POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Distributed Networked Predictive Control with application to a laboratory experimental setup

Tesi di Laurea Magistrale in
Automation and Control Engineering - Ingegneria dell'Automazione

Author: **Alex Ferrario**

Student ID: 940722
Advisor: Prof. Marcello Farina
Co-advisors: Prof. Luca Bascetta
Academic Year: 2020-21

A mamma, papà e Loris, grazie per essermi sempre stati vicino e avermi sempre sostenuto, se sono la persona di oggi è solamente merito vostro

Abstract

Nowadays, distributed and networked strategies are widely used to control modern infrastructures where geographically dispersed and complex engineering systems can be regarded as "networks" of simpler interacting units.
In this thesis we will implement and test a state-of-the-art model Predictive Control-based Distributed algorithm, named DPC. The tests will be carried out on a laboratory experimental setup devised specifically for this work, consisting of unicycle mobile robots virtually connected through springs and dampers, moving on a bi-dimensional plane. The realized setup will be characterized by non-idealities typical of networked control systems, like delays and disturbances. These undesired phenomena typically deteriorate the performances of control algorithms designed specifically for nominal systems.
To overcome this problem we will present a robust networked variant of the DPC algorithm, capable to deal with the delays and the disturbances affecting the experimental setup.
These phenomena will be characterized by analyzing the used laboratory equipment, i.e., the E-PUCK mobile robots and the Decawave UWB positioning system. Moreover, a specific calibration strategy for the compensation of the biases affecting the position measurement will be provided. In this way, such setup can be used as a versatile benchmark case study for development and testing of networked control strategies for multi-agent systems.
Finally the novel, robust networked DPC algorithm will be tested, and its effectiveness will be discussed.

Keywords: Distributed control, Networked control, Model predictive control, Laboratory setup

Abstract in lingua italiana

Oggigiorno, le tecniche di controllo distribuito e su rete trovano ampio utilizzo nella regolazione di moderni sistemi ingegneristici, dove impianti complessi e dispersi su grandi aree geografiche possono essere visti come una rete di sottosistemi interagenti tra loro.
In questa tesi implementeremo e testeremo un algoritmo distribuito basato sul controllo predittivo, chiamato DPC. I test verranno eseguiti su un setup sperimentale sviluppato specificamente per questo lavoro, dove dei robot mobili a uniciclo virtualmente connessi da molle e smorzatori saranno liberi di muoversi all'interno di un arena. Il setup realizzato è caratterizzato da non idealità tipiche dei sistemi di controllo su rete, come ritardi e disturbi di comunicazione. Questi effetti indesiderati solitamente contribuiscono a degradare le prestazioni fornite da algoritmi di controllo sviluppati appositamente per sistemi nominali.
Per superare questo problema, presenteremo una variante robusta specifica per il controllo su rete del DPC, in grado di sormontare le problematiche introdotte dai ritardi e disturbi che contraddistinguono il setup realizzato.
Queste non idealità verranno caratterizzate analizzando nel dettaglio i dispositivi utilizzati, in particolare i robot E-PUCK e il sistema di posizionamento Decawave basato su tecnologia a banda ultra-larga (UWB). Inoltre verrà presentata una strategia di calibrazione del sistema di posizionamento in grado di migliorare notevolmente le misure di posizione fornite, rimuovendo gli errori di localizzazione. Il setup sperimentale così caratterizzato, potrà essere utilizzato come una versatile piattaforma di riferimento su cui sviluppare e testare strategie di controllo per sistemi multi agente.
Infine, la nuova variante robusta per sistemi su rete dell'algoritmo distribuito basato su controllo predittivo verrà testata, e la sua efficacia verrà analizzata.

Parole chiave: Controllo distribuito, Controllo su rete, Controllo predittivo, Setup sperimentale

Contents

1 | Introduction

1.1. Distributed control

In the last decades infrastructures and engineering systems have grown widely, both in terms of complexity and geographical dispersion, together with the needs of automation. This has spurred the research in control to provide growing performances, in terms of cost, savings and optimality.

The usage of the traditional centralized controllers is widespread, where a unique controller is in charge of controlling the whole plant, taking measurements from all the sensors deployed in the system, and compute the control actions to be sent to all the actuators. This control strategy ensures the greatest performances achievable, since it relies on the perfect knowledge of the system evolution, but at the cost of a high complexity and great computational requirements, that can hugely increase together with the process dimensions.

