)}{s^7+a_{xy,1}s^6+a_{xy,2}s^5+a_{xy,3}s^4+a_{xy,4}s^3+a_{xy,5}s^2+a_{xy,6}s+a_{xy,7}} \tag{4.59}$$

$$G_{CL,x,y}\left(s\right)=\begin{cases} k_{xy}= & K_{P,x,y}k_{\dot{x}\dot{y}} \\ b_{xy,1}= & b_{\dot{x}\dot{y},1} \\ b_{xy,2}= & b_{\dot{x}\dot{y},2} \\ b_{xy,3}= & b_{\dot{x}\dot{y},3} \\ b_{xy,4}= & b_{\dot{x}\dot{y},4} \\ a_{xy,1}= & a_{\dot{x}\dot{y},1} \\ a_{xy,2}= & a_{\dot{x}\dot{y},2} \\ a_{xy,3}= & a_{\dot{x}\dot{y},3}+K_{P,x,y}k_{\dot{x}\dot{y}} \\ a_{xy,4}= & a_{\dot{x}\dot{y},4}+K_{P,x,y}k_{\dot{x}\dot{y}}b_{\dot{x}\dot{y},1} \\ a_{xy,5}= & a_{\dot{x}\dot{y},4}+K_{P,x,y}k_{\dot{x}\dot{y}}b_{\dot{x}\dot{y},2} \\ a_{xy,6}= & a_{\dot{x}\dot{y},4}+K_{P,x,y}k_{\dot{x}\dot{y}}b_{\dot{x}\dot{y},3} \\ a_{xy,7}= & K_{P,x,y}k_{\dot{x}\dot{y}}b_{\dot{x}\dot{y},4} \end{cases} \tag{4.60}$$

Altitude Controller

This control level is responsible for the vertical movement of the drone. It is a cascade control where the inner level is responsible for velocity control and the outer level is responsible for position control. In this case, the output is the thrust signal Ω_f $[rad/s]$.

Velocity Controller The process that regulates this controller is a pure integrator system with a gain of value $\Omega_f 8C_T/m$. The controller that is usually implemented in these systems to obtain a closed-loop first-order response is a proportional gain. However, it has been experimentally verified that this causes a very slow response in the outer loop if an overdamped system is desired. While this theoretical response is acceptable, in practice it causes the drone to take a long time to move away from the ground surface. This produces the ground effect to act more prominently on the drone and can make the drone's operation unstable. A PI controller is implemented in its parallel algorithm. For the theoretical development of the closed loop controller, (4.63), the generic form (4.61) for the process and (4.62) for the controller will be considered. The parameterized closed-loop response is shown in (4.63).

$$G_{\dot{z}}\left(s\right)=\Omega_e\frac{8C_T}{ms}=\frac{k}{s} \tag{4.61}$$

$$C_{\dot{z}}\left(s\right)=K_{P,\dot{z}}+K_{I,\dot{z}}\frac{1}{s}=\frac{K_{P,\dot{z}}s+K_{I,\dot{z}}}{s} \tag{4.62}$$

$$G_{CL,\dot{z}}\left(s\right)=\frac{\frac{K_{P,\dot{z}}s+K_{I,\dot{z}}}{s}\frac{8C_T}{ms}}{1+\frac{K_{P,\dot{z}}s+K_{I,\dot{z}}}{s}\frac{8C_T}{ms}}=\frac{kK_{P,\dot{z}}s+kK_{I,\dot{z}}}{ms^2+kK_{P,\dot{z}}s+kK_{I,\dot{z}}}=\frac{\frac{kK_{P,\dot{z}}}{m}\left(s+\frac{K_{I,\dot{z}}}{K_{P,\dot{z}}}\right)}{s^2+\frac{kK_{P,\dot{z}}}{m}s+\frac{kK_{I,\dot{z}}}{m}} \tag{4.63}$$

Position Controller This process controls the vertical position of the drone. It receives as input signal the position reference in the Z axis and generates as response the vertical velocity setpoint to be reached by the drone. The process to be controlled responds to the series combination of (4.63) and a pure integrator. The controller implemented at this level is a PI controller. For the theoretical development of the closed loop controller, (4.66), the generic form (4.64) for the process and (4.65) for the controller will be considered. The parameterized closed-loop response is shown in (4.67).

$$G_z\left(s\right)=\frac{\frac{kK_{P,\dot{z}}}{m}\left(s+\frac{K_{I,\dot{z}}}{K_{P,\dot{z}}}\right)}{s^3+\frac{kK_{P,\dot{z}}}{m}s^2+\frac{kK_{I,\dot{z}}}{m}s} \tag{4.64}$$

$$C_z\left(s\right)=K_{P,z}+K_{I,z}\frac{1}{s}=\frac{K_{P,z}s+K_{I,z}}{s} \tag{4.65}$$

$$G_{CL,z}\left(s\right)=\frac{\frac{kK_{P,\dot{z}}}{m}\left(s+\frac{K_{I,\dot{z}}}{K_{P,\dot{z}}}\right)\left(K_{P,z}s+K_{I,z}\right)}{s^4+\frac{kK_{P,\dot{z}}}{m}s^3+\frac{kK_{I,\dot{z}}}{m}s^2+\frac{kK_{P,\dot{z}}}{m}\left(s+\frac{K_{I,\dot{z}}}{K_{P,\dot{z}}}\right)\left(K_{P,z}s+K_{I,z}\right)} \tag{4.66}$$

$$G_{CL,z}\left(s\right)=\frac{k_z\left(s^2+b_{z,1}s+b_{z,2}\right)}{s^4+a_{z,1}s^3+a_{z,2}s^2+a_{z,3}s+a_{z,4}}\begin{cases}k_z= & \frac{kK_{P,\dot{z}}K_{P,z}}{m}\\ b_{z,1}= & \frac{K_{I,\dot{z}}}{K_{P,\dot{z}}}+\frac{K_{I,z}}{K_{P,z}}\\ b_{z,2}= & \frac{K_{I,\dot{z}}K_{I,z}}{K_{P,\dot{z}}K_{P,z}}\\ a_{z,1}= & \frac{kK_{P,\dot{z}}}{m}\\ a_{z,2}= & \frac{kK_{I,\dot{z}}}{m}+\frac{kK_{P,\dot{z}}K_{P,z}}{m}\\ a_{z,3}= & \frac{kK_{P,\dot{z}}K_{P,z}}{m}\left(\frac{K_{I,\dot{z}}}{K_{P,\dot{z}}}+\frac{K_{I,z}}{K_{P,z}}\right)\\ a_{z,4}= & \frac{kK_{I,\dot{z}}K_{I,z}}{m}\end{cases} \tag{4.67}$$

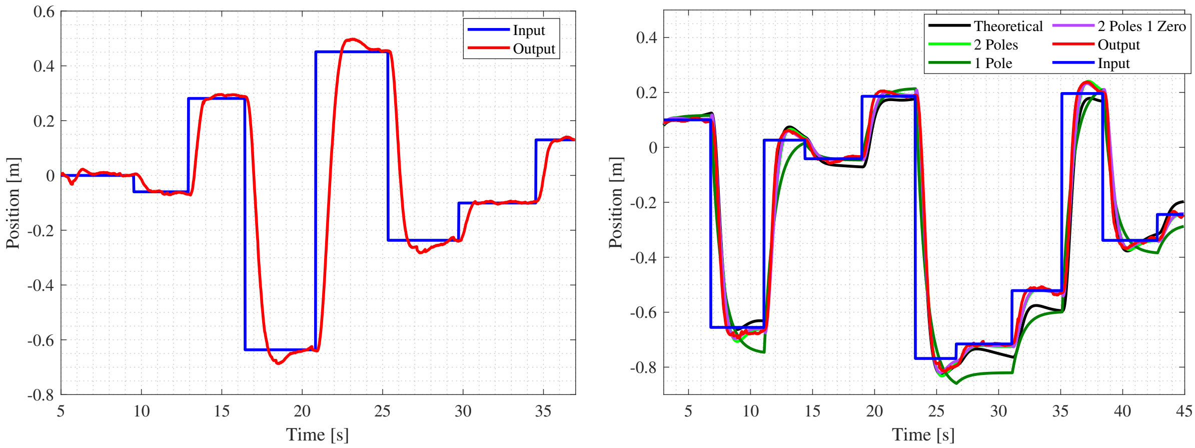
Throughout this section, all the theoretical closed-loop models involved in drone position control have been obtained. It can be noted that, in all cases, high-order models have been obtained. It is possible to work with these models or to reduce their order by including filters between controllers. However, it should be kept in mind that filters can slow down the dynamics of the system too much. An alternative solution with better results is to use lower-order approximations. These approximations can be performed using specific techniques for order reduction or by estimating low-order models from experimental data.

4.4 Parameters identification

After determining the theoretical linear model of quadcopters, in this section, the real model of the Crazyflie 2.X robot will be identified using the Matlab System Identification toolbox. In this process, the identification will be performed in a closed loop by using the target position as an input signal and, as output, the instant robot position. The process will be carried out individually for the movement in each axis to avoid the coupling effects in the data used to identify each model. The identification process is performed for the basic configuration of the Crazyflie 2.1 sensor (flow deck and Lighthouse deck, total weight 31.3 $[g]$) and for the case in which it includes the multiranger deck (weight 2.3 $[g]$). In this way, the results will show the robustness of the drone's internal control architecture versus a mass variation of 7.34 %. All data used have been linearized to the initial state of the experience.

4.4.1 X axis

Figure 4.3a shows the input and output data used to estimate the transfer function to the basic sensor setup. The optimal balance between the accuracy and order of the transfer function is analyzed using different transfer functions through cross-validation. The first option checked is the theoretical model (4.60), which provided an accuracy of 87.48%. To reduce the order of the model, the new transfer functions have one pole, two poles, and two poles with a zero. The results obtained provide an accuracy of 79.14%, 91.57%, and 92.2%, respectively. Figure 4.3b shows the graphical results of the validation process. Therefore, it can be concluded that the model that offers greater accuracy with a reduced order corresponds to the transfer function with two poles (4.68). This model has a gain $k = 1.014$, natural frequency $\omega_n = 2.005$ $[rad/s]$, and dumping factor $\zeta = 0.677$.



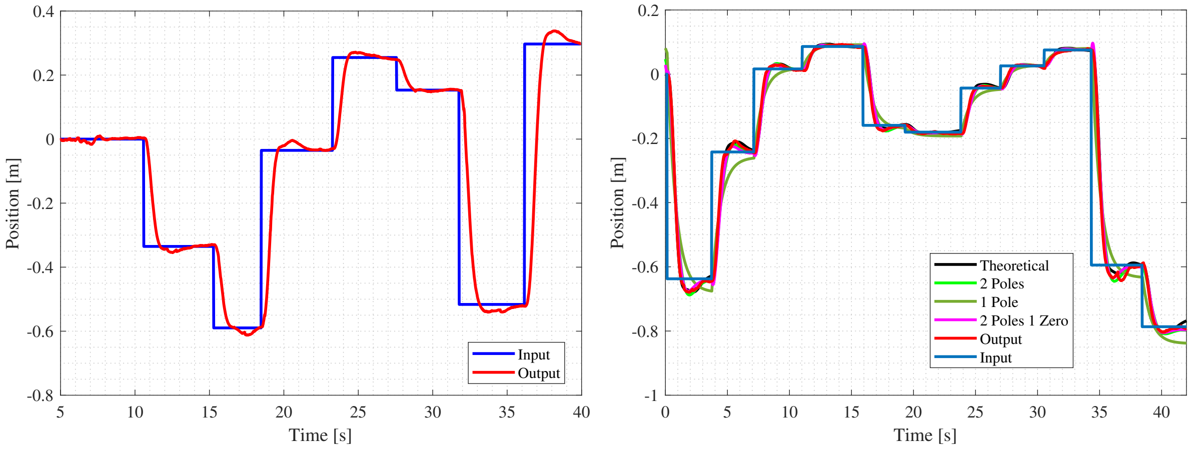
(a) Data used to estimation.

(b) Cross-validation.

Figure 4.3. Crazyflie X-axis model estimation. Basic setup.

$$G_{CL,B,x} = \frac{4.0479}{s^2 + 2.714s + 4.019} \tag{4.68}$$

The process is repeated for the setup with the multiranger deck. Figure 4.4a, shows the input and output data used in the estimation process. Different transfer functions are evaluated by cross-validation. The first option checked is the theoretical model (4.60), which provided an accuracy of 96.24%. Again, the reduced order models considered are transfer functions with one pole, two poles, and two poles with a zero. The results obtained provide an accuracy of 87.04%, 95.05%, and 95.64%, respectively. Figure 4.4b shows the graphical results of the validation process. Therefore, it can be concluded that the model that offers greater accuracy with a reduced order corresponds to the transfer function with two poles (4.69). This model has a gain $k = 1.012$, natural frequency $\omega_n = 2.270$ $[rad/s]$, and dumping factor $\zeta = 0.661$. The results show a better estimation of the model and similar parameters in both cases, except for a small increase in the natural frequency. It indicates that the control architecture is robust enough to assimilate small changes in the drone equipment.



(a) Data used to estimation.

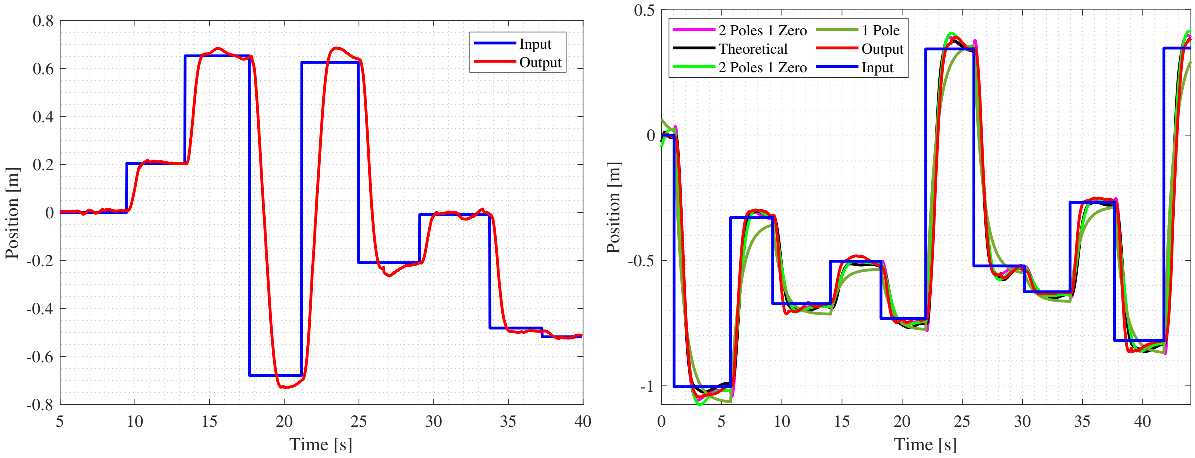
(b) Cross-validation.

Figure 4.4. Crazyflie X-axis model estimation. Multiranger setup.

$$G_{CL,M,x} = \frac{5.219}{s^2 + 2.998s + 5.154} \tag{4.69}$$

4.4.2 Y axis

Figure 4.5a shows the input and output data used to estimate the transfer function to the basic sensor setup. The optimal balance between the accuracy and order of the transfer function is analyzed using different transfer functions by cross-validation. The first option checked is the theoretical model (4.60), which provided an accuracy of 89.73%. To reduce the model order, the new transfer functions have one pole, two poles, and two poles with a zero. The results obtained provide an accuracy of 77.03%, 88.31%, and 89.07%, respectively. Figure 4.5b shows the graphical results of the validation process. Therefore, it can be concluded that the model that offers greater accuracy with a reduced order corresponds to the transfer function with two poles (4.70). This model has a gain $k = 1.017$, natural frequency $\omega_n = 2.017$ $[rad/s]$, and dumping factor $\zeta = 0.685$.



(a) Data used to estimation.

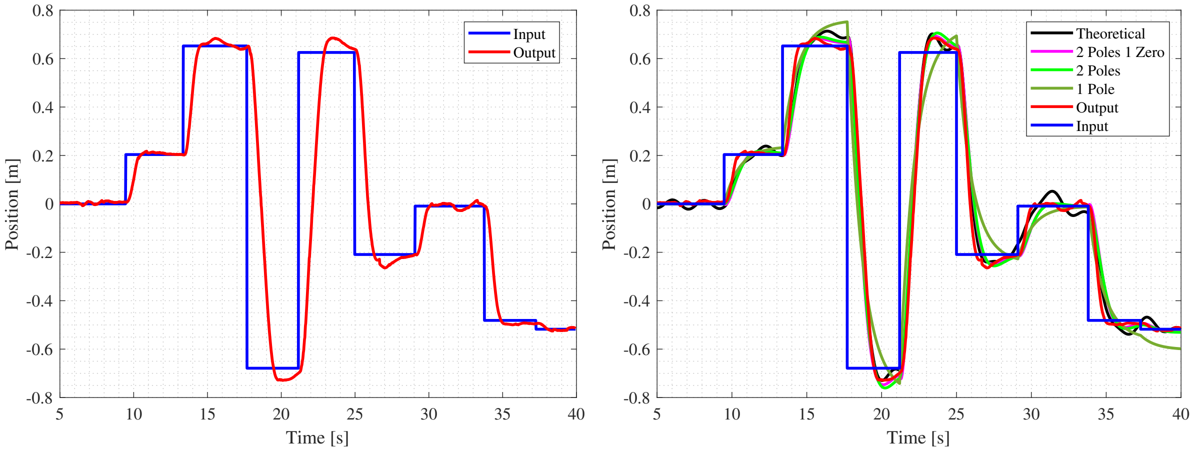
(b) Cross-validation.

Figure 4.5. Crazyflie Y-axis model estimation. Basic setup.

$$G_{CL,B,y} = \frac{4.136}{s^2 + 2.763s + 4.068} \tag{4.70}$$

The process is repeated for the setup with the multiranger deck. Figure 4.6a, shows the input and output data used in the estimation process. Different transfer functions are evaluated by cross-validation. The first option checked is the theoretical model (4.60), which provide an accuracy of 90.63%. Again, the reduced order models considered are transfer functions with one pole, two poles, and two poles with a zero. The results obtained provided an accuracy of 79.21%, 87.89%, and 88.31%, respectively. Figure 4.6b shows the graphical results of the validation process. Therefore, it can be concluded that the model that offers greater accuracy with a reduced order corresponds to the transfer function with two poles (4.71). This model has a gain $k = 1.022$, natural frequency $\omega_n = 1.719$ $[rad/s]$, and dumping factor $\zeta = 0.692$. The results show a better estimation of the model and similar parameters in both cases, except for a small decrease in the natural frequency. It indicates that the control architecture is robust enough to assimilate small changes in drone equipment. In addition, the results obtained on the Y-axis closely correspond to those obtained on the X-axis, as expected due to the symmetry of the robots.

$$G_{CL,M,y} = \frac{3.022}{s^2 + 2.384s + 2.957} \tag{4.71}$$



(a) Data used to estimation.

(b) Cross-validation.

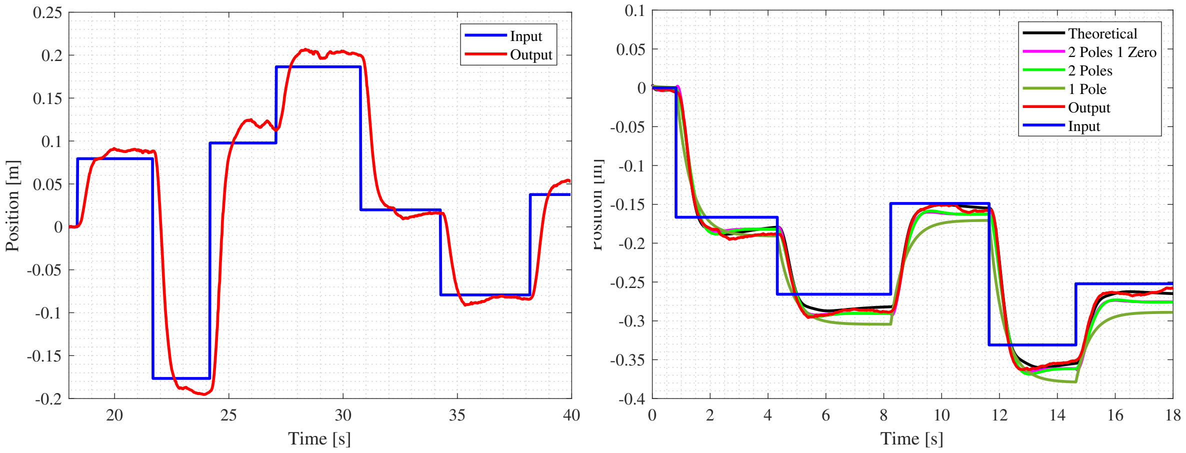
Figure 4.6. Crazyflie Y-axis model estimation. Multiranger setup.

4.4.3 Z axis

Figure 4.7a shows the input and output data used to estimate the transfer function to the basic sensor setup. The optimal balance between the accuracy and order of the transfer function is analyzed using different transfer functions by cross-validation. The first option checked is the theoretical model (4.67), which provided an accuracy of 94.26%. To reduce the model order, the new transfer functions have one pole, two poles, and two poles with a zero. The results obtained provided an accuracy of 77.07%, 91.06%, and 91.54%, respectively. Figure 4.7b shows the graphical results of the validation process. Therefore, it can be concluded that the model that offers greater accuracy with a reduced order corresponds to the transfer function with two poles (4.72). This model has a gain $k = 1.094$, natural frequency $\omega_n = 3.326$ $[rad/s]$, and dumping factor $\zeta = 0.734$.

$$G_{CL,B,z} = \frac{12.1}{s^2 + 4.883s + 11.06} \tag{4.72}$$

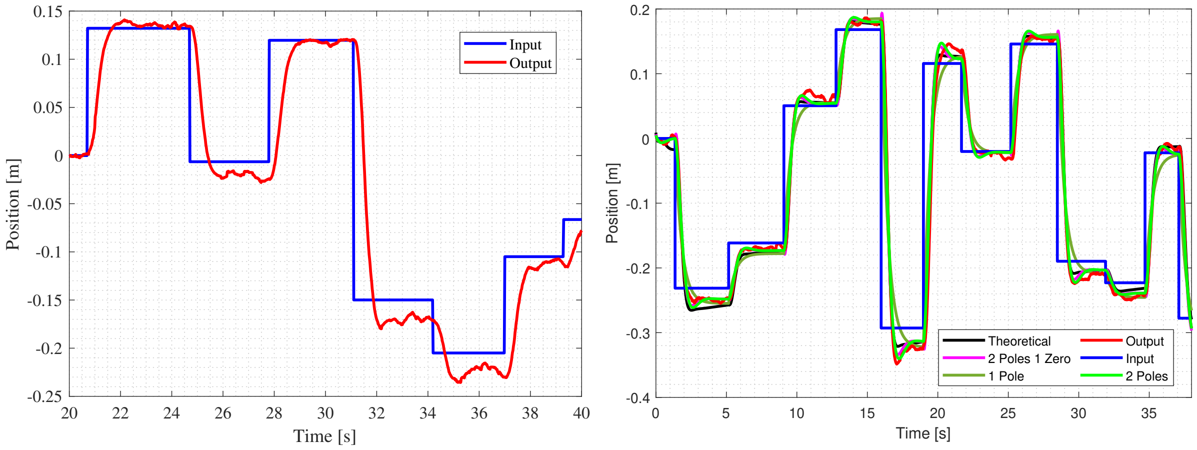
The process is repeated for the setup with the multiranger deck. Figure 4.8a, shows the input and output data used in the estimation process. Different transfer functions are evaluated by cross-validation. The first option checked is the theoretical model (4.67), which provided an accuracy of 93.25%. The reduced order models considered again are transfer functions with one pole, two poles, and two poles with a zero. The results obtained provided an accuracy of 83.14%, 92.71%, and 93.68%, respectively. Figure 4.8b shows the graphical results of the validation process. Therefore, it can be concluded that the model that offers greater accuracy with a reduced order corresponds to the transfer function with two poles (4.73). This model has a gain $k = 1.073$, natural frequency $\omega_n = 3.739$ $[rad/s]$, and dumping factor $\zeta = 0.6836$. The results show a better estimate, but a system with lower damping due to the greater inertia in the robot is more significant on the Z-axis due to the effect of gravity.



(a) Data used to estimation.

(b) Cross-validation.

Figure 4.7. Crazyflie Z-axis model estimation. Basic setup.



(a) Data used to estimation.

(b) Cross-validation.

Figure 4.8. Crazyflie Z-axis model estimation. Multiranger setup.

$$G_{CL,M,y} = \frac{12.25}{s^2 + 4.62s + 11.42} \tag{4.73}$$

4.5 Experimental evaluation and discussion

Once the robot model has been established, its experimental validation is carried out on real and simulated robots. To this end, a case study focused on drone altitude control is presented next. The position and trajectory control algorithm for a Crazyflie is then designed. For this, a control structure of the parallel PID type is implemented according to [204]:

$$u_k = P_k + I_k + D_k \tag{4.74}$$

$$P_k = K_p \cdot e_k \tag{4.75}$$

$$I_k = I_{k-1} + K_i \cdot T_k \cdot e_{k-1} \tag{4.76}$$

$$D_k = \frac{K_d}{K_d + N_d \cdot T_k \cdot K_d} D_{k-1} + \frac{K_d \cdot N_d}{\frac{K_d}{K_p} + N_d \cdot T_k} \cdot (e_k - e_{k-1}) \tag{4.77}$$

where P_k, I_k y D_k are the proportional, integral and derivative actions, respectively, T_k is the time since last controller execution and N_d is the characteristic parameter of the derivative filter whose value is 20. In this analysis, three cases were studied. The first case involved periodic control with a period of 10 $[ms]$. The second case adopted an event-based controller with a constant threshold of 1 $[cm]$. The third case also uses an event-based controller but with an adaptive trigger threshold based on a minimum threshold ($c_o^i = 1$ $[cm]$), setpoints parameter ($a_i = 1.5$), and noise rejection (see (1.2) for the definition of these parameters). The PID control gains are kept the same in all three cases.

Figure 4.9 shows the path followed by the drone during the experiences. This path consists of a trajectory along a square of 1 $[m]$ on the XY plane. On the Z-axis, there is a 20% reduction in the setpoint during $t : 10$ $[s] < t_k < 20$ $[s]$. Figure 4.10a shows the behaviour of the robot on

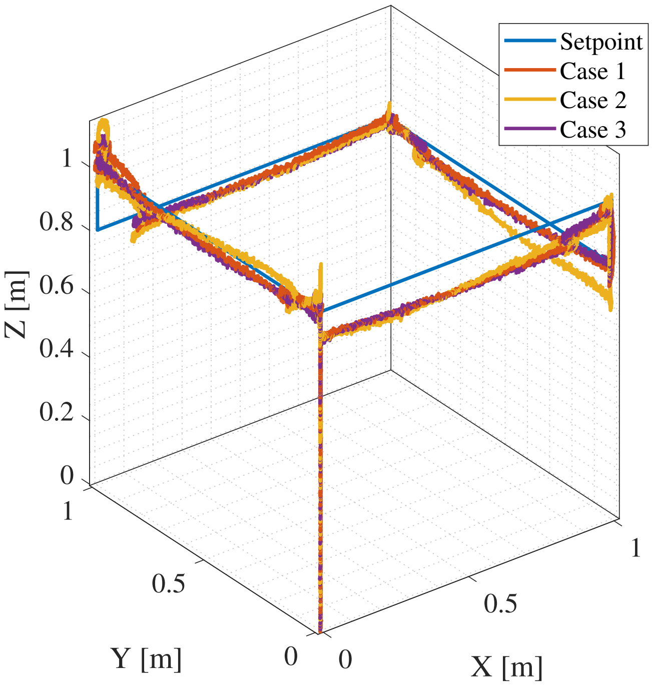
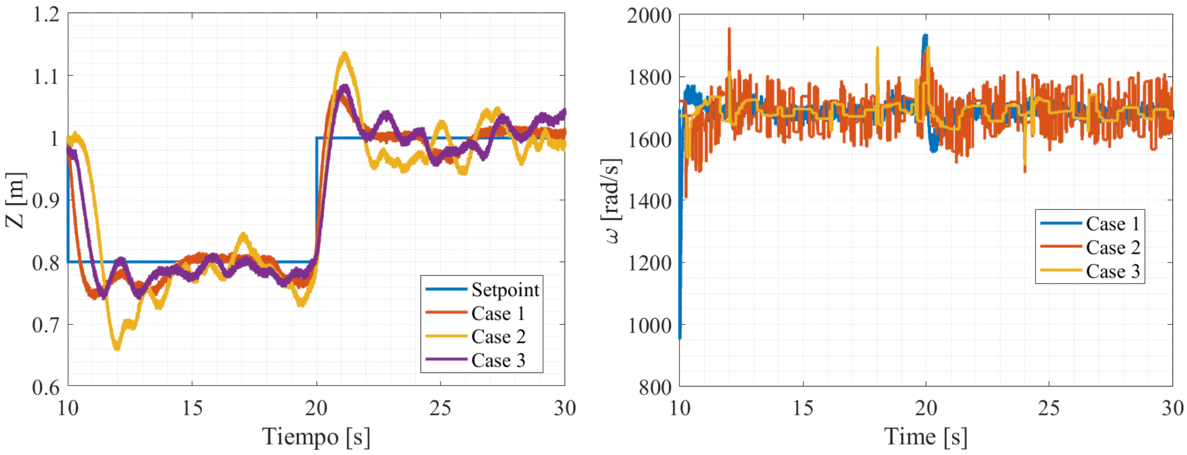


Figure 4.9. Trajectory followed.

the Z-axis. Figure 4.10b shows the evolution of the control signal applied to the rotors.



(a) Z-Axis.

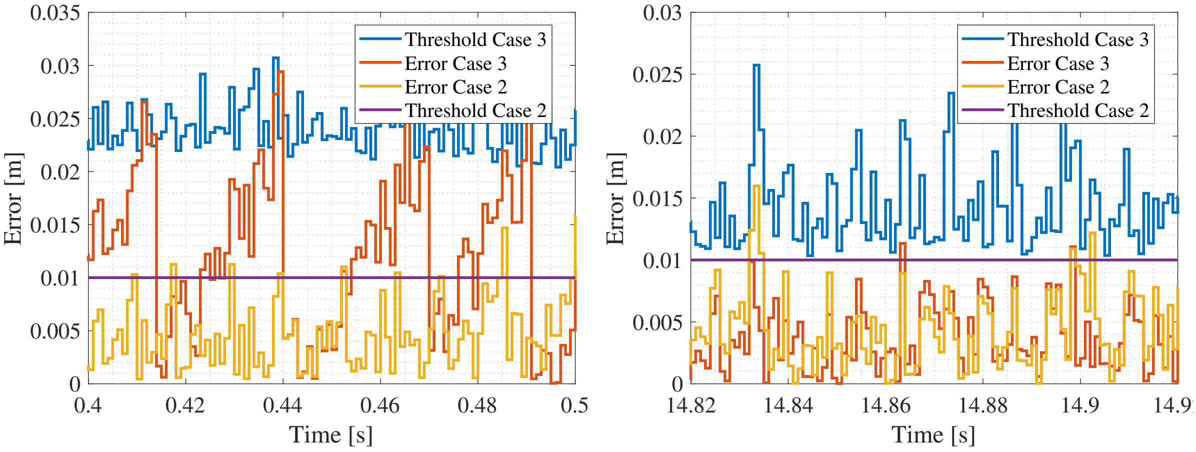
(b) Control signal.

Figure 4.10. Graph.

After analyzing the controllers, it was found that there are no significant differences in the time frame. Hence, the event-based proposal allows the utilization of the same controller parameters. To perform a deeper analysis of the results, IAE, ITAE, and the number of controller executions N are evaluated. Table 4.3 shows the results obtained. In this scenario, the differences are highlighted. As expected, the use of event-based strategies results in a significant reduction in the number of controller executions. For case 2, the reduction is 65.77%, while for case 3, a decrease of 93.0% is achieved. In addition, this third case represents an improvement in the IAE of 3.29%. However, the ITAE represents a worsening of 20.06% (below that achieved in case 2). Therefore, by adopting a compromise solution, it is demonstrated that event-based implementation with an adaptive threshold substantially improves over periodic performances. For a deeper understanding of the event generation triggering, Figure 4.11 illustrates the performance of the error over two different time periods. Upon comparing the results, it becomes evident that the noise-related term has greater significance during the steady regime as opposed to the transient regime.

Index	**Case 1**	**Case 2**	**Case 3**
N	3000	1027	207
$IAE\ [m]$	2.2568	2.6417	2.1825
$ITAE\ [m \cdot s]$	12.485	21.224	14.989

Table 4.3. Analytical results for square trajectory.



(a) Transient regime.

(b) Steady regime.

Figure 4.11. Events generation.

4.6 Conclusions

This chapter presents the physical modeling of quadcopters, as they are the air vehicles included in RP. An internal representation based on state space has been used, which includes force and momentum equations. A linear state space model is also derived, resulting in a MIMO system. A common two-level control architecture for this type of vehicle has been presented made up of a cascade system for each one. The upper level, which includes the altitude controller, the X-Y controller, and the Yaw controller, is responsible for controlling the position and speed of the robot. The other level manages the stability control, and it is defined by the attitude and rate controllers. Several control decoupling architectures and conventional PID-based implementations are analyzed for the different controllers of the system.

Parameters identification of the theoretical model has been performed from the real Crazyflie 2.X robot experimental data. This identification has been carried out independently for each axis to avoid coupling effects. Finally, an experimental evaluation focused on drone altitude control has been presented. The performed experiences demonstrate the potential of event-based control strategies to improve the efficiency of systems with computational constraints.

Chapter 5

Formation control: Experimental evaluation

5.1 Introduction

Among the cooperative tasks that a multi-agent system can perform, achieving a formation is used, for instance, in sampling, monitoring, or surveillance tasks [205–207]. A formation aims to drive multiple agents to achieve prescribed constraints on their states.

In general, the interaction between agents is modeled in terms of a graph, in such a way that two agents are connected in the graph if they need to interact with each other to achieve the control objective. There are different ways to characterize a formation that leads to different formation control strategies. If we focus on this characterization in terms of the sensing capability and the topology of agents, the natural question is what variables are sensed and what variables are controlled by the multi-agent systems to achieve the formation, and this yields the categorization introduced in Chapter 1 for the existing control strategies [44], that is recalled next:

- Position-based control: Agents can sense their own positions with respect to a global coordinate system, and they can actively control them to achieve the desired formation so that the agents can move without interacting with each other. Thus, this type of strategy will not be considered in this chapter.
- Displacement-based control: Agents can sense and control relative positions to their neighbors to achieve the desired formation. Since this is defined in terms of desired displacements with respect to a global coordinate system, the orientation of the global coordinate systems is required.
- Distance-based control: Agents measure the relative positions to their neighbors, but the variables that are controlled are the inter-agent distances. Thus, the orientation of local coordinate systems is not necessarily aligned with each other. However, the existing control laws are non-linear so that the analysis is more challenging.

Regarding the topology, the first classifications yield to directed or undirected formations. In the first case, the flow of information between some pairs of agents is unidirectional. For instance, if at least one agent moves freely and others control relative positions/distances to it, then the graph is directed and this is usually known as a leader-follower approach, and the formation will move according to the leader trajectory. By contrast, if the communication between agents is bidirectional, the movement of the formation will depend on the dynamics and the initial conditions of the agents.

Despite the directionality of the existing links between agents, these graph interconnections can have different meanings and this generally depends on the measurement capabilities. For instance, if the agents are equipped with instrumentation that allows the measurement of the variables that need to be sensed, then communication might not be required [208]. However, if these variables cannot be directly accessed by the sensors, then the agents would need to transmit the measurements to other nodes in the network, and then the topology is interpreted as a communication graph [209]. In this case, aspects that concern the communication such as the protocol or the network structure can affect the control performance, and thus, this is an issue that should handled with care. Moreover, the frequency of the communication has a direct impact on the control performance but also on the battery life in autonomous devices. Enlarging the intervals of communication can result in a degradation of the control. However, the use of protocols based on events [62], which consider the state of the system to decide when to transmit information, provides a better trade-off between performance and average communication frequency.

Moreover, when the number of agents in the system is large, it is usually referred to as a swarm. A major challenge in controlling multi-agent swarms is the synthesis of controllers in a scalable manner, i.e., the control design cannot become more complex for a large number of robots. In this regard, a decentralization of the control architecture is always preferred. In the context of swarm robotics, decentralization usually means that the computation is executed onboard. However, in some devices the computation capacity is limited, and sometimes, there exist tasks that require a central unit, for example, in some indoor positioning systems. Additionally, when the system includes some virtual agents or digital twins, then the concern is on the computational resources that the simulation tools require. Therefore, we can assert that scalability is sometimes not trivial, and the study of how much impact have the increase in the number of agents over the control is an interesting aspect.

This chapter will focus on the performance evaluation of the developed experimental platform, Robotic Park (RP). For that purpose, several experiences have been performed for different setups in the context of formation control of MAS. More specifically, we study the formation control of MARS in the 3D space but imposing geometric constraints, such as placing the formation over virtual lines or surfaces. The features that have been covered in such systematic evaluation are:

- Scalability: The impact over the control performance and other parameters that have the increase in the number of agents. Since the number of physical agents is limited by the available robots and other constraints imposed, for example, by the positioning system, we enlarge the swarm with virtual agents. Then, different aspects such as the CPU usage, the Real-Time Factor (RTF), and the system performance have been studied.

- Control architecture: The control architecture (see Section 2.5) follows a hierarchical structure, in which some of the layers allow different implementations: centralized in a central unit, decentralized in different ROS 2 nodes, or onboard. Therefore, the control performance for these cases and also in connection with the previous feature will be examined.

- Communication protocol: First, the impact of the frequency of communication over the control performance is studied in the case of periodic transmissions. After that, an event-based protocol is implemented and the impact of the parameters of the trigger function is covered.

The focus is put on undirected topologies and distance-based controllers, although the platform allows the implementation of directed graphs and other control approaches (see for instance [151]).

The rest of the chapter is organized as follows. Section 5.2 provides some background for formation control. Then, the control law for the coordination between agents that has been implemented is presented in Section 5.3, as well as some details about the event-based communication protocol and Mixed Reality (MR) implementation. The analysis of the performance of the platform concerning the different features that have been covered is given in Sections 5.4 (scalability) and 5.5 (control architecture). Finally, Section 5.6 presents the conclusions of the chapter.

5.2 Preliminaries

5.2.1 Graph theory

Consider a set $\mathcal{N}$ of N agents. The topology of the MAS can be modeled as a static undirected graph $\mathcal{G}$. This section reviews some facts from algebraic graph theory [210]. The graph $\mathcal{G}$ is described by the set of agent-nodes $\mathcal{V}$ and the set of edges $\mathcal{E}$.

For each agent i, $\mathcal{N}_i$ represents the neighborhood of i, i.e., $\mathcal{N}_i = \{j \in \mathcal{V} : \ (i,j) \in \mathcal{E}\}$. Note that $|\mathcal{N}_i| = \deg v_i$, where $|\cdot|$ represents the cardinality of the set $\mathcal{N}_i$ and deg is the degree of the vertex v_i associated to the node i.

Assume that the edges have been labeled as e_k and arbitrarily oriented, and its cardinality is labeled as N_e. Then the incidence matrix $H(\mathcal{G}) = [h_{ik}] \in \mathbb{R}^{N \times N_e}$ is defined as $h_{ik} = -1$ if v_i is the tail of the edge e_k, $h_{ik} = 1$ if v_i is the head of e_k, and $h_{ik} = 0$ otherwise. The

Laplacian matrix $L(\mathcal{G}) \in \mathbb{R}^{N \times N}$ of a network of agents is defined as $L(\mathcal{G}) = H(\mathcal{G})H^{\top}(\mathcal{G})$. The Laplacian matrix $L(\mathcal{G})$ is positive semidefinite, and if $\mathcal{G}$ is connected and undirected, then $0 = \lambda_1(\mathcal{G}) < \lambda_2(\mathcal{G}) \leq \cdots \leq \lambda_N(\mathcal{G})$, where $\{\lambda_j(\mathcal{G})\}$ are the eigenvalues of $L(\mathcal{G})$. The adjacency matrix of $\mathcal{G}$ is $A(\mathcal{G}) = [a_{ij}]$, where $a_{ij} = 1$ if there is an edge between two vertices v_i and v_j, and 0 otherwise. Matrices $H(\mathcal{G})$, $L(\mathcal{G})$ and $A(\mathcal{G})$ can be simply denoted by H, L and A, respectively, when it is clear from the context.

5.2.2 Graph rigidity

A framework is a realization of a graph at given points in Euclidean space. We consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N vertices embedded in $\mathbb{R}^m$, with $m = 2$ or $m = 3$ by assigning to each vertex i a location $p_i \in \mathbb{R}^m$. Define the composite vector $p = (p_1, \ ..., \ p_n) \in \mathbb{R}^{mn}$. A framework is a pair $(\mathcal{G}, p)$.

For every framework $(\mathcal{G}, p)$, we define the rigidity function $f_{\mathcal{G}}(p) : \mathbb{R}^{2N} \to \mathbb{R}^{N_e}$ given by

$$f_{\mathcal{G}}(p) = (\ldots, \|z_k\|^2, \ldots),$$

where $\|z_k\|^2 = \|p_i - p_j\|^2$, corresponds to the edge k in $\mathcal{E}$ that connects two vertices i and j. Note that this function is not unique and depends on the ordering given to the edges.

The formal definition of rigidity and global rigidity can be found in [211]. But roughly speaking, a framework $(\mathcal{G}, p)$ is rigid if it is not possible to smoothly move some vertices of the framework without moving the rest while maintaining the edge lengths specified by $f_{\mathcal{G}}(p)$.