Modern infrastructures can be often regarded as networks of a (often large) number of simple interacting units, e.g. natural gas treatment plants, fleet of autonomous vehicles, data acquisition sensor networks, irrigation channels, electric grids and many more.

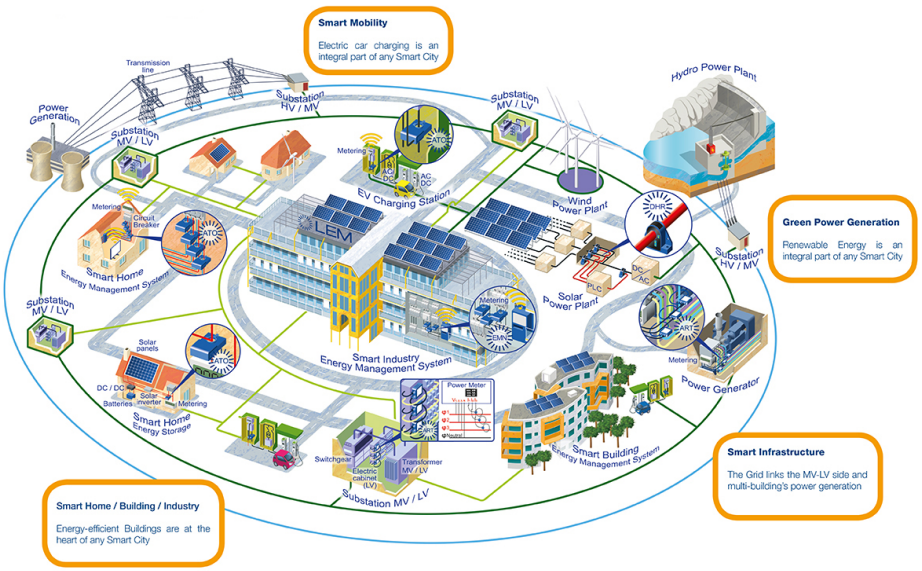


Figure 1.1: Smart grid.

The advent of smart grids can be considered as an example of a paradigm shift in engineering systems and automation. Indeed, in the past, classical energy distribution systems consisted in few large generation plants placed far from users, providing electricity to passive loads. Nowadays, there has been a dramatic change. For instance, due to advent of renewable technologies such as solar panels, wind turbines, etc, at the user side one can supply energy. The opportunities are manifold, e.g., the possibility for a grid to operate in islanded mode, or the possibility of forming clusters of microgrids, which can take the role of a virtual power plant.

This technological change has lead to the necessity of redesigning the control and optimization schemes at different levels, i.e.

- Single generation unit level: wind/solar power plants, smart homes/buildings and industries must achieve stable operating condition and reactive power control
- Microgrid level: the microgrid, regarded as a system, must guarantee the fulfillment of energy needs, maximize gain from local production, minimize purchase from the grid and guarantee frequency/voltage regulation
- Aggregation level: among the many challenges, clusters of multiple microgrids, should be able to provide active/reactive power reserves.

To meet these targets, traditional centralized control structures are not suitable, since control is needed at different levels, where all the entities in the network must be coordinated, and possibly with other entities from other grids placed at large geographical

distances. Systems also require high flexibility, high reliability, e.g., to failure of single agents, and high performing communication channels. To handle this complexity, a single controller is deemed not suitable, and advanced control architectures must be used.

For all these reasons, the adoption of hierarchical, distributed and decentralized control architectures is required.

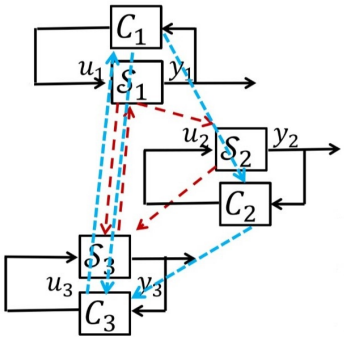


Figure 1.2: Distributed control architecture.