Let us take the following approximation of $f_{\mathcal{G}}(p)$:

$$f_{\mathcal{G}}(p + \delta p) = f_{\mathcal{G}}(p) + R(p)\delta p + O(\delta p^2),$$

where $R(p) = J_{f_{\mathcal{G}}}(p)$ denotes the Jacobian matrix of $f_{\mathcal{G}}(p)$, and δp is an infinitesimal displacement of p. The matrix $R(p)$ is called the *rigidity matrix* of the framework $(\mathcal{G}, p)$. Analyzing the properties of $R(p)$ allows to infer further properties of the framework. Next we present some existing results:

Definition 1. [211]. A framekwork $(\mathcal{G}, p)$ is infinitesimally rigid if $\text{rank}(R(p)) = 2N - 3$ in $\mathbb{R}^2$ or $\text{rank}(R(p)) = 3N - 6$ in $\mathbb{R}^3$.

Therefore, the kernel of $R(p)$ has dimension 3 and 6 in $\mathbb{R}^2$ and $\mathbb{R}^3$, respectively, which corresponds to the rigid body motions that make that $R(p)\delta p = 0$ with $\delta p \neq 0$. In $\mathbb{R}^2$, this corresponds to translation along x, translation along y, and the rotation about z. Similary, in $\mathbb{R}^3$ the rigid body motions are translations along x, y, z and rotations about x, y, z.

Finally, the concept of *minimum rigidity* is introduced.

Definition 2. [212]. A graph is minimally rigid if it is rigid and the removal of a single edge causes it to lose rigidity. Mathematically, this condition can be checked by the number of edges N_e, so that if $N_e = 2N - 3$ in $\mathbb{R}^2$ or $N_e = 3N - 6$ in $\mathbb{R}^3$ the graph is minimally rigid.

5.3 Formation control

The control architecture follows a hierarchical structure, where different control levels are defined, as shown in Figure 5.1. The controllers defined at the bottom levels are in charge of the positioning of the robots. These controllers have been presented in the previous chapters. Therefore, this chapter will focus on the coordination level, and more specifically on coordination controllers that allow the achievement of a formation.

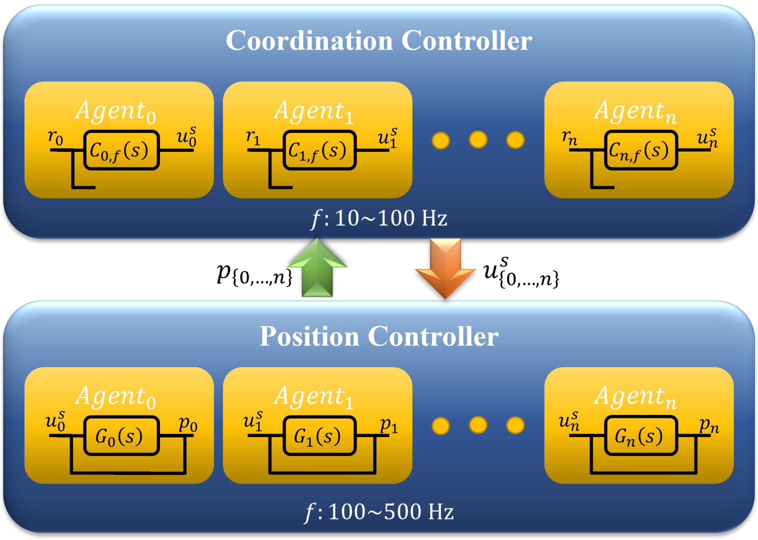


Figure 5.1. Multi-Robot hierarchical control.

Since the stabilization controllers are designed and implemented separately, this allows us to consider simplified models of the robots for the upper control levels. Indeed, single integrator models will be used for the target positions.

5.3.1 Control law

In [213], a distributed control law is proposed for formation control, where the control law is derived from a potential function based on an undirected and infinitesimally rigid graph. More specifically, the potential function has the form

$$W_1 = \frac{1}{4} \sum_{(i,j)\in\mathcal{E}} (d_{ij}^2 - {d_{ij}^*}^2)^2, \tag{5.1}$$

where $d_{ij} = \|p_i - p_j\|$ and d_{ij}^* is the prescribed distance for the edge $(i,j) \in \mathcal{E}$. The gradient descent control law for each agent i derived from the potential function (5.1) is then

$$u_i = -\nabla_{p_i} W_1 = -\sum_{j \in \mathcal{N}_i} (d_{ij}^2 - {d_{ij}^*}^2)(p_i - p_j). \tag{5.2}$$

The control system $\dot{p}_i = u_i$ with u_i (5.2) can be studied using the infinitesimal distance rigidity property as follows [214]:

Theorem 1. *All agents close to a target infinitesimally distance rigid formation with the controller* (5.2) *exponentially converge to a formation consistent with desired distances.*

Note that even though $W_1 = 0$ in (5.1) only at the desired formation, i.e., when $d_{ij} = d_{ij}^*$, there exist other equilibria sets that correspond to $\nabla_{p_i} W_1 = 0$, including collinearity (in $\mathbb{R}^2$) and collinearity and coplanarity (in $\mathbb{R}^3$) of the agents.

5.3.2 Virtual constraints

In this thesis, we focus on the deployment of the formation restricted to subspaces, that in $\mathbb{R}^3$ are surfaces or lines. This provides an alternative formalism to the definition of rigid formation in $\mathbb{R}^3$ that would require a number of links between agents of $3N - 6$ [214].

To this end, we define an optimization problem subject to constraints and Lagrange multipliers are used. Let us define a function f that measures the distance of an agent i to the subspace as

$$f(x, y, z, p_i) = (x - p_{i,x})^2 + (y - p_{i,y})^2 + (z - p_{i,z})^2 \tag{5.3}$$

where $p_i = (p_{i,x}, p_{i,y}, p_{i,z})^\top$ are the coordinates in $\mathbb{R}^3$ of the robot i.

The constraints given by the subspace are represented by a set of functions

$$g(x, y, z) = \{g_\ell(x, y, z) = 0,\ \ell = 1, \ldots, n_c\},$$

where the value of the number of constraints n_c depends on the dimension of the subspace. Then, the optimization problem is

$$\min_{p \in \mathbb{R}^3} f(p, p_i) \tag{5.4}$$

$$\text{subject to } g_\ell(p) = 0,\ \ell = 1, \ldots, n_c. \tag{5.5}$$

Let us consider the Lagrangian for each agent i

$$\mathcal{L}(p, p_i, \lambda) = f(p, p_i) + \sum_{\ell=1}^{n_c} \lambda_\ell c_\ell(p),$$

where λ_ℓ are the Lagrange multipliers. Then, $p_i^* = (x_i^*, y_i^*, z_i^*)^\top$ is a minimum of the optimization problem (5.4)-(5.5) for agent i if

$$\nabla f\left(p_i^*, p_i\right) + \nabla g\left(p_i^*\right) \cdot \lambda = 0, \tag{5.6}$$

and $g_\ell(p_i^*) = 0$, $\forall \ell = 1, \ldots, n_c$. This point p_i^* represents the point of the subspace with a minimum value of (5.3) for the agent i.

Then, we can define a potential function

$$W_2 = \frac{1}{2} \sum_{i=1}^{N} f(p_i^*, p_i), \tag{5.7}$$

and the control action derived from it as

$$u_i = \nabla_{p_i} W_2 = -(p_i - p_i^*). \tag{5.8}$$

Thus, considering both the term for the formation control (5.2) and the term to of the virtual constraints (5.8), we obtain the following control law for the coordination controller:

$$u_i = -\kappa_1 \sum_{j \in \mathcal{N}_i} (d_{ij}^2 - {d_{ij}^*}^2)(p_i - p_j) - \kappa_2 (p_i - p_i^*), \tag{5.9}$$

where $\kappa_1, \kappa_2 \in \mathbb{R}_{>0}$.

Note the potential function $W = W_1 + W_2$, where W_1 is (5.1) and W_2 is (5.7), is positive definite and $W = 0$ when the system achieves the formation and satisfies the constraints of the subspace defined by $g(x, y, z) = 0$. Then, the control law (5.9) allows the local satisfaction of the control objective asymptotically. The result follows from Theorem 1.

Next, we present some examples of virtual constraints which provide an analytical expression for p_i^*.

Lines

Let us consider the case of a straight line with parametric equations:

$$\begin{cases} g_x\left(x, y, z, \gamma\right) = x - p_{0,x} - m_x \gamma \\ g_y\left(x, y, z, \gamma\right) = y - p_{0,y} - m_y \gamma \\ g_z\left(x, y, z, \gamma\right) = z - p_{0,z} - m_z \gamma \end{cases} \tag{5.10}$$

where $p_0 = (p_{0,x}, p_{0,y}, p_{0,z})^\top$ is a point of the straight line and $m = (m_x, m_y, m_z)^\top$ represents the direction.

If the optimization problem (5.4)-(5.5) is solved, this provides an analytical value for p_i^* as:

$$p_i^* = p_0 - m\frac{(p_0 - p_i)^\top \cdot m}{\|m\|^2} \tag{5.11}$$

Plane

In the case of a plane, the constraint is given by a single equation

$$g(x, y, z) = ax + by + cz - d.$$

If we denote by $v = (a, b, c)^\top$, which represents the normal vector to the plane, the solution of the optimization problem (5.4)-(5.5) provides the following value for p_i^*:

$$p_i^* = \frac{v^\top p_i - d}{v^\top v} v + p_i.$$

Sphere

A sphere is characterized by the following constraint:

$$g(x, y, z) = x^2 + y^2 + z^2 - R^2. \tag{5.12}$$

The optimization problem in this case provides a value for p_i^* given by

$$p_i^* = \frac{p_i}{\|p_i\|^2}.$$

5.3.3 Obstacle avoidance

Another aspect that should be handled in a MAS is the collision avoidance between robots. Although the obstacle avoidance is not a coordination control law strictly speaking, we consider it here since it requires the measurement of relative distances between robots.

Thus, an additional control term is derived from repulsive potential fields as follows:

$$U_k = \begin{cases} \frac{1}{2}\eta(\frac{1}{d_k} - \frac{1}{d_0})^2 & \textit{if} \;\; d_k \leq d_0 \\ 0 & \textit{if} \;\; d_k > d_0 \end{cases} \tag{5.13}$$

where U_k denotes the repulsive potential of sensor k, d_0 is a threshold that activates the repulsive potential, d_k is the value of the distance between the sensor and the obstacle, and η is a constant that characterizes the field. Then, the resulting repulsive force F_k is defined by:

$$F_k = -\nabla U_k = \begin{cases} \eta(\frac{1}{d_k} - \frac{1}{d_0})\frac{1}{d_k^2}\frac{p_k - p_o}{d_k} & if\ \ d_k \leq d_0 \\ 0 & if\ \ d_k > d_0 \end{cases} \tag{5.14}$$

where $p_k - p_o$ is the relative position between the robot and the obstacle. Hence, the sum of all repulsive forces is $F = \sum\limits_k F_k$, which has an impact on the goal position according to the following expression:

$$u^{oa} = h \cdot v \cdot \frac{F}{\|F\|} \tag{5.15}$$

where u^{oa} is the deviation of the goal position signal received from the coordination level, h is the period of the controller, and v is a constant velocity.

5.3.4 Event-based sampling and control

The fundamentals of event-based control were introduced in Chapter 1. Instead of considering the periodic sampling, the sampling instances are defined recursively as

$$t_{k+1} = inf\{t : t > t_k, f(e(t), x(t)) > 0\} \tag{5.16}$$

where $f(e(t), x(t))$ is the trigger function. In this case, we define the error function $e_i(t) = p_i(t) - \hat{p}_i(t)$ for each robot i, where $\hat{p}_i$ is the last transmitted measurement. The bound for the error is defined as constant value c_i so that the robot will transmit its updated position to its neighbors whenever

$$f_i(e_i) = \|e_i(t)\| - c > 0. \tag{5.17}$$

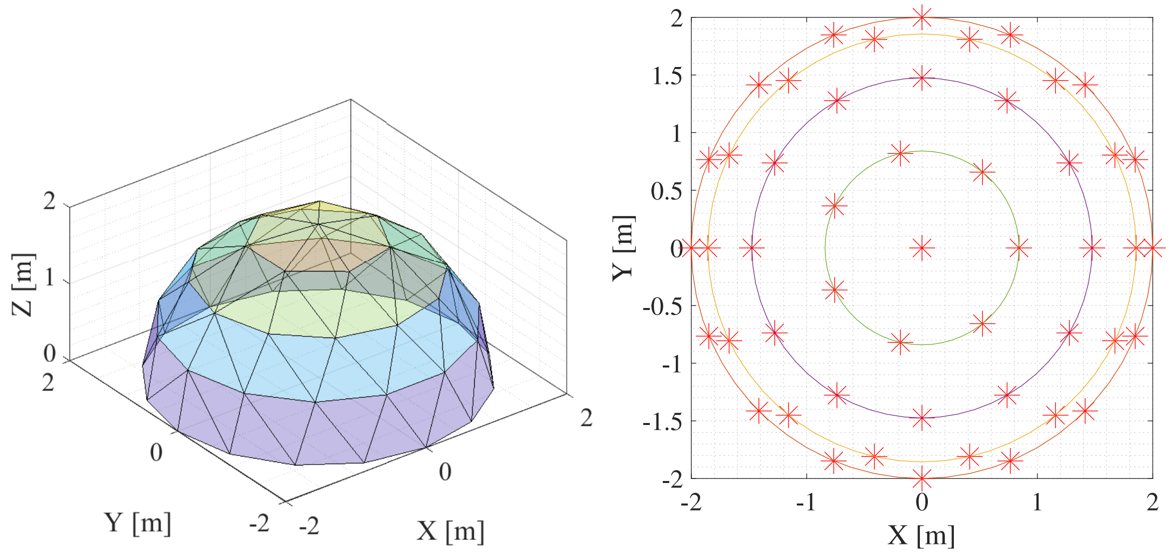
5.3.5 Mixed Reality and Digital Twins

The concept of MR has been handled throughout the manuscript. In the MARS experiments that will be presented in the following section, the joint use of virtual and real agents is a must for several reasons, including limited availability of the physical robots, constraints on the physical space for experimentation, or limitations imposed by the positioning systems.

As explained in Chapter 2, the ROS 2 architecture does not differentiate between virtual or real robots, and therefore, from the point of view of the controller all the positions of the robots are treated in the same way. However, from the point of view of the computational load, physical agents will demand more resources, since two nodes will be running in ROS 2: the DT driver and the physical robot driver itself.

5.4 Analysis of scalability

The main goal of this section is to explore the limit in the number of agents supported by two of the main simulators used in the Robotic Park environment and detailed in Section 2.8. For that purpose, we consider a formation control problem in which the complexity of the formation basically depends on the number of agents, N. Specifically, the desired formation is defined in such a way that the agents should be uniformly distributed over a semi-spherical virtual surface. The ground level of the dome ($z = 0$) is composed of mobile robots, and the rest of the robots are drones. The uniform distribution of points over a sphere is a classical mathematical problem that is difficult to solve analytically, and recent approximate solutions have been proposed [215]. Then, we assume that the target distances d^*_{ij} are given, and that the constraints given by the virtual surface are (5.12) but restricting z coordinate to be positive. Figure 5.2 shows an example of the desired 3D formation and the projection over the XY plane for $N = 50$ and a semi-sphere centered at (0,0,0) and a radius $R = 2$ m.



(a)

(b)

Figure 5.2. Example of formation with $N = 50$ agents and $R = 2$ m. **(a)** Desired 3D formation. **(b)** Projection over the XY plane.

5.4.1 Experiment description

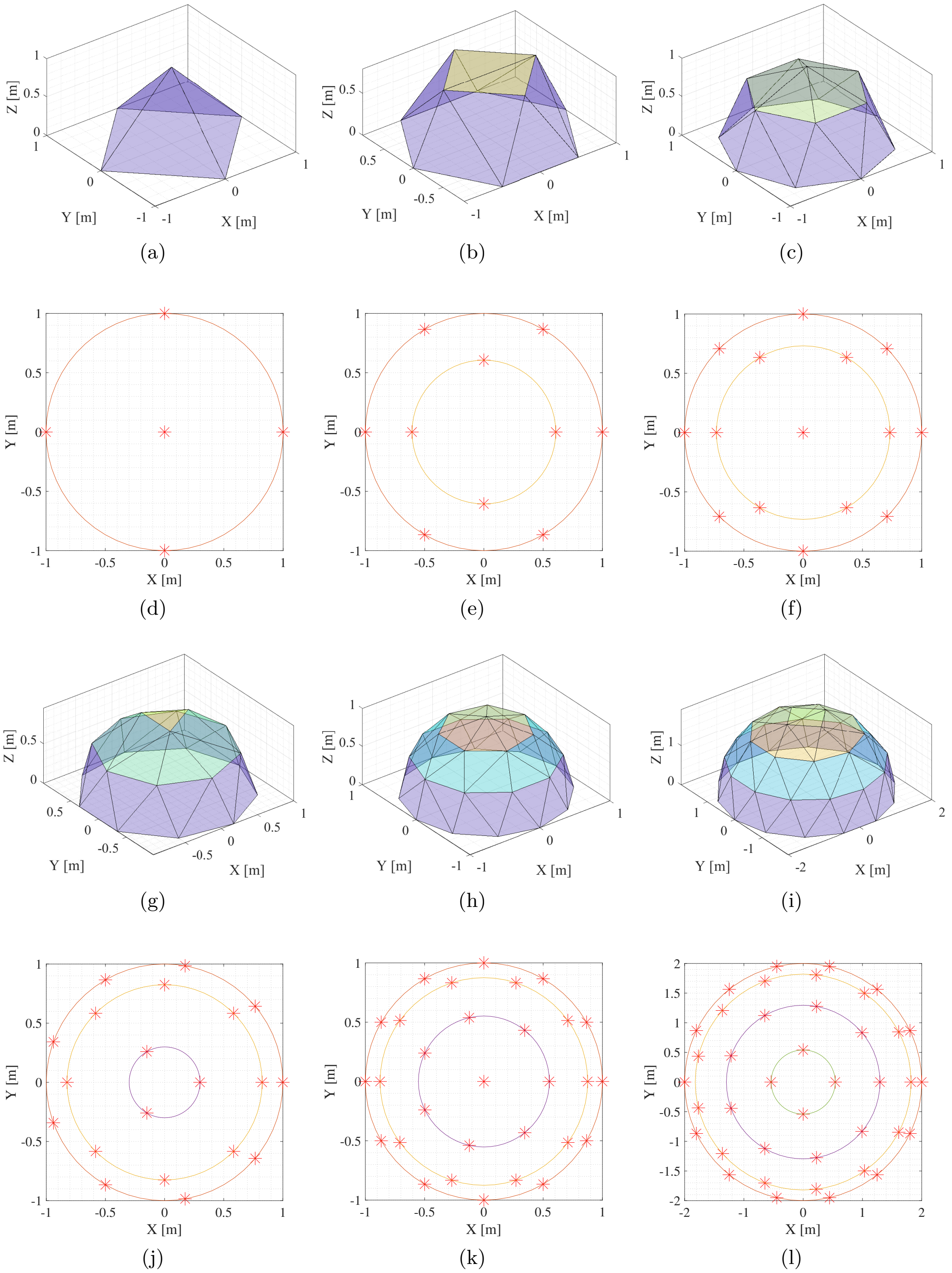
The proposed experiments to study the scalability are characterized by an increasing number of agents located along the surface of a hemisphere. A sweep is made from the simplest case with 5 agents to the limits that the simulation tools are able to support, 40 agents. In the hemisphere, the agents are placed at different levels. The level of height $z = 0$ consists of the Khepera IV ground robots moving in the XY plane. The rest of the agents are of the Crazyflie type and can move in three-dimensional space. Next, we briefly describe the developed experiments:

- The MARS of experiment A (see figures 5.3a and 5.3d) consists of a total of 5 agents, 4 of which are Khepera IV and 1 Crazyflie. In this case, all the robots are real and only their corresponding digital twins are running in the virtual environment.
- In experiment B (see figures 5.3b and 5.3e), the MRS is composed of 10 agents: 4 Crazyflies, and 6 Khepera. In this case, 4 real Crazyflies and 4 real Kheperas are used. In the virtual environment, 2 Kheperas run in addition to the virtual twins of the real robots.
- In experiment C (see figures 5.3c and 5.3f), the MRS is composed of 15 agents: 7 Crazyflies, and 8 Khepera. In this case, 5 real Crazyflies and 4 real Kheperas are used. The rest of the agents up to 15 are completely digital.
- The fourth experiment, D (see figures 5.3g and 5.3j), employs a total of 20 agents, 11 of which are Kheperas and 9 are Crazyflies. In this experience, 6 Crazyflies and 4 Kheperas are real. The rest of the agents up to 20 are completely digital.
- The acMARS in experiment E (see figures 5.3h and 5.3k) is composed by 30 agents. In this case, the distribution of agents is 18 Crazyflies and 12 Khepera.
- For the last experiment, F, depicted in figures 5.3i and 5.3l, the number of robots is 40 (26 Crazyflies and 14 Khepera).

From experiences A to C the proportion of real robots is more than 50% (reached in D). From D to F, the number of real robots is maintained at 6 Crazyflies and 4 Kheperas, increasing in this way the proportion of digital agents in these experiences progressively. This information is summarized in Table 5.1 and the final spatial distribution is shown in Figure 5.3. The video showing the real and virtual environment for the experiment E is available online: `https://youtu.be/4H3YZ-sr2mw`.

Table 5.1. Number of agents for each experiment. The DTs of real robots are included in the virtual robots column.

			Real Robots		Virtual Robots	
Experiment	**Figure**	**N**	**Crazyflie**	**Khepera**	**Crazyflie**	**Khepera**
A	5.3a, 5.3d	5	1	4	1	4
B	5.3b, 5.3e	10	4	4	4	6
C	5.3c, 5.3f	15	5	4	7	8
D	5.3g, 5.3j	20	6	4	11	9
E	5.3h, 5.3k	30	6	4	18	12
F	5.3i, 5.3l	40	6	4	26	14



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

Figure 5.3. MARS formation experiment representations. A: (**a**) 3D, (**b**) 2D agents distribution; B: (**c**) 3D, (**d**) 2D agents distribution; C: (**e**) 3D, (**f**) 2D agents distribution; D: (**g**) 3D, (**h**) 2D agents distribution; E: (**i**) 3D, (**j**) 2D agents distribution; F: (**k**) 3D, (**l**) 2D agents distribution.

To quantitatively evaluate the experiments, several parameters have been considered. On the one hand, we try to measure how computationally demanding are these experiments when the number of agents increases, and on the other hand, the impact on the performance of the system. These are the performance indices analyzed:

- Global CPU percentage. This value represents the current system-wide CPU utilization as a percentage.
- CPU percentage. It represents the individual process CPU utilization as a percentage. It can be > 100.0 in case of a process running multiple threads on different CPUs.
- Real-Time Factor (RTF). It shows a ratio of calculation time within a simulation (simulation time) to execution time (real-time).
- Integral Absolute Error (IAE). This index weights all errors equally over time. It gives global information about the agents.
- Integral of Time-weighted Absolute Error (ITAE). In systems that use step inputs, the initial error is always high. Consequently, to make a fair comparison between systems, errors maintained over time should have a greater weight than the initial errors. In this way, ITAE emphasizes reducing the error during the initial transient response and penalizes larger errors for longer.

5.4.2 Results

CPU usage, RTF, and formation error (IAE and ITAE) are used as criteria to compare and assess the simulation performance of Gazebo and Webots for all experiments described in Table 5.1. In each experiment, the number of agents increases and goes from 5 (experiment A) to 40 (experiment F).

CPU consumption

Results about CPU consumption are presented in Figure 5.4. Figure 5.4a shows the global CPU usage of the system. Note that this parameter increases with the number of robots for both simulators. However, Gazebo CPU consumption is higher than Webots in all experiments and reaches 100% for $N = 40$ (Experiment F). Indeed, for this later case, Gazebo was unable to run correctly as the timeout of the tool is exceeded when loading the robots at the beginning of the simulation. None of the simulators was able to carry out an experiment with 50 agents. Furthermore, from a performance perspective, a significant difference is observed between the two tools in terms of resource consumption. Figure 5.4b shows the specific CPU usage for the processes running in each simulation tool, since Gazebo is split into two processes: Gzclient, responsible for running the GUI, and Gzserver, responsible for the physics engine and sensors. The results show that in all cases recorded, the resource consumption of Gazebo (Gzclient+Gzserver) is more than double that of Webots. It is a clear indicator that Webots is a more efficient tool in terms of CPU usage management. Note that for experiments C, D and E the threshold of 100% is exceeded for Gazebo's processes. This just indicates that these processes have more than one thread running.

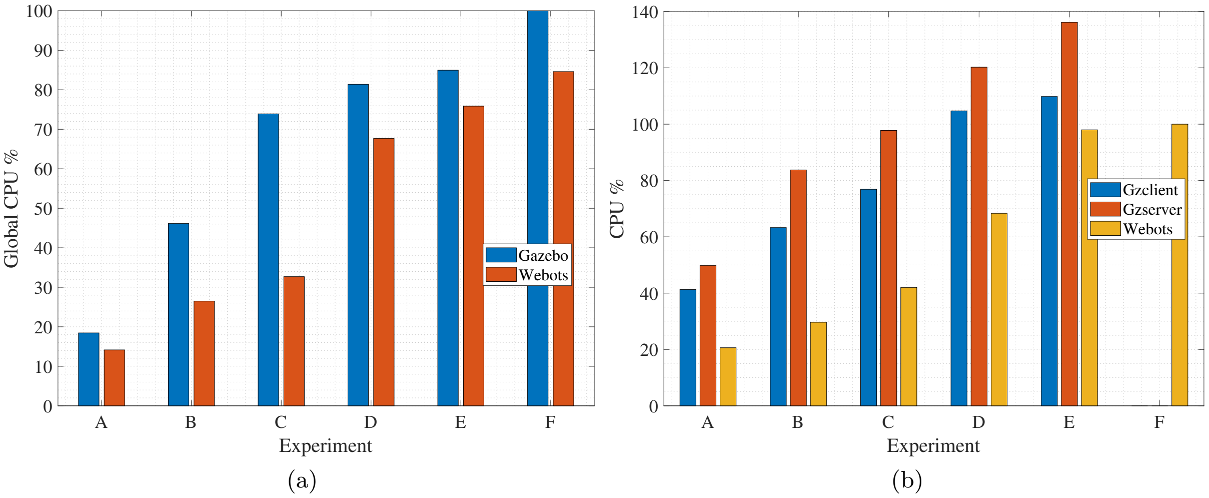


Figure 5.4. (**a**) General CPU % usage. (**b**) Simulation tools CPU % usage.

Real-Time Factor

The Real-Time Factor (RTF) is the ratio between the real execution time and the simulation time. This factor is easily accessible in the simulators. If it reaches the unit value, the simulation is running in real-time, and when RTF>1 it means that the process is running at an accelerated rate. Since we are performing experiments with real and virtual robots, even if all nodes are running in simulated time, this index should be as close to 1 as possible.

Table 5.2 shows the RTF obtained for each experience. Results are similar for both simulators when the number of agents is under 15. Gazebo achieves good performance in experiment C with 15 agents but when increasing the number up to 20 the RTF drops below 0.75. In this case, experiment D ($N = 20$), Webots still maintains a high performance. Experiment E increases the number of agents up to 30, and Webots performance is slightly lower but maintains a value above 0.8 while Gazebo falls below 0.5. Finally, in experiment F, Webots' RTF drops to a value of 0.56. There is a clear decrease in RTF when the number of robots increases. As Gazebo runs in two processes when the physics simulation in the successive experiments struggles, Gazebo decreases the RTF sooner, scaling worse than Webots to a large number of robots. Anyway, maintaining a high RTF allows running models in real time while using hardware-in-the-loop simulation to test controllers. In this sense, for experiences as presented in this paper, where batteries in real agents are a limiting factor, the highest RTF is the best. In fact, batteries for Crazyflies can only support 3-5 min flight time so allowing RTF under 0.75 is not possible.

Table 5.2. Real-Time Factor results in Gazebo and Webots.

Experiment	**Size**	**Gazebo**	**Webots**
A	5 agents	0.995	0.977
B	10 agents	0.967	0.977
C	15 agents	0.866	0.962
D	20 agents	0.716	0.941
E	30 agents	0.477	0.831
F	40 agents	–	0.563

System performance

Once metrics related to simulator computational efficiency have been examined, an analysis of convergence times to achieve the desired formation for both simulators is carried out. For this goal, first, the time evolution of the total error weighted by the number of agents for the different experiences is depicted in Figure 5.5. In a qualitative way, results are consistent for both simulators as similar behaviors are found among different tests in all cases. To obtain a quantitative analysis, we compute the IAE and ITAE weighted by the total number of agents and the experiment duration time. The results are shown in Table 5.3.

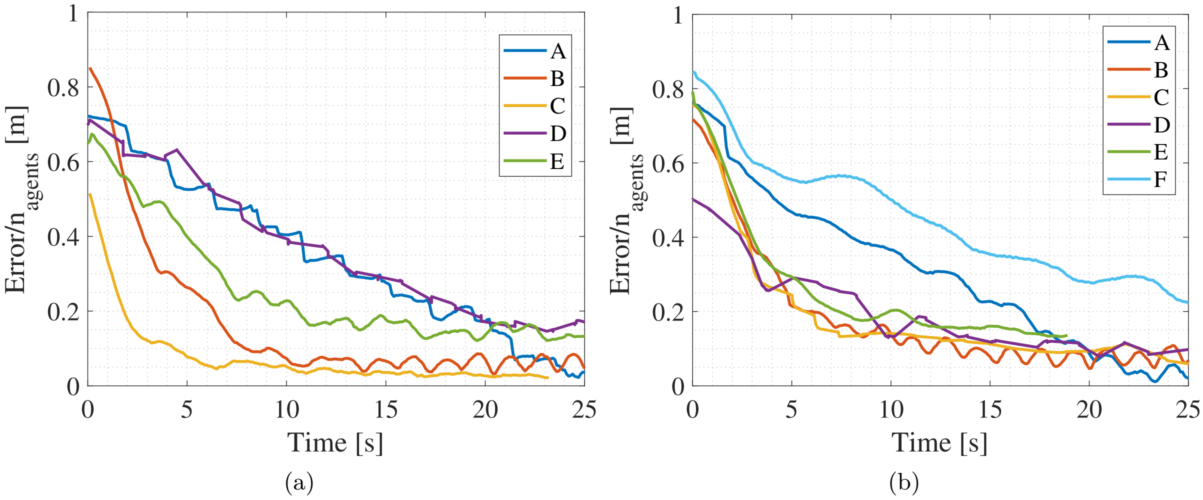


Figure 5.5. Instant total errors weighted by the total number of agents: (**a**) Gazebo. (**b**) Webots.

Data show that both tools scale well with the number of agents up to the limit supported by each simulator. Indeed, most of the values in Table 5.3 are in the same range. However, many

Table 5.3. IAE and ITAE in Gazebo and Webots experiences weighted by the total number of agents and experiments duration time.

	IAE (m/s)		ITAE (m)	
Experiment	**Gazebo**	**Webots**	**Gazebo**	**Webots**
A	0.4485	0.3879	6.8471	3.5558
B	0.3421	0.2481	4.2980	2.5427
C	0.4293	0.3062	5.0444	1.9985
D	0.4619	0.3182	5.0498	2.4613
E	0.5404	0.3108	5.7722	3.5415
F	–	0.6511	–	8.3456

factors might influence this result such as the number of digital and real agents, the number of connectivity links, the initial conditions, etc.

5.5 Architecture and protocol analysis

5.5.1 Experiment description

In this section, we try to determine the maximum sampling period that allows to maintain an adequate performance in the formation control problem. Moreover, we study a comparative with event-based sampling and we try to determine the maximum threshold of the trigger function with an equivalent performance to the periodic sampling. Finally, the number of transmitted messages for both cases is compared.

The MARS is composed in this case of five aerial robots (Crazyflie 2.1) and four ground robots (Khepera IV). The graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is undirected and the following set of nodes is defined:

$$\mathcal{V} = \{O, KH01, \dots, KH04, CF01, \dots, CF05\}, \tag{5.18}$$

where O represents the origin in $\mathbb{R}^3$, "KH" represents the Khepera IV robots, and "CF" represents the Crazyflie 2.1. The three-dimensional representation of a target formation is depicted in Figure 5.6. The desired formation is defined as follows:

- The robots of the same type connected through the graph should maintain an inter-distance $d^*_{ij} = 0.7071$ m.

- The distance between the ground and the aerial robots should be $d^*_{ij} = 0.9468$ m.

- Finally, the ground robots should maintain a distance to O of $d^*_{i0} = 0.5$ m, the aerial robots $CF02, \cdots, CF04$ a distance of $d^*_{i0} = 1.0\ m$ and the robot $CF01$ $d^*_{i0} = 1.36\ m$.

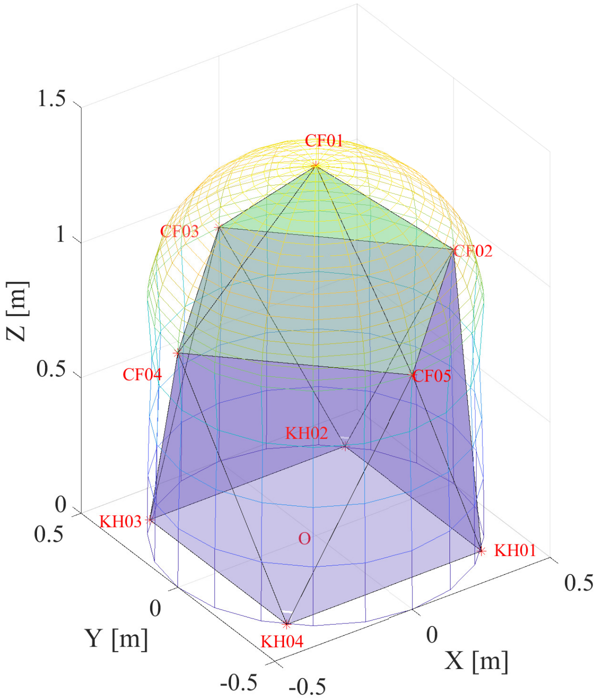


Figure 5.6. Three-dimensional representation of the target formation.

We analyze three different communication architectures for the coordination control:

1. **Centralized.** In this scenario, the coordination controllers are executed in a single ROS 2 node. This node subscribes to all the positions of the robots, executes the controllers for each one, and provides the tracking reference for the corresponding position controllers for each robot. These target positions are published in the network as topics and are read by each robot.

2. **Distributed in ROS 2.** In this case, each robot has its own coordination controller, which is executed in a different node. In our implementation, this node is the same used for the communication with the rest of the ROS 2 network, which results in a more efficient implementation.

3. **Distributed on board.** In this latter case, the coordination controller is executed in the micro-controller onboard each robot, and therefore ROS 2 is only used as a communication channel between robots.

For each of these scenarios, an interval for the sampling parameters has been evaluated for the periodic and event-based sampling. For the period sampling, the maximum frequency is 50 $[Hz]$, which is the limit value that the Phyton API admits for the internal positioning of the aerial robots. The minimum frequency is 0.5 $[Hz]$, which is the frequency at which the performance has been observed to degrade considerably. The rest of the studied frequency values are 20 $[Hz]$, 10 $[Hz]$, 5 $[Hz]$, and 1 $[Hz]$. For the evaluation of the event-based protocol, the threshold values that have been studied are 1 $[cm]$, 2 $[cm]$, 4 $[cm]$, 6 $[cm]$ and 8 $[cm]$. Overall, eleven protocols have been evaluated (including both time-triggered and event-triggered). Each case has been repeated seven times, and a statistical study has been performed for the results.

The indices used to provide a quantitative evaluation of the performance are the following:

- *Settling time, t_s.* The time required for the formation to reach and stay within a range of the 5% of the error for the target formation with respect to the initial conditions.
- *Average sampling period, T.* It represents the average time between transmitted messages for the robots along the experiment duration.
- *IAE* It integrates the absolute error of the overall system over time.
- *ITAE.* It integrates the absolute error of the overall system multiplied by the time over time.
- *% CPU percentage, η.* As in the previous section, the % of CPU (including all the kernels) usage has been registered as well as the % usage of the kernel in which the node responsible for the communication is executed.

5.5.2 Results

All the experiments have been performed following the same sequence of instructions, initial positions, and duration for a fair comparison. Initially, all the aerial robots receive a take-off order and they are commanded to the plane $z = 0.7$ $[m]$. After that, the formation control experiment starts and the next 18 $[s]$ are registered. Figure 5.7 shows the trajectories followed by the system (red lines) and the final formation (black dashed lines) for one of the experiments. Figure 5.8 shows the formation error over time for the most representative experiments. The error in each case has been computed as an average for the seven performed experiences.

A qualitative analysis of the results shows that the system converges to the desired formation and that the distributed onboard implementation (green line) has the fastest response. The behavior remains similar for the different sampling conditions. For the centralized and the distributed in ROS 2 cases, some differences can be appreciated, especially for the periodic sampling examples.

For a sampling frequency of 50 $[Hz]$ (see Figure 5.8a), the centralized control presents a faster response than the distributed implementation. This difference is not appreciated at 10 $[Hz]$, where almost identical responses are obtained (see Figure 5.8b). When the sampling frequency continues to decrease, the degradation of the performance is more significant in the centralized architecture. For the event-based sampling, we observe that independently of the threshold

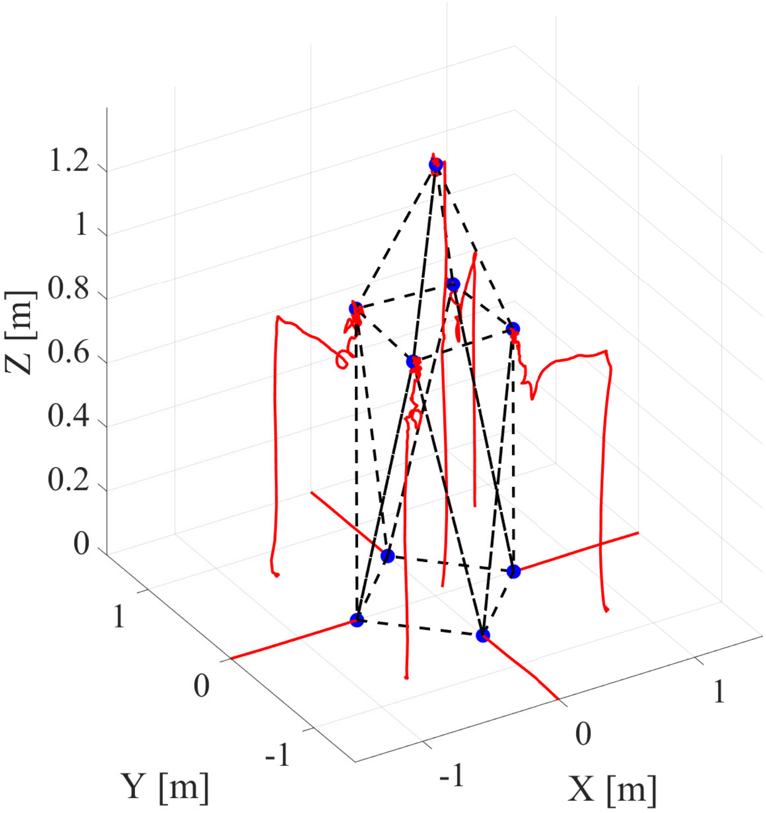


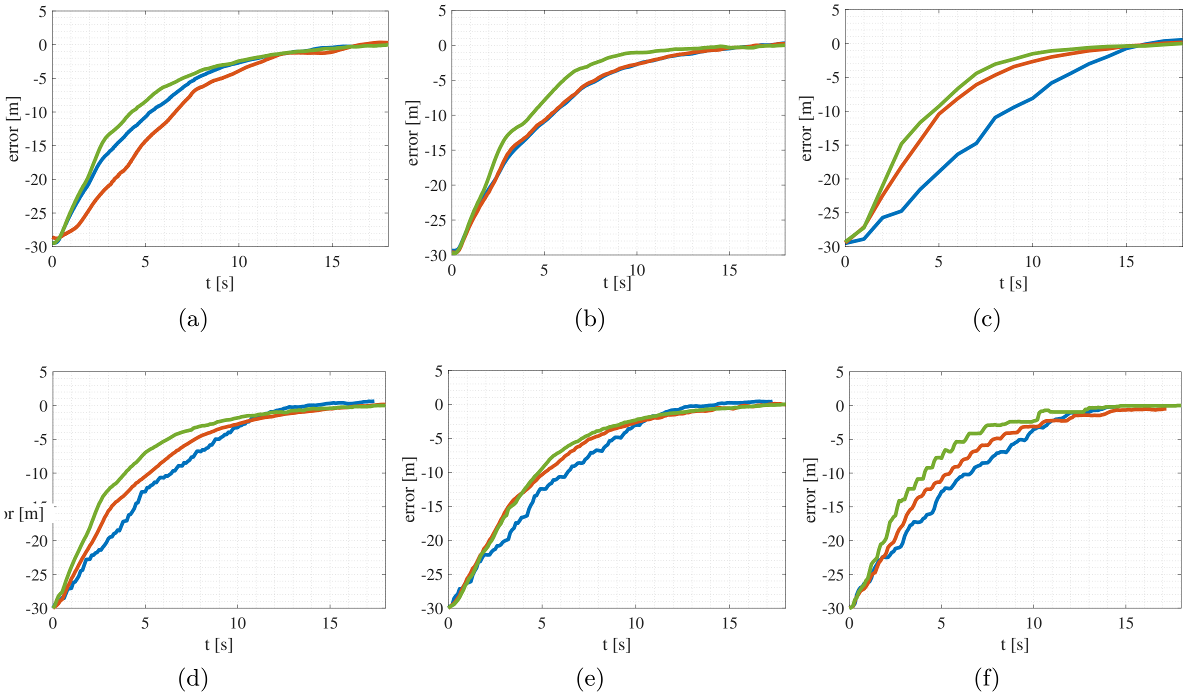
Figure 5.7. Trajectories followed by the robots for the centralized architecture and 10 $[Hz]$ sampling frequency experiment.

value, the distributed onboard implementation gets the best performance, and the centralized the slowest response.

If the settling time is analyzed, the average and standard deviation for the two sampling methods are shown in Figure 5.9. In the periodic case, the system performance worsens for sampling frequencies < 5 $[Hz]$, and there exists a significant dispersion of the results (higher standard deviation). For the event-based protocols, thresholds of 1 $[cm]$ and 2 $[cm]$ get similar values to the periodic case with frequencies higher than 5 $[Hz]$. When the threshold is enlarged, the slowing down of the response is similar in the three analyzed architectures.