For instance, decentralized and distributed control strategies are developed under the prospective that the system can be regarded as a network of simpler units with physical interactions among them (red arrows in Figure 1.2), each endowed with a dedicated controller. This new (decomposition-based) perspective, introduces a number of advantages with respect to classic centralized strategies. In particular, each regulator has a reduced computational load (scalability), since it is committed to control just a small part of the whole complex process. Also, a smaller communication burden is involved. Local model uncertainties can be easily compensated, and controllers can be made robust to deal with them. Moreover, distributed control schemes are versatile, since single parts can be modified or completely changed, without requiring the re-design of the complete system (modularity).

Decentralized structures do not require communication between local controllers, this can result in instability of the complete process, or in unsatisfactory global performances. On the other hand, distributed architectures require communication among controllers (blue arrows in Figure 1.2), so that cooperation can be enforced between regulators. The latter strategy lies in the middle ground between decentralized and centralized control ones. In fact, it keeps the advantages coming from the use of the decomposition-based approach but, thanks to communication, it can provide nearly optimal performances. Highly complex communication networks can allow for performances comparable with the one ideally provided by centralized architectures. Note that the network complexity can

be considered as a tuning parameter, to be selected to provide satisfactory performances, but at the same time to limit the control problem complexity.

1.2. Networked control

As discussed above, nowadays modern infrastructures and engineering systems are complex. In many cases, systems are provided with a large number of sensors, actuators, and control systems. This requires a relevant exchange of information, that would result in a huge amount of connections between nodes, i.e. controllers, sensors and actuators. Moreover, modern plants may be geographically dispersed among wide areas. For these reasons, in large-scale and complex plants all the necessary information is often shared through serial communication channels, where the nodes are connected by means of a communication network, leading to the concept of networked control systems.

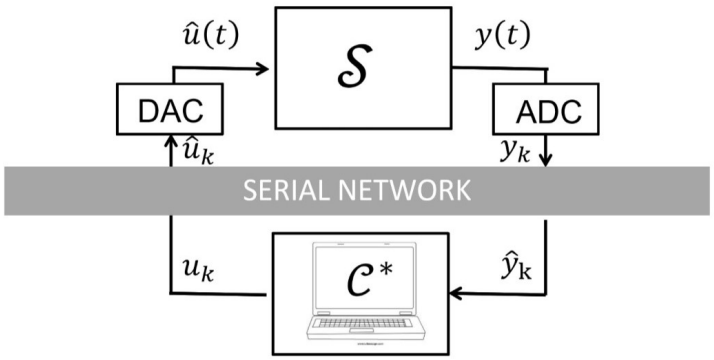


Figure 1.3: Networked control system over serial communication network.

During years a multitude of different network technologies and protocols have been introduced, spanning from the CAN protocol firstly adopted in the 80's to modern protocols integrated through wireless communication channels.

Networked control systems have a long list of benefits. Often, the use of control networks for communication is the only technical solution if the distances are large or the nodes are too many. It also allows for a relevant cost reduction in terms of savings in wiring, since a serial network consists of a unique line instead of hundreds of individual wires. It has been estimated [11] that, as the system grows past the deadline of 100 I/O points, the added expense of network hardware is offset by savings in wiring time. This benefit grows exponentially as the size of the system.

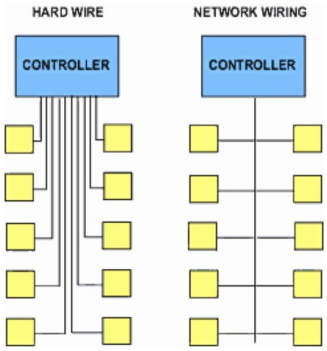


Figure 1.4: Serial communication network. (From [11])

The use of control networks allows for modularity, since connectivity is achieved by software and it does not matter where a component of the network is physically placed. For this reason each node can be easily installed or reconfigured, e.g. for substitution for maintenance. Control networks also provide diagnostic capabilities, since information about the status of single components can be shared with the network, allowing for malfunctioning detection.

The use of serial communication channels introduces a number of benefits, but also some challenges. For example, constraints on the amount of information that can be transmitted per time-unit are introduced, due the limited bandwidth of the channels, which has a major impact on the trade-off between sampling and quantization.
Also, significant delays may affect the system, which can be bounded or unbounded, and time-varying, depending on the type of network used. Delays are due to congestions, competition in accessing the network, and transmission time, which in turn is related to the adopted physical link.
Also, the possibility of packet dropouts must be considered. Indeed, packets may be lost due to errors in nodes and links, buffer overflows during congestions, too long delays, or scheduling time.

Networked control systems must be designed to be robust with respect to all these undesired effects, and must be always capable to provide stability and satisfactory performances.

1.3. Distributed and networked control systems

In Section 1.1, the advantages of distributed control architectures were described. As discussed, distributed strategies rely on communication between controllers, which must

allow to share information and to achieve global stability and satisfactory performances. Since modern systems are complex and large-scale, such data often need control networks to efficiently support their transmission, as discussed in Section 1.2.

It is possible to subdivide the resulting control system into three network layers:

- **System interconnection network** layer 1, considering the dynamic interactions between subsystems
- **Decision network** layer 2, representing the communication among controllers
- **Communication network** layer 3, representing the transmission of data between agents.

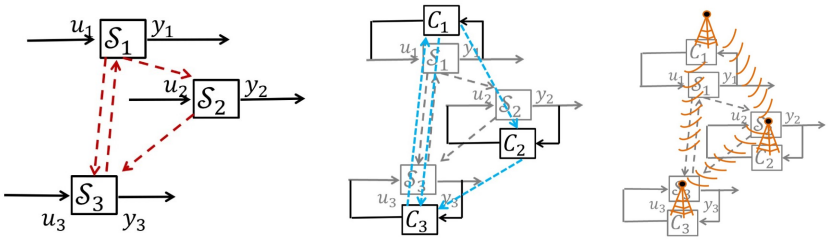


Figure 1.5: (a) Layer 1, (b) Layer 2, (c) Layer 3.

This leads to regard the problems of developing distributed control schemes and to cope with network-induced communication issues as two strictly coupled ones. Distributed and, at the same time, networked control systems guarantee all the benefits introduced in the previous sections, but at the cost of high complexity in the design phase. In fact interactions among subsystems, communication between controllers, and network non-idealties must be taken into account at the same time.

1.4. Purpose of the thesis

The first objective of this work is to devise a lightweight experimental setup of reduced dimensions where tests over the chosen algorithms could be carried out. Such setup can be used as a versatile benchmark case study for development and testing of networked control strategies for multi-agent systems, e.g., distributed and hierarchical approaches. The selected case study is a multi-agent system composed by carts moving on a bi-dimensional plane. Connections between agents can be virtually imposed by software enforcing couplings between agents.

The realization of such an experimental setup implied the usage of different laboratory equipments, like the E-PUCK mobile robots and the Decawave positioning system, which required a complete characterization. Regarding the real time positioning system, we also developed a suitable calibration strategy.

Secondly, a distributed state-of-the-art algorithm based on Model Predictive Control (MPC) is implemented and tested on the realized experimental setup. A complete evaluation of its performances is made, in particular when network non idealties are present. A new algorithm capable to overcome the limitations of the original version is then presented and tested.

1.5. Structure of the thesis

The reminder of the thesis is organized as follows.

- In Chapter 2 the experimental setup developed to be a benchmark application for several distributed and hierarchical control strategies is described. A detailed analysis of the laboratory equipment used is presented, focusing in particular on the Decawave positioning system and the E-PUCK robots
- In Chapter 3 the disturbances and the delays affecting the experimental setup have been characterized, and a strategy for the compensation of the biases affecting the position measurements is presented. Furthermore, the case study in object is discussed
- In Chapter 4 the DPC algorithm [5] is presented, and the implementation details are discussed
- In Chapter 5 a modified version of the DPC algorithm is presented, capable to deal with the non-idealities introduced by the adoption of a networked control system architecture
- In Chapter 6 several simulations testing the performances of the modified version of the DPC compared with the ones of the original version are reported
- In Chapter 7 the novel robust networked variant of the DPC algorithm is tested on the experimental setup
- In Chapter 8 the conclusions, together with the possible future developments of this thesis work, are discussed.

2 | Experimental setup

In this chapter the necessary components for the realization of the real case study later described in Section 3.3, selected in order to highlight the features of the methodological problem addressed in this thesis, are introduced and then analyzed in details.