To complete the quantitative analysis, Figure 5.10 shows the results for the IAE and ITAE indices. For both parameters, similar conclusions can be inferred than for the settling time. In general, the distributed onboard architecture offers the best behavior.

Regarding CPU usage, Figure 5.11 shows the % of usage including all the kernels in the system. As expected, higher values for the sampling frequencies demand a more intensive use of the CPU resources, and better results are obtained with the distributed onboard architecture. The centralized and distributed in ROS 2 obtain a similar behavior. For the event-based sampling,



(a) (b) (c)

(d) (e) (f)

Figure 5.8. A comparative for the time response. Blue: Centralized; Red: Distributed in ROS 2; Green: Distributed onboard. Periodic sampling frequency: **(a)** 50 $[Hz]$, **(b)** 10 $[Hz]$, **(c)** 1 $[Hz]$; Event-based sampling with threshold: **(d)** 1 $[cm]$, **(e)** 2 $[cm]$, **(f)** 4 $[cm]$

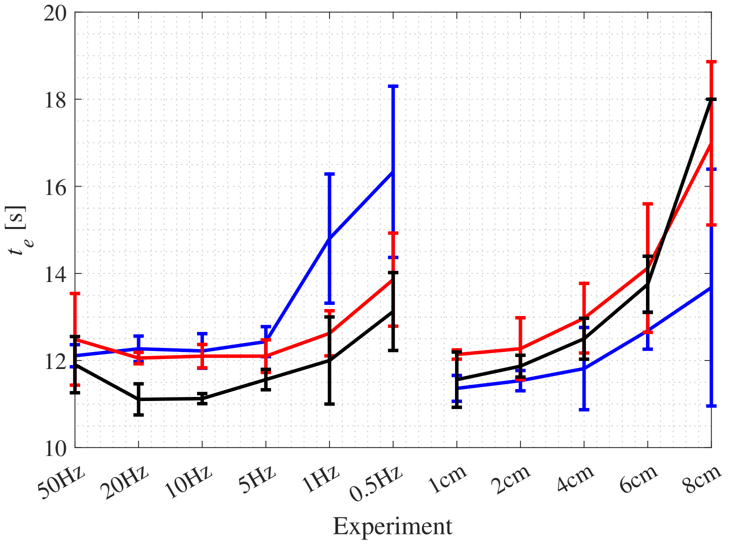
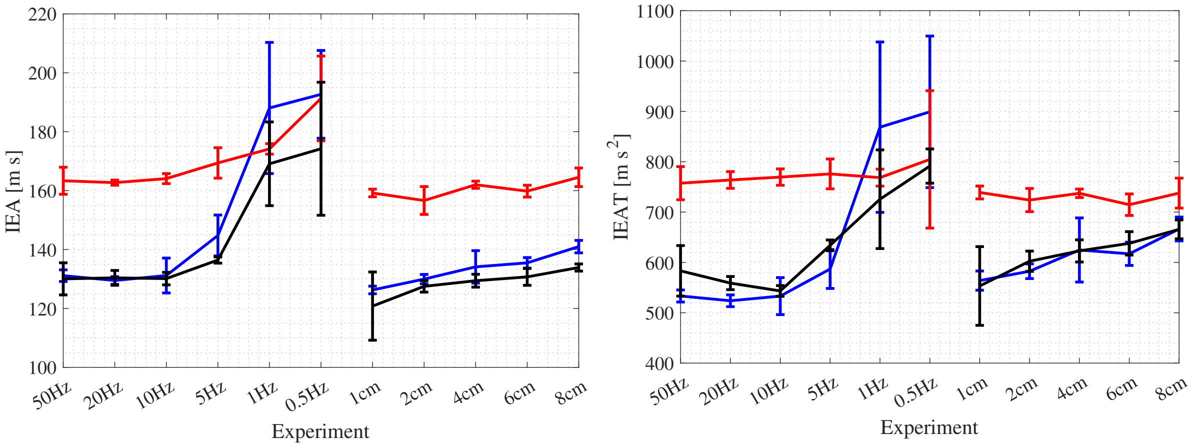


Figure 5.9. Settling time for the formation error. Blue: Centralized; Red: Distributed in ROS 2; Black: Distributed onboard.

the value of the threshold does not have much impact on this index, and the obtained values are similar to the 5 $[Hz]$ case.



(a) Integral Absolute Error.

(b) Integral Time-weighted Absolute Error.

Figure 5.10. Results for IAE and ITAE. Blue: Centralized; Red: Distributed in ROS 2; Black: Distributed onboard.

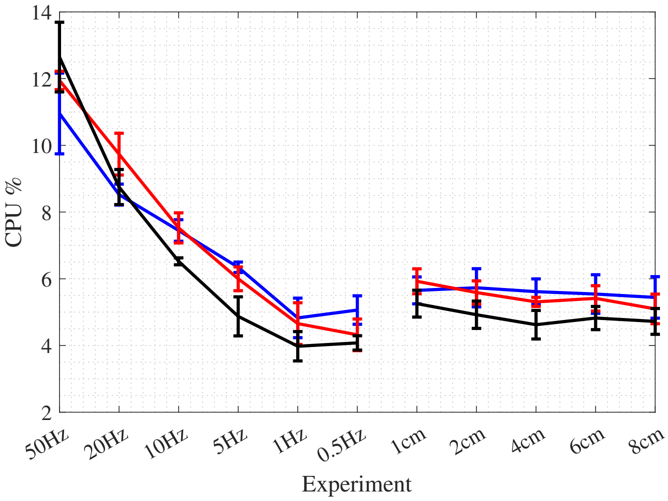
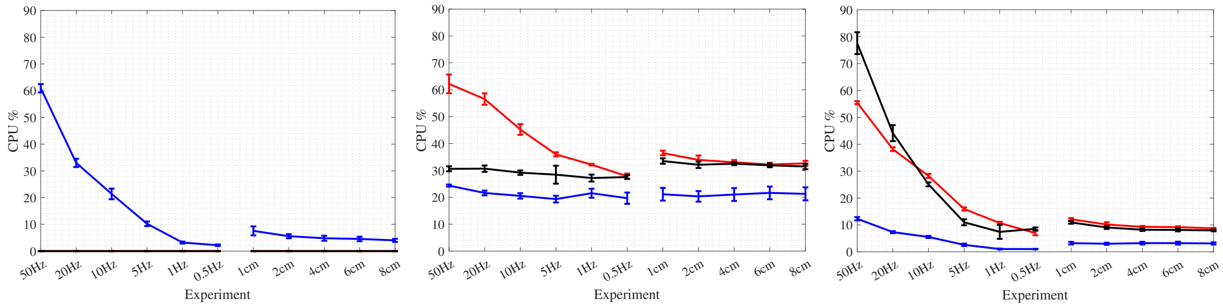


Figure 5.11. Percentage of used CPU.

If we analyze separately the consumed resources by the control node, in the centralized architecture, (see Figure 5.12a), there exists a correlation between CPU consumption and the frequency of the controller, whereas there are no big differences in the event-triggered case.

The % of CPU usage by the nodes handling the communication with the aerial robots and the ground robots are shown in Figures 5.12b and 5.12c, respectively. In general terms, the higher the frequency, the more intensive the use of the CPU. However, as in the previous case, the threshold does not have much effect on this parameter. For the distributed architectures (red and black lines), the fact that agents' poses need to be broadcasted to the neighbors causes a more intensive use of the communication nodes and channel.



(a) Centralized controller.

(b) Crazyflies 2.1.

(c) Kheperas IV.

Figure 5.12. CPU usage percentage for the nodes in charge of the centralized controller, the communication with the Crazyflies 2.1, and the Khepera IV. Blue: Centralized; Red: Distributed in ROS 2; Black: Distributed onboard.

Finally, the average sampling period T_s for the different scenarios is shown in Figure 5.13. This parameter has only relevance in the event-triggered case, but we show the results for the periodic sampling to establish a comparison. Separate values of T_s are computed for the aerial and the ground robots. The Crazyflies (dashed lines) obtain similar results independently of the architecture. However, the Khepera robots (solid lines) present different behaviors, especially for small values of the threshold. In this case, the centralized architecture offers larger average sampling periods and small values for the standard deviation. Moreover, in all cases, the distributed onboard architecture outperforms the distributed in ROS 2 implementation. However, the standard deviation increases considerably for thresholds over the 4 $[cm]$.

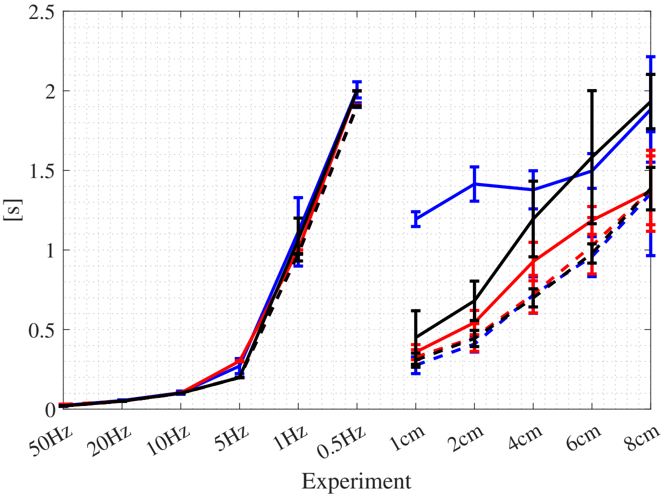


Figure 5.13. Average sampling period. "−": Khepera IV; "−−": Crazyflie 2.1; Blue: Centralized; Red: Distributed in ROS 2; Black: Distributed on board.

5.6 Conclusions

In this chapter, the formation control problem has been studied, since it is one of the fundamentals of cooperative tasks in MARS. First, an approach for the extension of rigid formations to virtual spatial constraints is provided. Then, a set of experiments has been presented to evaluate the performance of Robotic Park for MARS experiences. Aspects such as the scalability, the control architecture, and the communication protocol have been evaluated. The results show that the platform obtains good results up to $N = 40$ robots when real and virtual agents are combined. Regarding the architecture, the distributed onboard implementation obtains the best trade-off between system performance and resource usage, though it requires more intensive use of communication resources than a centralized implementation. This communication load can be alleviated by means of event-based protocols. The results show that an adequate choice of the event threshold can provide similar performances than the periodic transmission of messages, but reducing considerably the average sampling period.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis has addressed the development from scratch of a multi-agent robotic platform, Robotic Park. Consequently, a set of challenges intrinsic to the deployment of the physical systems have been tackled.

Specifically, Chapter 2 focuses on implementing Robotic Park, and then all the hardware and software components that make it up have been detailed. Based on a middleware layer developed in ROS 2, the designed architecture allows the support of real, virtual, and hybrid experiences, offers flexibility to define the control and communication architecture, and eases the replacement and addition of elements without reconfiguring the rest of the system. Moreover, other relevant strengths of the platform are the heterogeneity and interoperability of agents.

As a prerequisite to perform MARS experiences, a deep knowledge of the agents' modeling and control is essential. In RP, ground and aerial robots for indoor environments have been used. All aspects regarding the modeling, and the control architecture up to the positioning controller have been addressed in Chapters 3 and 4, respectively. Whereas the model for the differential robots is relatively simple, the high-order model for quadcopters makes suitable a model identification of the closed-loop system. This task was accomplished and resulted in a second-order model with no zeros in the system. Additionally, the study of these robots concluded by implementing event-based control techniques in their position control. Results have demonstrated a 50% reduction of the transmissions in velocity control and a 90% reduction in position control. This represents a key factor in freeing up communication and computational resources, which also increases the battery life of the devices.

Once the individual control of the agents was completed and the experimental platform was operational (both physically and virtually), MARS experiences were developed from different approaches in Chapter 5. Regarding the control architecture, this represents an upper-level respect to the positioning controller, and then, cooperative strategies can be designed without interfering with the other control layers. In this sense, three main contributions of this dissertation regarding MARS control can be highlighted.

Firstly, the Robotic Park architecture allows different levels of controller decentralization, and hence, three different configurations have been evaluated. The results show that, in general, the implementation of distributed controllers onboard robots is preferred. Secondly, it has been shown that a proper tuning of event-triggering rules applied to inter-agent communications improves transmission efficiency and computational resource consumption. This not only enhances the individual performance of each agent but also allows scaling of the system size. Formation control experiments have been carried out with up to 30 agents without performance degradation.

And finally, a line of great interest in current research, present throughout the development of this thesis, is the work combining virtual and physical agents. This is known as Mixed Reality experiences and enables experiments that cannot be performed exclusively in the physical environment. Examples include equipping simple robots with complex sensors such as lidars without additional weight or energy consumption but at the expense of highly accurate virtual environments. If the virtual element represents an active physical agent and there is bidirectional communication in real-time, the virtual element becomes a Digital Twin (DT). This thesis has developed drivers that enable communication between DT of RP agents. These twins have been used to increase the sensing capabilities of real robots and as a safety backup in case of critical failures of real robots.

6.2 Future works

This work can be extended in several directions. Some suggestions are listed below.

- In the Robotic Park hardware layer, the incorporation of new robots with different locomotion systems (such as omnidirectional, Ackermann, with legs, or spherical) would increase the range of experiences that could be carried out on the platform. These new additions imply new challenges in the designs of the control layer.

- The low-cost vision-based positioning system that is already operational has the potential to detect and track objects within the platform environment. However, the first experiences show that the error obtained is not negligible if it is used as the only positioning system. In this way, some improvements are necessary in the resolution of the optical sensors, the optimization of the field of view, and the fine-tuning of the algorithm for pairing detections between cameras.

- At Robotic Park software level, it is important to provide more autonomous navigation to the robots by making use of SLAM techniques under limited computation capabilities. Additionally, studying some security aspects such as confidentiality and integrity of the experimental infrastructure when, for instance, the platform is operated remotely, will be considered.

- The upper levels of the control architecture in Robotic Park have not been explored in the experiences. Then, more complex tasks that include perception, planning, localization, and much more will be accomplished. In this sense, it is interesting to address dynamic task planning. To this end, the modularity of the platform allows easy incorporation of new software tools such as Nav2 [216] or Mobile Robot Programming Toolkit (MRPT) [217].
- An interesting line of research is the application of intelligent control techniques, such as fuzzy logic, machine learning, and deep learning. In the formation control of heterogeneous multi-robot systems, these techniques are appropriated for intelligent coordination and adaptation to dynamic and changing environments.

Bibliography

[1] “IEEE Standard Ontologies for Robotics and Automation,” *IEEE Std 1872-2015*, pp. 1–60, 2015.

[2] M. Russo and M. Ceccarelli, “A survey on mechanical solutions for hybrid mobile robots,” *Robotics*, vol. 9, no. 2, p. 32, 2020.

[3] F. Kadar and M. Tătar, “A review on mobile robots with multimodal locomotion,” in *IFToMM International Symposium on Science of Mechanisms and Machines (SYROM)*, (Cham, Switzerland), pp. 337–349, Springer, 2022.

[4] R. Shrestha, R. Bajracharya, and S. Kim, “6g enabled unmanned aerial vehicle traffic management: A perspective,” *IEEE Access*, vol. 9, pp. 91119–91136, 2021.

[5] E. A. Lee, “Cyber physical systems: Design challenges,” in *2008 11th IEEE international symposium on object and component-oriented real-time distributed computing (ISORC)*, (Orlando, FL, USA), pp. 363–369, IEEE, 2008.

[6] D. Serpanos, “The cyber-physical systems revolution,” *Computer*, vol. 51, no. 3, pp. 70–73, 2018.

[7] J. Barbosa, P. Leitão, D. Trentesaux, A. W. Colombo, and S. Karnouskos, “Cross benefits from cyber-physical systems and intelligent products for future smart industries,” in *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*, (Poitiers, France), pp. 504–509, IEEE, 2016.

[8] M. García-Valls, A. Dubey, and V. Botti, “Introducing the new paradigm of social dispersed computing: Applications, technologies and challenges,” *Journal of Systems Architecture*, vol. 91, pp. 83–102, 2018.

[9] S. F. Ochoa, G. Fortino, and G. Di Fatta, “Cyber-physical systems, internet of things and big data,” *Future Generation Computer Systems*, vol. 75, pp. 82–84, 2017.

[10] W. Duo, M. Zhou, and A. Abusorrah, “A survey of cyber attacks on cyber physical systems: Recent advances and challenges,” *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. 5, pp. 784–800, 2022.

[11] A. Gatouillat, Y. Badr, B. Massot, and E. Sejdić, “Internet of medical things: A review of recent contributions dealing with cyber-physical systems in medicine,” *IEEE internet of things journal*, vol. 5, no. 5, pp. 3810–3822, 2018.

[12] Y. Zhao, Z. Liu, and W. S. Wong, "Resilient platoon control of vehicular cyber physical systems under dos attacks and multiple disturbances," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 10945–10956, 2021.

[13] M. K. Hasan, A. A. Habib, Z. Shukur, F. Ibrahim, S. Islam, and M. A. Razzaque, "Review on cyber-physical and cyber-security system in smart grid: Standards, protocols, constraints, and recommendations," *Journal of Network and Computer Applications*, vol. 209, p. 103540, 2023.

[14] A. Puliafito, G. Tricomi, A. Zafeiropoulos, and S. Papavassiliou, "Smart cities of the future as cyber physical systems: Challenges and enabling technologies," *Sensors*, vol. 21, no. 10, p. 3349, 2021.

[15] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," *IEEE Access*, vol. 6, pp. 28573–28593, 2018.

[16] R. Owoputi and S. Ray, "Security of multi-agent cyber-physical systems: A survey," *IEEE Access*, vol. 10, pp. 121465–121479, 2022.

[17] H. Rezaee and F. Abdollahi, "Average consensus over high-order multiagent systems," *IEEE Transactions on Automatic Control*, vol. 60, no. 11, pp. 3047–3052, 2015.

[18] Y. Cao, W. Yu, W. Ren, and G. Chen, "An overview of recent progress in the study of distributed multi-agent coordination," *IEEE Transactions on Industrial informatics*, vol. 9, no. 1, pp. 427–438, 2012.

[19] A. Zidan, M. Khairalla, A. M. Abdrabou, T. Khalifa, K. Shaban, A. Abdrabou, R. El Shatshat, and A. M. Gaouda, "Fault detection, isolation, and service restoration in distribution systems: State-of-the-art and future trends," *IEEE Transactions on Smart Grid*, vol. 8, no. 5, pp. 2170–2185, 2016.

[20] Y. Hedin and E. Moradian, "Security in multi-agent systems," *Procedia Computer Science*, vol. 60, pp. 1604–1612, 2015.

[21] A. Gautam and S. Mohan, "A review of research in multi-robot systems," in *2012 IEEE 7th international conference on industrial and information systems (ICIIS)*, (Chennai, India), pp. 1–5, IEEE, 2012.

[22] A. Madridano, A. Al-Kaff, D. Martín, and A. De La Escalera, "Trajectory planning for multi-robot systems: Methods and applications," *Expert Systems with Applications*, vol. 173, p. 114660, 2021.

[23] Y. Tan and Z.-y. Zheng, "Research advance in swarm robotics," *Defence Technology*, vol. 9, no. 1, pp. 18–39, 2013.

[24] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intelligence*, vol. 7, pp. 1–41, 2013.

[25] P. Iñigo-Blasco, F. Diaz-del Rio, M. C. Romero-Ternero, D. Cagigas-Muñiz, and S. Vicente-Diaz, "Robotics software frameworks for multi-agent robotic systems development," *Robotics and Autonomous Systems*, vol. 60, no. 6, pp. 803–821, 2012.

[26] H. L. Kwa, J. Leong Kit, and R. Bouffanais, “Balancing collective exploration and exploitation in multi-agent and multi-robot systems: A review,” *Frontiers in Robotics and AI*, vol. 8, p. 771520, 2022.

[27] C. García, P. F. Cárdenas, L. J. Puglisi, and R. Saltaren, “Design and modeling of the multi-agent robotic system: Smart,” *Robotics and Autonomous Systems*, vol. 60, no. 2, pp. 143–153, 2012.

[28] J. Xie and C.-C. Liu, “Multi-agent systems and their applications,” *Journal of International Council on Electrical Engineering*, vol. 7, no. 1, pp. 188–197, 2017.

[29] A. A. Mutlag, M. K. Abd Ghani, M. A. Mohammed, A. Lakhan, O. Mohd, K. H. Abdulkareem, and B. Garcia-Zapirain, “Multi-agent systems in fog–cloud computing for critical healthcare task management model (chtm) used for ecg monitoring,” *Sensors*, vol. 21, no. 20, p. 6923, 2021.

[30] F. Yang, Y. Qiao, S. Wang, C. Huang, and X. Wang, “Blockchain and multi-agent system for meme discovery and prediction in social network,” *Knowledge-Based Systems*, vol. 229, p. 107368, 2021.

[31] N. Singh, I. Elamvazuthi, P. Nallagownden, G. Ramasamy, and A. Jangra, “Routing based multi-agent system for network reliability in the smart microgrid,” *Sensors*, vol. 20, no. 10, p. 2992, 2020.

[32] S. Gao, R. Song, and Y. Li, “Coordinated control of multiple euler–lagrange systems for escorting missions with obstacle avoidance,” *Applied Sciences*, vol. 9, no. 19, p. 4144, 2019.