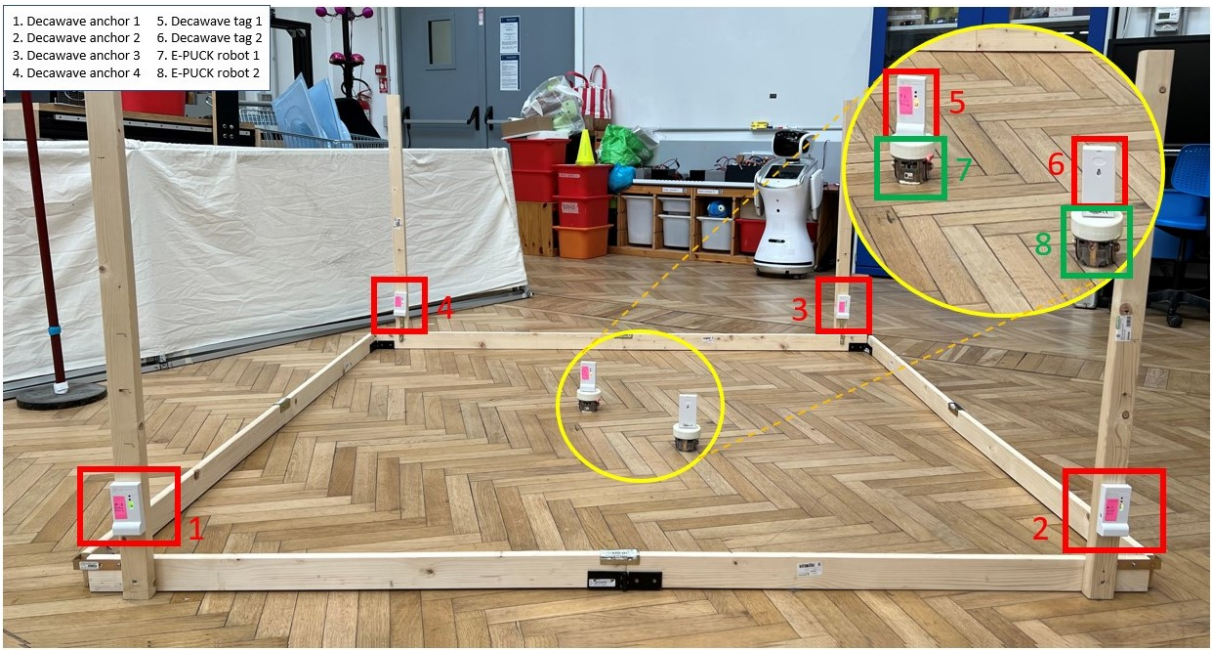


Figure 2.1: Experimental setup.

In particular, the experimental setup consists of two unicycle robotic agents (E-PUCK robots) moving inside an arena of 2x2 meters. The position of the robots is obtained using the Decawave Real Time Localization System (DRTLS) relying on the Ultra-Wide-Band (UWB) technology. The system can greatly benefit of a distributed controller, in view of the fact that strong interactions between the two subsystems will be virtually imposed by software. Also, the variables will be measured through sensors prone to significant noise and delays, which are the typical non-idealities affecting networked control systems. For these reasons this setup perfectly fits to be a benchmark application to test distributed and networked control algorithms.

The distributed control philosophy, at the foundation of this work, implies that each subsystem is endowed with its own regulator, while receiving information from its neighbors through an appropriate communication network. In this application, due to the limited computational power of the mobile robots, a central computer is in charge of running the controllers (implemented as if they were indeed distributed) of all units, while also receiving the measurements from all the sensors in the network. The host computer then sends the control actions, computed at each sampling interval, back to the mobile robots.

The regulator of each agent is implemented in a complete independent fashion and just the strictly necessary information is shared among local controllers, following the desired topology of the virtual network; thus, the obtained scheme is exactly equivalent as each subsystem is running its own controller.

The communication network is based on the Bluetooth technology [1], which is implemented in a large variety of mobile robots, and is ready to use on most of the recent computers. It is perfectly suitable for small scale indoor applications, providing satisfactory performances in terms of reliability and transmission speed.