[33] T. Krajník, M. Nitsche, J. Faigl, P. Vaněk, M. Saska, L. Přeučil, T. Duckett, and M. Mejail, “A practical multirobot localization system,” *Journal of Intelligent & Robotic Systems*, vol. 76, pp. 539–562, 2014.

[34] J. Hu and A. Lanzon, “An innovative tri-rotor drone and associated distributed aerial drone swarm control,” *Robotics and Autonomous Systems*, vol. 103, pp. 162–174, 2018.

[35] J. Faigl, T. Krajník, V. Vonásek, and L. Přeučil, “On localization uncertainty in an autonomous inspection,” in *2012 IEEE International Conference on Robotics and Automation*, (Saint Paul, MN, USA), pp. 1119–1124, IEEE, 2012.

[36] C. Lesire, G. Infantes, T. Gateau, and M. Barbier, “A distributed architecture for supervision of autonomous multi-robot missions: Application to air-sea scenarios,” *Autonomous Robots*, vol. 40, pp. 1343–1362, 2016.

[37] J. J. Roldán, E. Peña-Tapia, P. Garcia-Aunon, J. Del Cerro, and A. Barrientos, “Bringing adaptive and immersive interfaces to real-world multi-robot scenarios: Application to surveillance and intervention in infrastructures,” *IEEE Access*, vol. 7, pp. 86319–86335, 2019.

[38] T. Nestmeyer, P. Robuffo Giordano, H. H. Bülthoff, and A. Franchi, “Decentralized simultaneous multi-target exploration using a connected network of multiple robots,” *Autonomous robots*, vol. 41, pp. 989–1011, 2017.

[39] A. Ribeiro and J. Conesa-Muñoz, *Multi-robot systems for precision agriculture*, pp. 151–175. Cham, Switzerland: Springer, 2021.

[40] A. Khamis, A. Hussein, and A. Elmogy, "Multi-robot task allocation: A review of the state-of-the-art," *Cooperative robots and sensor networks 2015*, pp. 31–51, 2015.

[41] T. Gao, H. Emadi, H. Saha, J. Zhang, A. Lofquist, A. Singh, B. Ganapathysubramanian, S. Sarkar, A. K. Singh, and S. Bhattacharya, "A novel multirobot system for plant phenotyping," *Robotics*, vol. 7, no. 4, p. 61, 2018.

[42] M. Guinaldo, J. Sánchez, and S. Dormido, "Control en red basado en eventos: de lo centralizado a lo distribuido," *Revista Iberoamericana de Automática e Informática industrial*, vol. 14, no. 1, pp. 16–30, 2017.

[43] H.-S. Ahn, *Formation control.* Cham, Switzerland: Springer, 2020.

[44] K.-K. Oh, M.-C. Park, and H.-S. Ahn, "A survey of multi-agent formation control," *Automatica*, vol. 53, pp. 424–440, 2015.

[45] Y. Q. Chen and Z. Wang, "Formation control: a review and a new consideration," in *2005 IEEE/RSJ International conference on intelligent robots and systems*, (Edmonton, AB, Canada), pp. 3181–3186, IEEE, 2005.

[46] H. Oh, A. R. Shirazi, C. Sun, and Y. Jin, "Bio-inspired self-organising multi-robot pattern formation: A review," *Robotics and Autonomous Systems*, vol. 91, pp. 83–100, 2017.

[47] C. Vrohidis, P. Vlantis, C. P. Bechlioulis, and K. J. Kyriakopoulos, "Reconfigurable multi-robot coordination with guaranteed convergence in obstacle cluttered environments under local communication," *Autonomous Robots*, vol. 42, no. 4, pp. 853–873, 2018.

[48] C. Robin and S. Lacroix, "Multi-robot target detection and tracking: taxonomy and survey," *Autonomous Robots*, vol. 40, pp. 729–760, 2016.

[49] M. Khaledyan, T. Liu, V. Fernandez-Kim, and M. de Queiroz, "Flocking and target interception control for formations of nonholonomic kinematic agents," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 4, pp. 1603–1610, 2019.

[50] W. Guanghua, L. Deyi, G. Wenyan, and J. Peng, "Study on formation control of multi-robot systems," in *2013 Third international conference on intelligent system design and engineering applications*, (Hong Kong, China), pp. 1335–1339, IEEE, 2013.

[51] D. Sun, A. Kleiner, and B. Nebel, "Behavior-based multi-robot collision avoidance," in *2014 IEEE international conference on robotics and automation (ICRA)*, (Hong Kong, China), pp. 1668–1673, IEEE, 2014.

[52] F. J. Mendiburu, M. R. Morais, and A. M. Lima, "Behavior coordination in multi-robot systems," in *2016 IEEE International Conference on Automatica (ICA-ACCA)*, (Curico, Chile), pp. 1–7, IEEE, 2016.

[53] G. A. Di Caro and A. W. Z. Yousaf, "Multi-robot informative path planning using a leader-follower architecture," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, (Xi'an, China), pp. 10045–10051, IEEE, 2021.

[54] D. Qian and Y. Xi, “Leader–follower formation maneuvers for multi-robot systems via derivative and integral terminal sliding mode,” *Applied Sciences*, vol. 8, no. 7, p. 1045, 2018.

[55] H. Xiao and C. P. Chen, “Leader-follower consensus multi-robot formation control using neurodynamic-optimization-based nonlinear model predictive control,” *IEEE access*, vol. 7, pp. 43581–43590, 2019.

[56] D. Roy, A. Chowdhury, M. Maitra, and S. Bhattacharya, “Multi-robot virtual structure switching and formation changing strategy in an unknown occluded environment,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Madrid, Spain), pp. 4854–4861, IEEE, 2018.

[57] A. Benzerrouk, L. Adouane, and P. Martinet, “Stable navigation in formation for a multi-robot system based on a constrained virtual structure,” *Robotics and Autonomous Systems*, vol. 62, no. 12, pp. 1806–1815, 2014.

[58] L. Arcos, C. Calala, D. Maldonado, and P. J. Cruz, “ROS based Experimental Testbed for Multi-Robot Formation Control,” in *2020 IEEE ANDESCON*, pp. 1–6, IEEE, 2020.

[59] D. J. Bennet and C. R. McInnes, “Distributed control of multi-robot systems using bifurcating potential fields,” *Robotics and Autonomous Systems*, vol. 58, no. 3, pp. 256–264, 2010.

[60] A. Renzaglia and A. Martinelli, “Potential field based approach for coordinate exploration with a multi-robot team,” in *2010 IEEE Safety Security and Rescue Robotics*, (Bremen, Germany), pp. 1–6, IEEE, 2010.

[61] R. Falconi, L. Sabattini, C. Secchi, C. Fantuzzi, and C. Melchiorri, “Edge-weighted consensus-based formation control strategy with collision avoidance,” *Robotica*, vol. 33, no. 2, pp. 332–347, 2015.

[62] W. Heemels, K. Johansson, and P. Tabuada, “An introduction to event-triggered and self-triggered control,” in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, (Maui, HI, USA), pp. 3270–3285, IEEE, 2012.

[63] E. Aranda-Escolastico, M. Guinaldo, R. Heradio, J. Chacon, H. Vargas, J. Sánchez, and S. Dormido, “Event-based control: A bibliometric analysis of twenty years of research,” *IEEE Access*, vol. 8, pp. 47188–47208, 2020.

[64] M. Sohani, “Event based pid control,” Master’s thesis, KTH, Automatic Control, 2009.

[65] W. Heemels, J. Sandee, and P. Van Den Bosch, “Analysis of event-driven controllers for linear systems,” *International Journal of Control*, vol. 81, no. 4, pp. 571–590, 2008.

[66] P. Tabuada, “Event-triggered real-time scheduling of stabilizing control tasks,” *IEEE Transactions on Automatic Control*, vol. 52, no. 9, pp. 1680–1685, 2007.

[67] C. De Persis, R. Sailer, and F. Wirth, “Parsimonious event-triggered distributed control: A zeno free approach,” *Automatica*, vol. 49, no. 7, pp. 2116–2124, 2013.

[68] M. Donkers and W. Heemels, “Output-based event-triggered control with guaranteed $\updownarrow_{\infty}$-gain and improved and decentralized event-triggering,” *IEEE Transactions on Automatic Control*, vol. 57, no. 6, pp. 1362–1376, 2012.

[69] M. Guinaldo, D. Dimarogonas, K. Johansson, J. Sanchez, and S. Dormido, “Distributed event-based control strategies for interconnected linear systems,” *IET Control Theory and Applications*, vol. 7, no. 6, pp. 877–886, 2013.

[70] C. Nowzari and J. Cortés, “Zeno-free, distributed event-triggered communication and control for multi-agent average consensus,” in *2014 American Control Conference*, (Portland, OR, USA), pp. 2148–2153, IEEE, 2014.

[71] S. Dormido, J. Sánchez, and E. Kofman, “Muestreo, control y comunicación basados en eventos,” *Revista Iberoamericana de Automática e Informática Industrial RIAI*, vol. 5, no. 1, pp. 5–26, 2008.

[72] E. Garcia and P. J. Antsaklis, “Model-based event-triggered control with time-varying network delays,” in *2011 50th IEEE Conference on Decision and Control and European Control Conference*, (Orlando, FL, USA), pp. 1650–1655, IEEE, 2011.

[73] D. Liuzza, D. V. Dimarogonas, M. Di Bernardo, and K. H. Johansson, “Distributed model based event-triggered control for synchronization of multi-agent systems,” *Automatica*, vol. 73, pp. 1–7, 2016.

[74] L. Ding, Q.-L. Han, X. Ge, and X.-M. Zhang, “An overview of recent advances in event-triggered consensus of multiagent systems,” *IEEE Transactions on Cybernetics*, vol. 48, no. 4, pp. 1110–1123, 2018.

[75] R. Socas, S. Dormido, and R. Dormido, “Event-based controller for noisy environments,” in *2014 Second World Conference on Complex Systems (WCCS)*, (Agadir, Morocco), pp. 280–285, IEEE, 2014.

[76] Y. Sahni, J. Cao, and S. Jiang, “Middleware for multi-robot systems,” *Mission-Oriented Sensor Networks and Systems: Art and Science: Volume 2: Advances*, pp. 633–673, 2019.

[77] J. Palanca, A. Terrasa, V. Julian, and C. Carrascosa, “Spade 3: Supporting the new generation of multi-agent systems,” *IEEE Access*, vol. 8, pp. 182537–182549, 2020.

[78] F. Bergenti, G. Caire, S. Monica, and A. Poggi, “The first twenty years of agent-based software development with jade,” *Autonomous Agents and Multi-Agent Systems*, vol. 34, no. 36, pp. 1–19, 2020.

[79] R. T. Vaughan and B. P. Gerkey, *Reusable Robot Software and the Player/Stage Project*, pp. 267–289. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[80] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, “Robot operating system 2: Design, architecture, and uses in the wild,” *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[81] S. Macenski, A. Soragna, M. Carroll, and Z. Ge, “Impact of ros 2 node composition in robotic systems,” *IEEE Robotics and Automation Letters*, vol. 8, no. 7, pp. 3996–4003, 2023.

[82] S. Macenski, T. Moore, D. V. Lu, A. Merzlyakov, and M. Ferguson, "From the desks of ros maintainers: A survey of modern & capable mobile robotics algorithms in the robot operating system 2," *Robotics and Autonomous Systems*, p. 104493, 2023.

[83] P. Bellavista, A. Corradi, L. Foschini, and A. Pernafini, "Data distribution service (dds): A performance comparison of opensplice and rti implementations," in *2013 IEEE symposium on computers and communications (ISCC)*, (Split, Croatia), pp. 000377–000383, IEEE, 2013.

[84] K. Belsare, A. C. Rodriguez, P. G. Sánchez, J. Hierro, T. Kołcon, R. Lange, I. Lütkebohle, A. Malki, J. M. Losa, F. Melendez, M. M. Rodriguez, A. Nordmann, J. Staschulat, and J. von Mendel, *Micro-ROS*, pp. 3–55. Cham, Switzerland: Springer, 2023.

[85] L. Siefke, V. Sommer, B. Wudka, and C. Thomas, "Robotic systems of systems based on a decentralized service-oriented architecture," *Robotics*, vol. 9, no. 4, p. 78, 2020.

[86] A. Afzal, D. S. Katz, C. Le Goues, and C. S. Timperley, "Simulation for robotics test automation: Developer perspectives," in *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, (Porto de Galinhas, Brazil), pp. 263–274, IEEE, 2021.

[87] R. K. Phanden, P. Sharma, and A. Dubey, "A review on simulation in digital twin for aerospace, manufacturing and robotics," *Materials today: proceedings*, vol. 38, pp. 174–178, 2021.

[88] J. Guo, M. Bilal, Y. Qiu, C. Qian, X. Xu, and K.-K. R. Choo, "Survey on digital twins for internet of vehicles: fundamentals, challenges, and opportunities," *Digital Communications and Networks*, 2022.

[89] A. Fuller, Z. Fan, C. Day, and C. Barlow, "Digital twin: Enabling technologies, challenges and open research," *IEEE Access*, vol. 8, pp. 108952–108971, 2020.

[90] M. Segovia and J. Garcia-Alfaro, "Design, modeling and implementation of digital twins," *Sensors*, vol. 22, no. 14, p. 5396, 2022.

[91] Z. Makhataeva and H. A. Varol, "Augmented reality for robotics: A review," *Robotics*, vol. 9, no. 2, p. 21, 2020.

[92] W. Hoenig, C. Milanes, L. Scaria, T. Phan, M. Bolas, and N. Ayanian, "Mixed reality for robotics," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Hamburg, Germany), pp. 5382–5387, IEEE, 2015.

[93] T. S. Sievers, B. Schmitt, P. Rückert, M. Petersen, and K. Tracht, "Concept of a mixed-reality learning environment for collaborative robotics," *Procedia Manufacturing*, vol. 45, pp. 19–24, 2020.

[94] F. J. Mañas-Álvarez, M. Guinaldo, R. Dormido, and S. Dormido, "Robotic park. multi-agent platform for teaching control and robotics," *IEEE Access*, vol. 11, pp. 34899–34911, 2023.

[95] F. J. Mañas-Álvarez, M. Guinaldo, R. Dormido, R. Socas, and S. Dormido, "Control basado en eventos mediante umbral relativo aplicado al control de altitud de cuadricópteros crazyflie 2.1," in *XLII Jornadas de Automática: libro de actas*, (Castelló, Spain), pp. 341–348, 2021.

[96] F. Mañas-Álvarez, R. Dormido, M. Guinaldo, R. Socas, and S. Dormido, "A vision based navigation platform for control learning," *IFAC-PapersOnLine*, vol. 55, no. 17, pp. 138–143, 2022.

[97] F. J. Mañas-Álvarez, M. Guinaldo, R. Dormido, and S. Dormido-Canto, "Scalability of cyber-physical systems with real and virtual robots in ros 2," *Sensors*, vol. 23, no. 13, p. 6073, 2023.

[98] F. J. Mañas-Álvarez, M. Guinaldo, R. Dormido, and S. Dormido, "Muestreo y comunicación: impacto en el control de formaciones en sistemas multi-robot heterogéneos," *Revista Iberoamericana de Automática e Informática industrial*, 2023.

[99] F. J. Mañas-Álvarez, M. Guinaldo, R. Dormido, and S. Dormido, "Análisis de la frecuencia de muestreo en sistemas multi-robot," in *XLIV Jornadas de Automática*, pp. 329–334, Universidade da Coruña. Servizo de Publicacións, 2023.

[100] I. Tejado, J. Serrano, E. Pérez, D. Torres, and B. M. Vinagre, "Low-cost hardware-in-the-loop testbed of a mobile robot to support learning in automatic control and robotics," *IFAC-PapersOnLine*, vol. 49, no. 6, pp. 242–247, 2016.

[101] F. Alkhawaja, M. Jaradat, and L. Romdhane, "Techniques of indoor positioning systems (IPS): A survey," in *2019 Advances in Science and Engineering Technology International Conferences (ASET)*, (Dubai, United Arab Emirates), pp. 1–8, IEEE, 2019.

[102] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3, (Sendai, Japan), pp. 2149–2154, IEEE, 2004.

[103] O. Michel, "Cyberbotics ltd. webots™: professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004.

[104] S. Wilson, P. Glotfelter, L. Wang, S. Mayya, G. Notomista, M. Mote, and M. Egerstedt, "The robotarium: Globally impactful opportunities, challenges, and lessons learned in remote-access, distributed control of multirobot systems," *IEEE Control Systems Magazine*, vol. 40, no. 1, pp. 26–44, 2020.

[105] S. Wilson, P. Glotfelter, S. Mayya, G. Notomista, Y. Emam, X. Cai, and M. Egerstedt, "The robotarium: Automation of a remotely accessible, multi-robot testbed," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2922–2929, 2021.

[106] S. Wilson and M. Egerstedt, "The robotarium: A remotely-accessible, multi-robot testbed for control research and education," *IEEE Open Journal of Control Systems*, pp. 12–23, 2023.

[107] D. Pickem, M. Lee, and M. Egerstedt, “The gritsbot in its natural habitat-a multi-robot testbed,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, (Seattle, WA, USA), pp. 4062–4067, IEEE, 2015.

[108] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. F. Chen, C. Choi, J. Dusek, Y. Fang, *et al.*, “Duckietown: an open, inexpensive and flexible platform for autonomy education and research,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, (Singapore), pp. 1497–1504, IEEE, 2017.

[109] J. Tani, L. Paull, M. T. Zuber, D. Rus, J. How, J. Leonard, and A. Censi, “Duckietown: an innovative way to teach autonomy,” in *Educational Robotics in the Makers Era 1*, (Cham, Switzerland), pp. 104–121, Springer, 2017.

[110] J. Enright, M. Hilstad, A. Saenz-Otero, and D. Miller, “The spheres guest scientist program: Collaborative science on the iss,” in *2004 IEEE Aerospace Conference Proceedings (IEEE Cat. No. 04TH8720)*, vol. 1, (Big Sky, MT, USA), IEEE, 2004.

[111] W. Jo, J. Kim, R. Wang, J. Pan, R. K. Senthilkumaran, and B.-C. Min, “SMARTmBOT: A ROS2-based Low-cost and Open-source Mobile Robot Platform,” *arXiv preprint*, 2022.

[112] S. Lupashin, M. Hehn, M. W. Mueller, A. P. Schoellig, M. Sherback, and R. D’Andrea, “A platform for aerial robotics research and demonstration: The flying machine arena,” *Mechatronics*, vol. 24, no. 1, pp. 41–54, 2014.

[113] J. Sun and R. Gill, “Vehicle platoon control with virtual path constraints,” in *2019 IEEE Conference on Control Technology and Applications (CCTA)*, (Hong Kong, China), pp. 456–461, IEEE, 2019.

[114] Q. Sun, W. Qi, C. Liang, B. Lin, F. Maurelli, and H. Qian, “Sailboat test arena (star): A remotely accessible platform for robotic sailboat research,” *Journal of Marine Science and Engineering*, vol. 11, no. 2, p. 297, 2023.

[115] R. Ou, C. Liang, X. Ji, and H. Qian, “Design and energy consumption optimization of an automatic hybrid sailboat,” in *2021 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, (Xining, China), pp. 1414–1418, IEEE, 2021.

[116] T. Zhivkov, E. Schneider, and E. Sklar, “Mrcomm: Multi-robot communication testbed,” in *Towards Autonomous Robotic Systems: 20th Annual Conference (TAROS)*, (London, UK), pp. 346–357, Springer, 2019.

[117] S. Gupta, S. Shekhar, K. Karpe, A. Ghosh, G. Js, P. Srinivas, M. Kumar, P. Sharma, A. Sinha, K. Singh, *et al.*, “Logiswarm: A low-cost multi-robot testbed for cooperative transport research,” *Multimedia Tools and Applications*, vol. 81, no. 19, pp. 27339–27362, 2022.

[118] T. Farnham, S. Jones, A. Aijaz, Y. Jin, I. Mavromatis, U. Raza, A. Portelli, A. Stanoev, and M. Sooriyabandara, “Umbrella collaborative robotics testbed and iot platform,” in *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, (Las Vegas, NV, USA), pp. 1–7, IEEE, 2021.

[119] I. Mavromatis, A. Stanoev, A. J. Portelli, C. Lockie, M. Ammann, Y. Jin, and M. Sooriyabandara, “Reliable iot firmware updates: A large-scale mesh network performance investigation,” in *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, (Austin, TX, USA), pp. 108–113, IEEE, 2022.

[120] K. DeMarco, E. Squires, M. Day, and C. Pippin, “Simulating collaborative robots in a massive multi-agent game environment (scrimmage),” in *Distributed Autonomous Robotic Systems: The 14th International Symposium*, (Cham, Switzerland), pp. 283–297, Springer, 2019.

[121] S. Jones, E. Milner, M. Sooriyabandara, and S. Hauert, “Dots: An open testbed for industrial swarm robotic solutions,” *arXiv preprint*, 2022.

[122] R. Torbati, S. Lohiya, S. Singh, M. S. Nigam, and H. Ravichandar, “Marbler: An open platform for standarized evaluation of multi-robot reinforcement learning algorithms,” *arXiv preprint*, 2023.