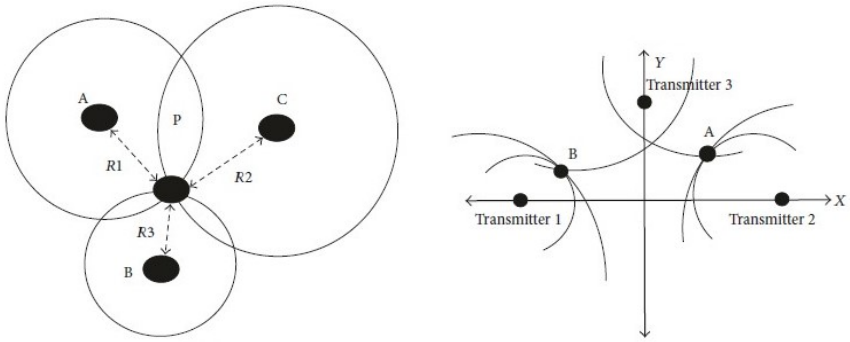
2.1. The localization system

Indoor localization is a wide area of interest. Where, along years, a large variety of technologies have been developed in order to provide a precise position measurement in closed spaces, such as industries and buildings.

2.1.1. Localization techniques

There exist plenty of different strategies to retrieve position measurements, like Two Way Ranging (TWR), which relies on Time of Flight (ToF), Time Difference of Arrival (TDoA), Received Signal Strength (RSS), Angle of Arrival (AoA), and many more.

Two way ranging is also called Time of Arrival (ToA). It relies on measuring the time taken by the signal to travel from the transmitter to the receiver. This value is then multiplied by the speed of light $c = 3 \times 10^8$ m/s to obtain the distance. To achieve that, exact synchronization between units is necessary. For very short lengths, performances can degenerate, since the time of flight can be really small, especially for radio frequency technologies. Therefore, timers with great precision and synchronization accuracy are mandatory.

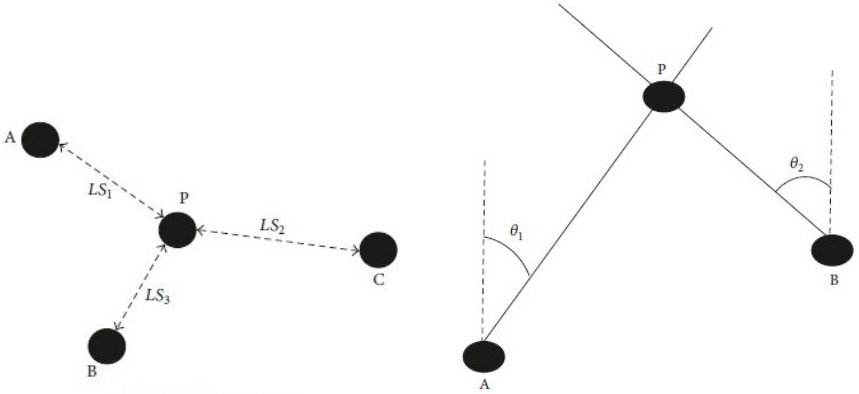


(a) (b)

Figure 2.2: (a) ToA and (b) TDoA methods

Time difference of arrival requires the definition of units used as anchors, whose position must be fixed, in charge of detecting and locating a transmitting device. The transmitter broadcasts signals at regular intervals. The latter are detected by the receivers, which can measure the time of arrival; the difference in travel times from each anchor is used to estimate the distance to each of them. The computation of the time difference eliminates the need to know the time of transmission, unlike in TWR, so the transmitter is not required to be in sync with the receiver. Synchronization is only required between all anchors.

Received signal strength method uses measurements of the signal field intensity at the receiving point. The distance of the transmitting unit can be estimated using signal propagation models. RSS is a completely asynchronous method, but does not always provide measurements that are sufficiently accurate to properly determine the location.



(a) (b)

Figure 2.3: (a) RSS and (b) AoA methods

Angle of arrival relies on angles instead of distances. In fact, using specific technologies, measurement of angles with which signals are received can be obtained. The receiver defines a line that departs from its position with such angle, where the target object is assumed to be. The combination of several lines from several reference devices places the target object at the intersection of these trajectories, so positioning is achieved; at least two reference points and two angles are needed. This method does not need synchronization, but complex and expensive hardware is required, it can be used in addition to other strategies to provide better performances.