[123] L. Pichierri, A. Testa, and G. Notarstefano, “Crazychoir: Flying swarms of crazyflie quadrotors in ros 2,” *IEEE Robotics and Automation Letters*, 2023.

[124] W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński, and P. Kozierski, “Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering,” in *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, (Miedzyzdroje, Poland), pp. 37–42, IEEE, 2017.

[125] J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian, “Crazyswarm: A large nanoquadcopter swarm,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, (Singapore), pp. 3299–3304, IEEE, 2017.

[126] W. Giernacki, J. Rao, S. Sladic, A. Bondyra, M. Retinger, and T. Espinoza-Fraire, “Dji tello quadrotor as a platform for research and education in mobile robotics and control engineering,” in *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, (Dubrovnik, Croatia), pp. 735–744, IEEE, 2022.

[127] J. M. Soares, I. Navarro, and A. Martinoli, “The khepera iv mobile robot: performance evaluation, sensory data and software toolbox,” in *Robot 2015: second Iberian robotics conference*, vol. 417, (Cham, Switzerland), pp. 767–781, Springer, 2016.

[128] R. Amsters and P. Slaets, “Turtlebot 3 as a robotics education platform,” in *Merdan, M., Lepuschitz, W., Koppensteiner, G., Balogh, R., Obdržálek, D. (eds) Robotics in Education. RiE 2019. Advances in Intelligent Systems and Computing*, pp. 170–181, Springer, 2020.

[129] C. Sandamini, M. W. P. Maduranga, V. Tilwari, J. Yahaya, F. Qamar, Q. N. Nguyen, and S. R. A. Ibrahim, “A review of indoor positioning systems for uav localization with machine learning algorithms,” *Electronics*, vol. 12, no. 7, p. 1533, 2023.

[130] J. Huang, S. Junginger, H. Liu, and K. Thurow, “Indoor positioning systems of mobile robots: A review,” *Robotics*, vol. 12, no. 2, p. 47, 2023.

[131] P. Pascacio, S. Casteleyn, J. Torres-Sospedra, E. S. Lohan, and J. Nurmi, “Collaborative indoor positioning systems: A systematic review,” *Sensors*, vol. 21, no. 3, p. 1002, 2021.

[132] A. Marquez, B. Tank, S. K. Meghani, S. Ahmed, and K. Tepe, "Accurate uwb and imu based indoor localization for autonomous robots," in *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, (Windsor, ON, Canada), pp. 1–4, IEEE, 2017.

[133] A. Taffanel, B. Rousselot, J. Danielsson, K. McGuire, K. Richardsson, M. Eliasson, T. Antonsson, and W. Hönig, "Lighthouse positioning system: dataset, accuracy, and precision for uav research," *arXiv preprint*, 2021.

[134] B. Acosta, W. Yang, and M. Posa, "Validating robotics simulators on real-world impacts," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6471–6478, 2022.

[135] M. S. P. De Melo, J. G. da Silva Neto, P. J. L. Da Silva, J. M. X. N. Teixeira, and V. Teichrieb, "Analysis and comparison of robotics 3d simulators," in *2019 21st Symposium on Virtual and Augmented Reality (SVR)*, (Rio de Janeiro, Brazil), pp. 242–251, IEEE, 2019.

[136] J. Weisz, Y. Huang, F. Lier, S. Sethumadhavan, and P. Allen, "Robobench: Towards sustainable robotics system benchmarking," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, (Stockholm, Sweden), pp. 3383–3389, IEEE, 2016.

[137] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt, "The robotarium: A remotely accessible swarm robotics research testbed," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, (Singapore), pp. 1699–1706, IEEE, 2017.

[138] E. Potokar, S. Ashford, M. Kaess, and J. G. Mangelson, "Holoocean: An underwater robotics simulator," in *2022 International Conference on Robotics and Automation (ICRA)*, (Philadelphia, PA, USA), pp. 3040–3046, IEEE, 2022.

[139] F. Rosique, P. J. Navarro, C. Fernández, and A. Padilla, "A systematic review of perception system and simulators for autonomous vehicles research," *Sensors*, vol. 19, no. 3, p. 648, 2019.

[140] M. Körber, J. Lange, S. Rediske, S. Steinmann, and R. Glück, "Comparing popular simulation environments in the scope of robotics and reinforcement learning," *arXiv preprint*, 2021.

[141] C. Calderón-Arce, J. C. Brenes-Torres, and R. Solis-Ortega, "Swarm robotics: Simulators, platforms and applications review," *Computation*, vol. 10, no. 6, p. 80, 2022.

[142] T. R. Coles, D. Meglan, and N. W. John, "The role of haptics in medical training simulators: A survey of the state of the art," *IEEE Transactions on haptics*, vol. 4, no. 1, pp. 51–66, 2010.

[143] C. L. Foronda, M. Fernandez-Burgos, C. Nadeau, C. N. Kelley, and M. N. Henry, "Virtual simulation in nursing education: a systematic review spanning 1996 to 2018," *Simulation in Healthcare*, vol. 15, no. 1, pp. 46–54, 2020.

[144] J. Collins, S. Chand, A. Vanderkop, and D. Howard, "A review of physics simulators for robotic applications," *IEEE Access*, vol. 9, pp. 51416–51431, 2021.

[145] E. Rohmer, S. P. Singh, and M. Freese, “V-rep: A versatile and scalable robot simulation framework,” in *2013 IEEE/RSJ international conference on intelligent robots and systems*, (Tokyo, Japan), pp. 1321–1326, IEEE, 2013.

[146] E. Coumans and Y. Bai, “Pybullet, a python module for physics simulation for games, robotics and machine learning.” `http://pybullet.org`, 2016–2021.

[147] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, “Sim-to-real: Learning agile locomotion for quadruped robots,” *arXiv preprint*, 2018.

[148] D. Xu, S. Nair, Y. Zhu, J. Gao, A. Garg, L. Fei-Fei, and S. Savarese, “Neural task programming: Learning to generalize across hierarchical tasks,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, (Brisbane, QLD, Australia), pp. 3795–3802, IEEE, 2018.

[149] G. Silano and L. Iannelli, *CrazyS: A Software-in-the-Loop Simulation Platform for the Crazyflie 2.0 Nano-Quadcopter*, pp. 81–115. Cham, Switzerland: Springer, 2020.

[150] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, “RotorS—A Modular Gazebo MAV Simulator Framework,” in *Koubaa, A. (eds) Robot Operating System (ROS). Studies in Computational Intelligence*, pp. 595–625, Cham, Switzerland: Springer, 2016.

[151] F.-J. Mañas-Álvarez, M. Guinaldo, R. Dormido, R. Socas, and S. Dormido, “Formation by consensus in heterogeneous robotic swarms with twins-in-the-loop,” in *ROBOT2022: Fifth Iberian Robotics Conference*, (Cham, Switzerland), pp. 435–447, Springer, 2023.

[152] R. Velázquez and A. Lay-Ekuakille, “A review of models and structures for wheeled mobile robots: Four case studies,” in *2011 15th International Conference on Advanced Robotics (ICAR)*, (Tallinn, Estonia), pp. 524–529, IEEE, 2011.

[153] G. Farias, E. Fabregas, E. Peralta, E. Torres, and S. Dormido, “A khepera iv library for robotic control education using v-rep,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 9150–9155, 2017.

[154] J. Cornejo, J. Magallanes, E. Denegri, and R. Canahuire, “Trajectory tracking control of a differential wheeled mobile robot: a polar coordinates control and lqr comparison,” in *2018 IEEE XXV International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*, (Lima, Peru), pp. 1–4, IEEE, 2018.

[155] U. Zangina, S. Buyamin, M. S. Z. Abidin, and M. S. Azimi, “Non-linear pid controller for trajectory tracking of a differential drive mobile robot,” *Journal of Mechanical Engineering Research and Developments*, vol. 43, no. 7, pp. 255–269, 2020.

[156] H. Xie, J. Zheng, R. Chai, and H. T. Nguyen, “Robust tracking control of a differential drive wheeled mobile robot using fast nonsingular terminal sliding mode,” *Computers & Electrical Engineering*, vol. 96, p. 107488, 2021.

[157] W. Jiang and W. Ge, “Modeling and $h\infty$ robust control for mobile robot,” in *2008 IEEE Conference on Robotics, Automation and Mechatronics*, (Chengdu, China), pp. 1108–1112, IEEE, 2008.

[158] R. Socas, S. Dormido, R. Dormido, and E. Fabregas, “Event-based control strategy for mobile robots in wireless environments,” *Sensors*, vol. 15, no. 12, pp. 30076–30092, 2015.

[159] A. Durand-Petiteville and V. Cadenat, “Visual predictive control for differential drive robots with parallel implementation on gpu,” *Computers and Electrical Engineering*, vol. 102, p. 108120, 2022.

[160] K. Khnissi, C. Seddik, and H. Seddik, “Smart navigation of mobile robot using neural network controller,” in *2018 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, (El Oued, Algeria), pp. 205–210, IEEE, 2018.

[161] N. S. Utami, A. Jazidie, and R. E. A. Kadier, “Path planning for differential drive mobile robot to avoid static obstacles collision using modified crossover genetic algorithm,” in *2019 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, (Surabaya, Indonesia), pp. 282–287, IEEE, 2019.

[162] A. Štefek, V. T. Pham, V. Krivanek, and K. L. Pham, “Optimization of fuzzy logic controller used for a differential drive wheeled mobile robot,” *Applied Sciences*, vol. 11, no. 13, p. 6023, 2021.

[163] J. Heikkinen, T. Minav, and A. D. Stotckaia, “Self-tuning parameter fuzzy pid controller for autonomous differential drive mobile robot,” in *2017 XX IEEE international conference on soft computing and measurements (SCM)*, (St. Petersburg, Russia), pp. 382–385, IEEE, 2017.

[164] G. Li, A. Yamashita, H. Asama, and Y. Tamura, “An efficient improved artificial potential field based regression search method for robot path planning,” in *2012 IEEE International Conference on Mechatronics and Automation*, (Chengdu, China), pp. 1227–1232, IEEE, 2012.

[165] G. Farias, G. Garcia, G. Montenegro, E. Fabregas, S. Dormido-Canto, and S. Dormido, “Reinforcement learning for position control problem of a mobile robot,” *IEEE Access*, vol. 8, pp. 152941–152951, 2020.

[166] E. Fabregas, G. Farias, E. Aranda-Escolástico, G. Garcia, D. Chaos, S. Dormido-Canto, and S. D. Bencomo, “Simulation and experimental results of a new control strategy for point stabilization of nonholonomic mobile robots,” *IEEE Transactions on Industrial Electronics*, vol. 67, no. 8, pp. 6679–6687, 2019.

[167] B. Han, Y. Zhou, K. K. Deveerasetty, and C. Hu, “A review of control algorithms for quadrotor,” in *2018 IEEE International Conference on Information and Automation (ICIA)*, (Wuyishan, China), pp. 951–956, IEEE, 2018.

[168] H. T. Nguyen, T. V. Quyen, C. V. Nguyen, A. M. Le, H. T. Tran, and M. T. Nguyen, “Control algorithms for uavs: A comprehensive survey,” *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, vol. 7, no. 23, 2020.

[169] E. Paiva, J. Soto, J. Salinas, and W. Ipanaqué, “Modeling, simulation and implementation of a modified pid controller for stabilizing a quadcopter,” in *2016 IEEE International Conference on Automatica (ICA-ACCA)*, (Curico, Chile), pp. 1–6, IEEE, 2016.

[170] A. A. Najm and I. K. Ibraheem, "Nonlinear pid controller design for a 6-dof uav quadrotor system," *Engineering Science and Technology, an International Journal*, vol. 22, no. 4, pp. 1087–1097, 2019.

[171] J. Barra, G. Scorletti, S. Lesecq, M. Zarudniev, and E. Blanco, "Attraction domain estimation of linear controllers for the attitude control of vtol vehicles: P/pi control of a quadrotor," in *2020 European Control Conference (ECC)*, (St. Petersburg, Russia), pp. 1644–1649, IEEE, 2020.

[172] I. Siti, M. Mjahed, H. Ayad, and A. El Kari, "New trajectory tracking approach for a quadcopter using genetic algorithm and reference model methods," *Applied Sciences*, vol. 9, no. 9, p. 1780, 2019.

[173] E. Kuantama, T. Vesselenyi, S. Dzitac, and R. Tarca, "Pid and fuzzy-pid control model for quadcopter attitude with disturbance parameter," *International journal of computers communications & control*, vol. 12, no. 4, pp. 519–532, 2017.

[174] J. P. Ortiz, L. I. Minchala, and M. J. Reinoso, "Nonlinear robust h-infinity pid controller for the multivariable system quadrotor," *IEEE Latin America Transactions*, vol. 14, no. 3, pp. 1176–1183, 2016.

[175] H. L. N. N. Thanh and S. K. Hong, "Quadcopter robust adaptive second order sliding mode control based on pid sliding surface," *IEEE Access*, vol. 6, pp. 66850–66860, 2018.

[176] J.-F. Guerrero-Castellanos, N. Marchand, S. Durand, A. Vega-Alonzo, and J. J. Téllez-Guzmán, "Event-triggered attitude control for flying robots using an event approach based on the control," in *2015 International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*, (Krakow, Poland), pp. 1–8, IEEE, 2015.

[177] H. Razmi and S. Afshinfar, "Neural network-based adaptive sliding mode control design for position and attitude control of a quadrotor uav," *Aerospace Science and Technology*, vol. 91, pp. 12–27, 2019.

[178] J. Moreno-Valenzuela, R. Pérez-Alcocer, M. Guerrero-Medina, and A. Dzul, "Nonlinear pid-type controller for quadrotor trajectory tracking," *IEEE/ASME Transactions on Mechatronics*, vol. 23, no. 5, pp. 2436–2447, 2018.

[179] S. Abdelhay and A. Zakriti, "Modeling of a quadcopter trajectory tracking system using pid controller," *Procedia Manufacturing*, vol. 32, pp. 564–571, 2019.

[180] N. P. Nguyen, D. Park, D. N. Ngoc, N. Xuan-Mung, T. T. Huynh, T. N. Nguyen, and S. K. Hong, "Quadrotor formation control via terminal sliding mode approach: Theory and experiment results," *Drones*, vol. 6, no. 7, p. 172, 2022.

[181] M. Shirzadeh, A. Amirkhani, N. Tork, and H. Taghavifar, "Trajectory tracking of a quadrotor using a robust adaptive type-2 fuzzy neural controller optimized by cuckoo algorithm," *ISA transactions*, vol. 114, pp. 171–190, 2021.

[182] F. Jiang, F. Pourpanah, and Q. Hao, "Design, Implementation, and Evaluation of a Neural-Network-Based Quadcopter UAV System," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 3, pp. 2076–2085, 2019.

[183] J. L. Mendoza-Soto and H. R. Cortés, "Generalized predictive control for trajectory tracking of a quadcopter vehicle," in *2017 international conference on unmanned aircraft systems (ICUAS)*, (Miami, FL, USA), pp. 206–212, IEEE, 2017.

[184] C. A. Iyer, "Model Predictive Control (MPC) of quadrotors using LPV techniques," Master's thesis, Universitat Politècnica de Catalunya, 2020.

[185] H. Merabti, I. Bouchachi, and K. Belarbi, "Nonlinear model predictive control of quadcopter," in *2015 16th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, (Monastir, Tunisia), pp. 208–211, IEEE, 2015.

[186] C. Luis and J. L. Ny, "Design of a trajectory tracking controller for a nanoquadcopter," technical report, Mobile Robotics and Autonomous Systems Laboratory, Polytechnique Montreal, Montréal, QC, Canada, 2016.

[187] F. Candan, A. Beke, and T. Kumbasar, "Design and deployment of fuzzy pid controllers to the nano quadcopter crazyflie 2.0," in *2018 Innovations in Intelligent Systems and Applications (INISTA)*, (Thessaloniki, Greece), pp. 1–6, IEEE, 2018.

[188] G. A. Garcia, A. R. Kim, E. Jackson, S. S. Keshmiri, and D. Shukla, "Modeling and flight control of a commercial nano quadrotor," in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, (Miami, FL, USA), pp. 524–532, IEEE, 2017.

[189] M. Greiff, "Modelling and control of the crazyflie quadrotor for aggressive and autonomous flight by optical flow driven state estimation," Master Thesis, Department of Automatic Control, Lund University, Lund, Sweden, 2017.

[190] S. Green and P. Månsson, "Autonomous control of unmanned aerial multi-agent networks in confined spaces," student paper, Department of Automatic Control, Lund University, Lund, Sweden, 2019.

[191] Greiff, Marcus, *Nonlinear Control of Unmanned Aerial Vehicles : Systems With an Attitude.* PhD thesis, Lund University, Lund, Sweden, Oct 2021.

[192] M. Odelga, P. Stegagno, and H. H. Bülthoff, "A fully actuated quadrotor uav with a propeller tilting mechanism: Modeling and control," in *2016 IEEE international conference on advanced intelligent mechatronics (AIM)*, (Banff, AB, Canada), pp. 306–311, IEEE, 2016.

[193] P. Zheng, X. Tan, B. B. Kocer, E. Yang, and M. Kovac, "Tiltdrone: A fully-actuated tilting quadrotor platform," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6845–6852, 2020.

[194] A. González-Morgado, C. Álvarez-Cía, G. H. Benot, and A. O. Baturone, "Uav fully-actuated: modelo, control y comparación con configuración coplanaria," *Revista Iberoamericana de Automática e Informática industrial*, 2023.

[195] R. Martínez, E. Zamora, H. Sossa, F. Arce, and L. A. Soriano, "Orientation modeling using quaternions and rational trigonometry," *Machines*, vol. 10, no. 9, p. 749, 2022.

[196] J. H. Blakelock, *Automatic control of aircraft and missiles.* Hoboken, NJ, USA: John Wiley & Sons, 1991.

[197] J. Peraire and S. Widnall, "Lecture l28 - 3d rigid body dynamics: Equations of motion.," 2009. MIT OpenCourseWare, Dynamics Fall 2009. Available online: `http://ocw.mit.edu`.

[198] J. Peraire and S. Widnall, "Lecture l29-3d rigid body dynamics," 2009. MIT OpenCourseWare, Dynamics Fall 2009. Available online: `http://ocw.mit.edu`.

[199] E. Greitzer, Z. Spakovszky, and I. Waitz, "Thermodynamics and propulsion," *Lecture Notes, Massachusetts Institute of Technology*, 2006.

[200] F. Sabatino, "Quadrotor control: modeling, nonlinearcontrol design, and simulation," Master Thesis, Department of Automatic Control, KTH Royal Institute of Technology, Stockholm University, Stockholm, Sweden, 2015.

[201] S. Bouabdallah, A. Noth, and R. Siegwart, "PID vs LQ control techniques applied to an indoor micro quadrotor," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3, (Sendai, Japan), pp. 2451–2456, IEEE, 2004.

[202] R. Garcia, F. Rubio, and M. Ortega, "Robust pid control of the quadrotor helicopter," *IFAC Proceedings Volumes*, vol. 45, no. 3, pp. 229–234, 2012.

[203] F. Morilla, J. Garrido, and F. Vázquez, "Control multivariable por desacoplo," *Revista Iberoamericana de Automática e Informática Industrial*, vol. 10, no. 1, pp. 3–17, 2013.

[204] V. Vasyutynskyy and K. Kabitzsch, "Time constraints in pid controls with send-on-delta," *IFAC Proceedings Volumes*, vol. 42, no. 3, pp. 48–55, 2009.

[205] N. E. Leonard, D. A. Paley, F. Lekien, R. Sepulchre, D. M. Fratantoni, and R. E. Davis, "Collective motion, sensor networks, and ocean sampling," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 48–74, 2007.

[206] B. Fidan, C. Yu, and B. D. Anderson, "Acquiring and maintaining persistence of autonomous multi-vehicle formations," *IET Control Theory & Applications*, vol. 1, no. 2, pp. 452–460, 2007.

[207] M. Aranda, G. López-Nicolás, C. Sagüés, and Y. Mezouar, "Formation control of mobile robots using multiple aerial cameras," *IEEE Transactions on Robotics*, vol. 31, no. 4, pp. 1064–1071, 2015.

[208] H. G. De Marina, M. Cao, and B. Jayawardhana, "Controlling rigid formations of mobile agents under inconsistent measurements," *IEEE Transactions on Robotics*, vol. 31, no. 1, pp. 31–39, 2014.

[209] G. Pantelimon, K. Tepe, R. Carriveau, and S. Ahmed, "Survey of multi-agent communication strategies for information exchange and mission control of drone deployments," *Journal of Intelligent & Robotic Systems*, vol. 95, pp. 779–788, 2019.

[210] C. Godsil and G. F. Royle, *Algebraic graph theory*, vol. 207. Springer Science & Business Media, 2001.

[211] L. Asimow and B. Roth, "The rigidity of graphs, II," *Journal of Mathematical Analysis and Applications*, vol. 68, no. 1, pp. 171–190, 1979.

[212] B. D. Anderson, C. Yu, B. Fidan, and J. M. Hendrickx, "Rigid graph control architectures for autonomous formations," *IEEE Control Systems Magazine*, vol. 28, no. 6, pp. 48–63, 2008.

[213] L. Krick, M. E. Broucke, and B. A. Francis, "Stabilisation of infinitesimally rigid formations of multi-robot networks," *International Journal of control*, vol. 82, no. 3, pp. 423–439, 2009.

[214] Z. Sun, U. Helmke, and B. D. Anderson, "Rigid formation shape control in general dimensions: an invariance principle and open problems," in *2015 54th IEEE Conference on Decision and Control (CDC)*, (Osaka, Japan), pp. 6095–6100, IEEE, 2015.

[215] D. P. Hardin, T. Michaels, and E. B. Saff, "A comparison of popular point configurations on $\mathbb{S}^2$," *Dolomites Research Notes on Approximation*, vol. 9, no. 1, 2016.

[216] S. Macenski, F. Martín, R. White, and J. G. Clavero, "The marathon 2: A navigation system," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Las Vegas, NV, USA), pp. 2718–2725, IEEE, 2020.

[217] J. L. B. Claraco and M. Perception, "Development of scientific applications with the mobile robot programming toolkit," *The MRPT reference book. Machine Perception and Intelligent Robotics Laboratory, University of Málaga, Málaga, Spain*, vol. 40, 2008.

[218] D. Burk, A. Völz, and K. Graichen, "Experimental validation of the open-source dmpc framework grampc-d applied to the remotely accessible robotarium," in *2021 IEEE International Conference on Mechatronics and Automation (ICMA)*, (Takamatsu, Japan), pp. 442–447, IEEE, 2021.

[219] Q. Yang and R. Parasuraman, "Game-theoretic utility tree for multi-robot cooperative pursuit strategy," in *ISR Europe 2022; 54th International Symposium on Robotics*, (Munich, Germany), pp. 1–7, VDE, 2022.

[220] J. Tani, A. F. Daniele, G. Bernasconi, A. Camus, A. Petrov, A. Courchesne, B. Mehta, R. Suri, T. Zaluska, M. R. Walter, *et al.*, "Integrated benchmarking and design for reproducible and accessible evaluation of robotic agents," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Las Vegas, NV, USA), pp. 6229–6236, IEEE, 2020.

[221] K. S. Lanchukovskaya, D. E. Shabalina, and T. V. Liakh, "Semantic image segmentation methods in the duckietown project," in *2022 IEEE 23rd International Conference of Young Professionals in Electron Devices and Materials (EDM)*, (Altai, Russian Federation), pp. 611–617, IEEE, 2022.

[222] K. Chaika, A. Filatov, A. Filatov, and K. Krinkin, "Automatic wheels and camera calibration for monocular and differential mobile robots," *Applied Sciences*, vol. 11, no. 13, p. 5806, 2021.

[223] C. M. Jewison, B. McCarthy, D. C. Sternberg, D. Strawser, and C. Fang, "Resource aggregated reconfigurable control and risk-allocative path planning for on-orbit servicing and assembly of satellites," in *AIAA Guidance, Navigation, and Control Conference*, (National Harbor, MD, USA), p. 1289, 2014.

[224] D. Fourie, B. Tweddle, S. Ulrich, and A. Saenz Otero, "Vision-based relative navigation and control for autonomous spacecraft inspection of an unknown object," in *AIAA guidance, navigation, and control (GNC) conference*, (Boston, MA, USA), p. 4759, 2013.

[225] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Dance of the flying machines: Methods for designing and executing an aerial dance choreography," *IEEE Robotics & Automation Magazine*, vol. 20, no. 4, pp. 96–104, 2013.

[226] R. Ritz, M. W. Müller, M. Hehn, and R. D'Andrea, "Cooperative quadrocopter ball throwing and catching," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Vilamoura-Algarve, Portugal), pp. 4972–4978, IEEE, 2012.

[227] Z. Zhang, Z. Yao, Q. Sun, and H. Qian, "Energy optimization of automatic hybrid sailboat," in *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, (Kuala Lumpur, Malaysia), pp. 480–485, IEEE, 2018.

[228] W. Qi, Q. Sun, H. Liu, Z. Sun, T. L. Lam, and H. Qian, "Collision avoidance for autonomous sailboats based on rrs protocol," in *2019 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, (Irkutsk, Russia), pp. 13–18, IEEE, 2019.

[229] Y.-J. Zhang, "Camera calibration," in *3-D Computer Vision: Principles, Algorithms and Applications*, pp. 37–65, Singapore: Springer Nature Singapore, 2023.

[230] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[231] C. R. Viala and A. J. S. Salmerón, "Procedimiento completo para el calibrado de cámaras utilizando una plantilla plana," *Revista Iberoamericana de Automática e Informática Industrial RIAI*, vol. 5, no. 1, pp. 93–101, 2008.

Appendix A

Experimetal Platforms overview

This appendix compiles in two tables the most relevant information about the experimental platforms studied in section 2.1. Table A.1 shows administrative information, such as reference publications, developers, publications where it is used, and website if available. Table A.2 details technical information, such as the type of robots, positioning system, or use of ROS/ROS 2.

Platform	Reference	Developer	Uses	Web
Robotic Park	[94]	UNED	-	Yes[1]
Robotarium	[104–106, 137]	Georgia Institute of Technology	[218, 219]	Yes[2]
Duckietown	[108, 109]	MIT	[220–222]	Yes[3]
SPHERES	[110]	MIT	[223, 224]	Yes[4]
SMARTmBOT	[111]	SMART Lab, Purdue University	-	Yes[5]
FMA	[112]	ETH Zürich	[113, 225, 226]	Yes[6]
STAr	[114]	Chinese University of Hong Kong	[115, 227, 228]	No
MRComm	[116]	King's College London, UK	-	No
LOGISWARM	[117]	SRMIST, Kattankulathur, India	-	No
M-sMR	[58]	Escuela Politécnica Nacional, Quito	-	No
UMBRELLA	[118]	Bristol Research and Innovation lab.	[119]	Yes[7]
CrazyChoir	[123]	OPT4SMART Project	-	Yes[8]
SCRIMMAGE	[120]	Georgia Tech Research Institute	-	Yes[9]

Table A.1. Experimentals Platforms Avaible (I)

[1] Robotic Park: https://blogs.uned.es/roboticpark/

[2] Robotarium: https://www.robotarium.gatech.edu/

[3] Duckietown: https://www.duckietown.org/

[4] SPHERES: https://www.nasa.gov/spheres/home

[5] SMARTmBOT: https://github.com/SMARTlab-Purdue/SMARTmBOT

[6] FMA: https://www.flyingmachinearena.ethz.ch/

[7] UMBRELLA: https://www.umbrellaiot.com/about/umbrella-testbed/

[8] CrazyChoir: https://opt4smart.github.io/crazychoir/

[9] SCRIMMAGE: http://www.scrimmagesim.org/

Platform	Enviroment	Agents	Positioning	ROS/ROS 2	Languajes
Robotic Park	Hybrid	Mobile & Aerial Robots	MoCap (Vicon), UWB Ultrasound and Infrared	Yes	C++ / Python
Robotarium	Real/Virtual	Mobile Robots	MoCap (Vicon)	No	Python / Matlab
Duckietown	Real/Virtual	Mobile Robots	Vision based	Yes	Python
SPHERES	Real/Virtual	Satellites	Own/Metrology System	No	C/C++
SMARTmBOT	Real	Low cost Mobile Robots	MoCap (Vicon)	Yes	C++ / Python
FMA	Hybrid	Aerial Robots	MoCap (Vicon)	No	-
STAr	Real	Mini Sail Boats	MoCap	No	Python
MRComm	Real/Virtual	Mobile Robot	-	Yes	Python
LOGISWARM	Real/Virtual	Mobile Robot	Vision based	Yes	-
M-sMR	Real	Mini Mobile Robots	Vision based	Yes	C++ / Python
UMBRELLA	Real/Virtual	Mobile Robots	Radio	Yes	-
CrazyChoir	Real/Virtual	Aerial Robots	MoCap (Vicon)	Yes	Python
SCRIMMAGE	Virtual	Mobile & Aerial Robots	-	Yes	-

Table A.2. Experimentals Platforms Avaible (II)

Appendix B

Performance Index

B.1 Temporal criteria

- Integral Square Error (ISE). This index reflects the square of the overall system error over time. In this index, the error is treated non-linearly and weights all errors equally over time.

$$ISE = \int_0^T e^2(t)dt \tag{B.1}$$

- Integral Absolute Error (IAE). This index represents the overall absolute error by weighting all errors equivalently over time. In this index, errors are treated linearly.

$$IAE = \int_0^T \|e(t)\|dt \tag{B.2}$$

- Integral of Time-weighted Square Error (ITSE). This index is a modification of the ISE in such a way that the quadratic errors are weighted by the time elapsed since the beginning of the experience. This index assigns more weight to errors that persist over time.

$$ITSE = \int_0^T t \cdot e^2(t)dt \tag{B.3}$$

- Integral of Time-weighted Absolute Error (ITAE). This index weights the overall absolute error by the time elapsed since the start of the experience. In systems that use step inputs, as in the case of MRSs at the start of formation, the initial error is always high. For comparisons, it is more relevant that the errors that are maintained over time have a greater weight than the initial errors.

$$ITAE = \int_0^T t \cdot \|e(t)\|dt \tag{B.4}$$

- Root Mean Square Error (RMSE). It summarizes the error over time.

- Overshoot ($OS\,(\%)$). It represents the system over-peak or over-elongation in %, which is the difference between the maximum peak value of the response and the steady-state value, relative to the steady-state value.
- Rise time (t_r). In the underdamped case, it is the time from when the output starts to evolve until it first reaches its steady state value.
- Peak time (t_p). The time taken for the system response to reach its peak value is measured from the instant the step is introduced.
- Settling time (t_s). In formation movement of MARS, time neccesary for formation to remain within a 5% error limit on the desired formation relative to the initial error.

B.2 Computational criteria

- Average frequency ($f\,(samples/s)$). With this parameter we analyze the number of transmissions carried out by the MARS' agents over the time of the experiment. This index shows the flow of information through the communication channel.
- CPU percentaje ($CPU_{node}\,(\%)$). This represents the individual process CPU utilization as a percentage. It can be >100.0 in case of a process running multiple threads on different CPUs.
- Global CPU percentage ($CPU_g\,(\%)$). This value represents the current system-wide CPU utilization as a percentage.
- Real-Time Factor (RTF). This shows a ratio of calculation time within a simulation (simulation time) to execution time (real time).

Appendix C

Camera Calibration

C.1 Introduction

For the use of images in real environment applications it is essential to know the characteristic parameters of the sensors used, in this case, the cameras and their lenses. In digital cameras or Charge-Coupled Device (CCD), the intrinsic parameters define the relationship between the measurements in the reference system of the real environment with the position of the pixels in the image (focal length f, field of view, aperture, scale factor, etc.). The extrinsic parameters define the position and orientation of the camera concerning the real environment using the **Camera Extrinsic Matrix** as follows

$$\begin{pmatrix} p_{c,x} \\ p_{c,y} \\ p_{c,z} \\ 1 \end{pmatrix} = \begin{pmatrix} R_{3\times 3} & t_{3\times 1} \\ 0_{1\times 3} & 1_{1\times 1} \end{pmatrix} p_w \tag{C.1}$$

where p_c represents the coordinates in the camera system, p_w are the coordinates in the global reference frame, R is the transformation matrix related to the orientation and t is the translation vector. In positioning system applications, it is important to define this parameter with the camera fixed to be able to estimate the real position of the objects to be detected. In Visual Simultaneous Localization And Mapping (VSLAM) applications, this estimation allows the reconstruction of the trajectory performed by the sensor.

However, the information in an image is 2D, so the camera model must include the transformation from the 3D environment to the image plane. In this case, we work with the pinhole model, which considers that the rays from one point in space pass through the center of the camera aperture and are projected onto the 2D (image) plane at the other end of the camera. This transformation is with loss as it cannot be undone (depth information is lost). The projection of the new point p_i can be determined by triangle similarity as

$$p_i = f \frac{p_c}{p_{c,z}} \tag{C.2}$$

Therefore, in matrix form, we have the following transformation matrix from the camera coordinate system to the image coordinate system.

$$\begin{pmatrix} p_{i,x} \\ p_{i,y} \\ p_{c,z} \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} p_{c,x} \\ p_{c,y} \\ p_{c,z} \\ 1 \end{pmatrix} \tag{C.3}$$

The next step to obtain the final pixel representation requires discretizing the previously obtained points. A parameter specific to cameras is the dimension of each pixel $(\rho_u \times \rho_v)$. Therefore, the coordinates in pixels are expressed as

$$(x_i, y_i) = \left(f \frac{p_{c,x}}{\rho_u p_{c,z}}, f \frac{p_{c,y}}{\rho_v p_{c,z}} \right) \tag{C.4}$$

Additionally, the standard sets the origin of the image coordinates in the upper left corner so that a translational movement in the pixels (c_x, c_y) is required. So, the final coordinates can be expressed as

$$(u, v) = \left(f \frac{p_{c,x}}{\rho_u p_{c,z}} + c_y, f \frac{p_{c,y}}{\rho_v p_{c,z}} + c_x \right) \tag{C.5}$$

Therefore, the complete conversion from image to pixel coordinates can be expressed in matrix form as (C.6) and is known as the **Camera Intrinsic Matrix**.

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} \frac{f}{\rho_u} & 0 & c_x & 0 \\ 0 & \frac{f}{\rho_v} & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} p_{c,x} \\ p_{c,y} \\ p_{c,z} \\ 1 \end{pmatrix} \tag{C.6}$$

It is also important to determine the image distortion parameters produced by the lens. For correct processing of the coordinates of the image points, the images have to be subjected to a correction process beforehand. There are two types of distortion: radial and tangential. **Radial distortions** are deformations produced by the lens geometry in the image captured on the sensor. This can be seen in the fact that straight lines degenerate into more pronounced curves as they move away from the center of the image. Mathematically it is expressed by the parameters $k_{1,2,3}$ and can be represented as

$$\begin{aligned} x_{distorted} &= x\left(1+k_1r^2+k_2r^4+k_3r^6\right) \\ y_{distorted} &= y\left(1+k_1r^2+k_2r^4+k_3r^6\right) \end{aligned} \tag{C.7}$$

Tangential distortion occurs when the lens is not perfectly aligned with the plane of the sensor. This causes areas of the image to appear closer than expected. Mathematically it is expressed by the parameters $p_{1,2}$ and can be represented as

$$\begin{aligned} x_{distorted} &= x+\left(2p_1xy+p_2\left(r^2+2x^2\right)\right) \\ y_{distorted} &= y+\left(p_1\left(r^2+2y^2\right)+2p_2xy\right) \end{aligned} \tag{C.8}$$

In-depth coverage of the characteristics of the different camera models and their calibration can be found in [229–231]

C.2 Calibration

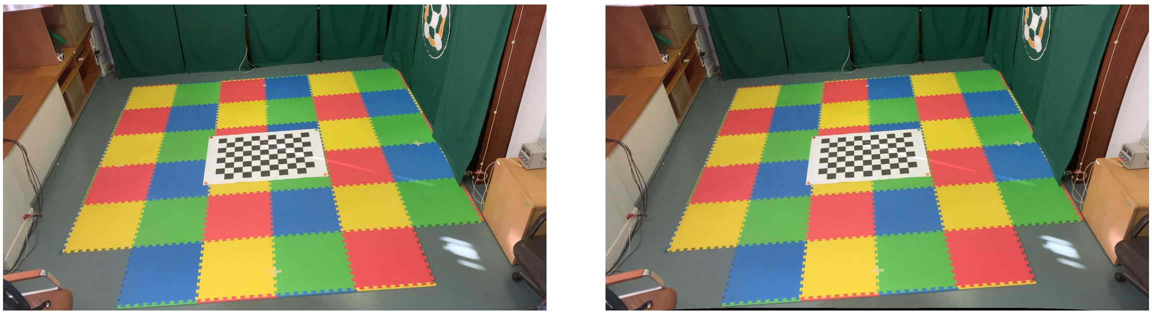
The calibration process of the cameras used in Robotic Park has been carried out using the OpenCV library, the library par excellence in artificial vision works. A calibration pattern of the type 11x8 A0 Checkerboards 90mm squares - 10x7 vertices, 11x8 squares has been used. The code used is shown in Code C.1, where the variable “mtx” is the Camera Intrinsic Matrix, “dist” is the vector containing the distortion parameters in the order $(k_1, k_2, p_1, p_2, k_3)$ and the extrinsic parameters are defined by the rotation matrix “rvecs” and the translation vector “tvecs”.

```python
import numpy as np
import cv2 as cv
import glob
# termination criteria
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 90, 0.001)
# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,
objp = np.zeros((7*10,3), np.float32)
objp[:,:2] = np.mgrid[0:7,0:10].T.reshape(-1,2)
# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.
images = glob.glob('*.png')
for fname in images:
    img = cv.imread(fname)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    # Find the chess board corners
    ret, corners = cv.findChessboardCorners(gray, (7,10), None)
    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)
        corners2 = cv.cornerSubPix(gray,corners, (90,90), (-1,-1), criteria)
        imgpoints.append(corners)
        # Draw and display the corners
        cv.drawChessboardCorners(img, (7,10), corners2, ret)
        cv.imshow('img', img)
        # cv.imwrite("calib.jpg", img)
        cv.waitKey(500)
ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, gray.
    shape[::-1], None, None)
print("M:",mtx)
print("D:",dist)
print("R:",rvecs)
print("T:",tvecs)
cv.destroyAllWindows()
```

Code C.1. Camera calibration code.

C.3 Results

The execution of Code C.1 has been performed on 20 images capturing the pattern used from different angles and positions, the last one being the final position occupied by the cameras in Robotic Park. Figure C.1 shows the results obtained on the iPhone XR camera, where Figure C.1a shows the original image and Figure C.1b the distortion-corrected image. The results obtained for the intrinsic parameters are shown in (C.9), the extrinsic parameters in (C.10), and the distortion parameters in (C.11).



(a) Original frame.

(b) Distortion-corrected frame.

Figure C.1. iPhone XR calibration.

$$M_{iPhone} = \begin{pmatrix} 1649.69 & 0.00 & 960.36 & 0.00 \\ 0.00 & 1654.98 & 542.08 & 0.00 \\ 0.00 & 0.00 & 1.00 & 0.00 \end{pmatrix} \tag{C.9}$$

$$R_{iPhone} = \begin{pmatrix} -0.0438 & -0.5909 & 0.8056 & -4.2080 \\ -0.9975 & 0.0699 & -0.0030 & -0.4917 \\ -0.0546 & -0.8037 & -0.5925 & 2.8478 \\ 0.0000 & 0.0000 & 0.0000 & 1.0000 \end{pmatrix} \tag{C.10}$$

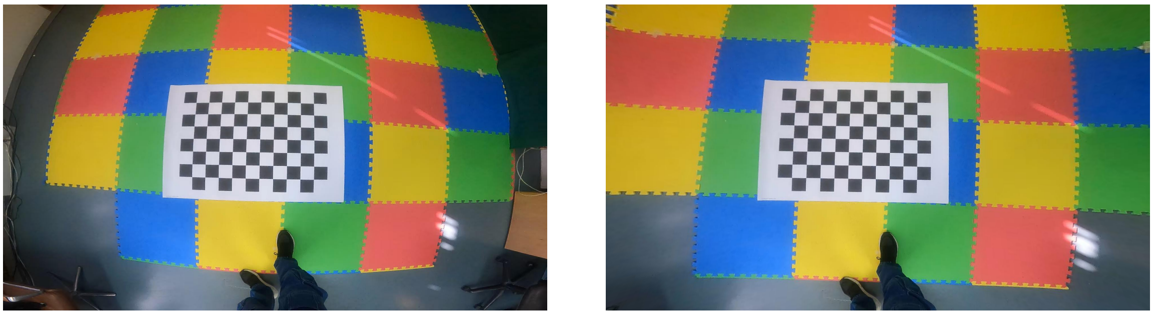
$$D_{iPhone} = \begin{pmatrix} 0.2098 & -0.6622 & 0.0000 & 0.0000 & 0.0000 \end{pmatrix} \tag{C.11}$$

Figure C.2 shows the results obtained on the GoPro HERO 9 Black camera, where Figure C.2a shows the original image and Figure C.2b the distortion-corrected image. The results obtained for the intrinsic parameters are shown in (C.12), the extrinsic parameters in (C.13), and the distortion parameters in (C.14).

$$M_{goPro} = \begin{pmatrix} 882.09 & 0.00 & 964.22 & 0.00 \\ 0.00 & 881.21 & 532.84 & 0.00 \\ 0.00 & 0.00 & 1.00 & 0.00 \end{pmatrix} \tag{C.12}$$

$$R_{goPro} = \begin{pmatrix} 0.9776 & -0.0465 & 0.2051 & -0.9831 \\ -0.1847 & -0.6560 & 0.7318 & -2.7041 \\ 0.1005 & -0.7533 & -0.6500 & 2.3478 \\ 0.0000 & 0.0000 & 0.0000 & 1.0000 \end{pmatrix} \tag{C.13}$$

$$D_{goPro} = \begin{pmatrix} -0.2803 & 0.1403 & 0.0000 & 0.0000 & -0.0475 \end{pmatrix} \tag{C.14}$$



(a) Original frame.

(b) Distortion-corrected frame.

Figure C.2. GoPro HERO 9 Black calibration.