Some of the previously cited techniques must be used together with the multilateration method, which combines the range measurements coming from the devices to locate the position of the desired tag. At least distance measurements from three 'known' points are required, in order to solve a trilateration problem and locate without ambiguity the tag location in 3D space.

$$
\begin{aligned}
(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 &= d_1^2 \\
(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 &= d_2^2 \\
(x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 &= d_3^2
\end{aligned}
$$

However, extending the number of available distance measurements is suggested, in order to guarantee feasibility of the localization problem also in case of malfunctioning of some sensors.

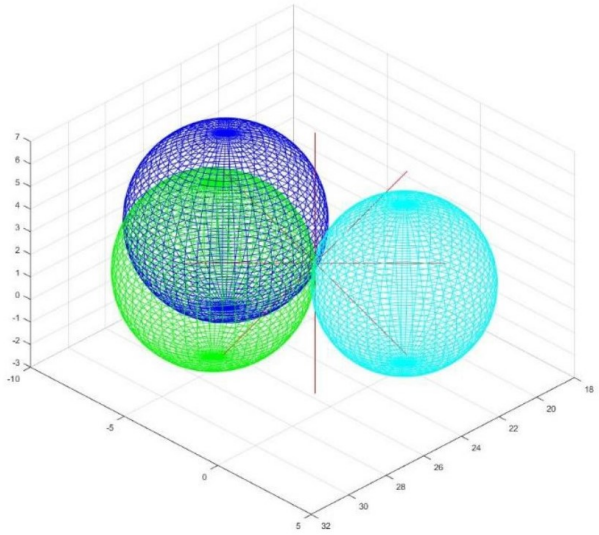


Figure 2.4: Trilateration.

Since measurements are affected by noise coming from multipath, scattering of the sig-

nal, no-Line-of-Sight (nLoS) condition and synchronization shifts, it is not possible to determine a single point as solution. Instead, a region taken as a sphere of radius ζ is considered as the area containing the correct position.

2.1.2. Localization technologies

The Global Positioning System (GPS) is nowadays a mature technology, capable to locate positions with an accuracy < 3 meters in Line-of-Sight (LoS) conditions. However, due to the attenuation of GPS signals as they cross through walls, in indoor environments the accuracy drops to < 50 meters. For this reason, different methods and technologies have been developed to achieve indoor localization, under the name of Indoor Positioning Systems (IPS).

A large selection of competitive solutions is available on the market, for example based on Radio Frequency (RF), like WI-Fi (WPS), Bluetooth, Zigbee, RFID and UWB; technologies which relies on visible and infrared light (IR), based on sound, both audible and ultrasonic, or related to magnetic fields and many more.

In the survey [2] a complete comparison is presented, where pros and cons of each technology are deeply discussed.
Vision appliances like cameras are the ones providing the best performances in terms of positioning precision, with an accuracy < 1 cm. However, for this work such technology is not taken into consideration, since it necessitate of a complex structure to place the cameras in the correct position. This is in contrast with the portability objectives of the desired experimental setup.

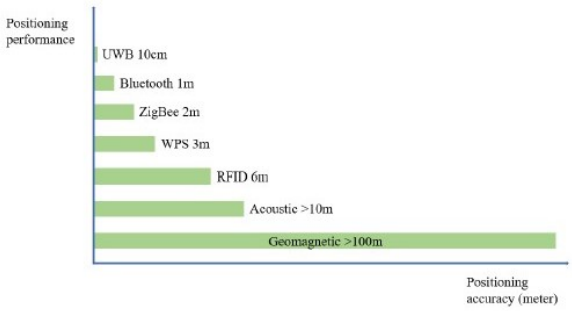


Figure 2.5: Accuracy of positioning technologies.

Among the other available solutions, the ones based on radio frequency can provide good performances in terms of precision and, thanks to the strong penetrating ability of such high working frequencies, they can better withstand nLoS conditions and transmit also trough walls and buildings.

In the RF field, WPS positioning is one of the most wide spread technology for indoor localization. Using the method of fingerprinting, where a pattern of known Wi-Fi bases with their relative strengths is matched to a database of known patterns associated with locations; distances are obtained by comparison with current data. Such technology has a low update rate and an accuracy < 10 meters, which is not fitted for our application.

Also Bluetooth technology is widespread and is considered a competitor to WI-FI. The adoption of Bluetooth Low Energy (BLE), due to its availability, low cost, and very low power consumption, allows fixed emitters to run on batteries for incredibly long time. It relies on received signal strength indication like WPS and, for this reason, is more likely to experience signal interference. Additionally, its working frequencies are lower compared to other concurrent and so it has reduced penetrating ability. This solution can provide accuracy < 1 meter and latency in the order of seconds between consecutive measurements, that are not suited for our purposes.

The Ultra-Wide-Band (UWB) technology is a highly accurate indoor positioning system, with high transmission rate, low transmit power and strong penetrating ability. In fact, UWB operates with a high bandwidth over a very wide frequency spectrum between 3.1 to 10.6 GHz. Its distance-based measurement is performed via time-of-flight. Location is computed based on how long it takes for signals to travel from one device to another, that, compared to the RSS method adopted by similar technologies, is much more robust to experience signal interference. This solution can provide an incredibly high accuracy (10-50 cm), with a really small latency, that makes it the best option for application where precision and high update rates are required. In fact, with the reduction of UWB chip cost in recent years, it has become the first choice for indoor high-precision positioning applications. These features perfectly fits our needs, for this reason we selected this technology to realize the real time positioning system.

	UWB	Chirp (CSS)	BLE	Wi-Fi
Location Accuracy*	10-50 cm	1-2 m	< 5 m	< 10 m
Range*	Optimal: 0-50 m Up to 200 m	Optimal: 10-500 m Up to 1000 m	Optimal: 0-25 m Up to 100 m	Optimal: 0-50 m Up to 500m
Latency*	< 1 ms to get location	< 1 ms to get location	Typically 3-5 s to get location	Typically 3-5 s to get location
Power Consumption	Low, option for embedded cell battery in select hardware options	Very low, option for embedded cell battery in select hardware options	Very low, option for embedded cell battery in select hardware options	Moderate
Cost	$$	$	$$	$$$ (Low $ with existing Wi-Fi access points)
Frequencies	3.1 – 10.6 GHz	ISM-band 2.4 GHz (2.4-2.4835)	2.4 GHz	2.4, 5 GHz
Data Rate	Up to 27 Mbps	Up to 2 Mbps	Up to 2 Mbps	Up to 1 Gbps

Figure 2.6: UWB vs other positioning technologies.

2.1.3. The adopted localization system

Nowadays, several devices available on the market are equipped with UWB technology, like Ubisense, BeSpoon, Decawave and many others. The study [9] provides a comparison among the principal competitors in a nLoS environment similar to an industrial warehouse. As a conclusion, Decawave systems give slightly better performances than BeSpoon systems and are significantly more reliable, in terms of accuracy and outlier content, than the Ubisense systems. Decawave sensors have found widespread use within the scientific community, as a large number of published works use them in their experimental tests. For these reasons we decided to use the MDEK1001 Development Kit from Decawave, which allows to set up a scalable real-time localization system (RTLS) using UWB transmissions.

Decawave UWB evaluation kit (MDEK1001) is described by the manufacturer as an "out-of-the-box scalable RTLS network solution". It consists of twelve DWM1001-DEV modules, each of which can be configured as an anchor or a tag, but also more advanced modes are supported that works as bridge to route data between the UWB network and IP network (gateway) or a tag (listener), although they are not used in this work.

Figure 2.7: DWM 1001-DEV module.

Devices configured as anchors are intended to be placed in fixed positions in the environment, and the nodes configured as tags are used on the mobile devices whose positions must be detected. A tag has the ability to measure distances with respect to various anchors; locally, each tag can calculate independently its own location, solving a trilateration problem using the received distances. Each mobile unit needs to know ranges with at least three anchors, to guarantee unambiguous positioning. The module incorporates Decawave DW1000 UWB transceiver which is used to perform the two-way ranging exchanges among network nodes, enabling the distance computation. The TWR strategy requires precise synchronization among units so, before each TWR transmission, the DWM1001-DEV unit performs an estimation of the clock drift with respect to the network time, fixed by the initiator, in order to keep the correct synchronization among the whole network and provide satisfactory performances.

The relative large number of available units allows to set up the network in a multitude of different ways, changing the number of anchors and tags to better fit the user needs. The position of each tag can be access by direct connection with the desired unit, using the USB or BLE connectivity.
The typical configuration for using the localization system is shown in Figure 2.8, where there are eight tags and four anchors. In this way the maximum number of mobile devices can be located; variation with more anchors can be made, in order to extend the measuring area.

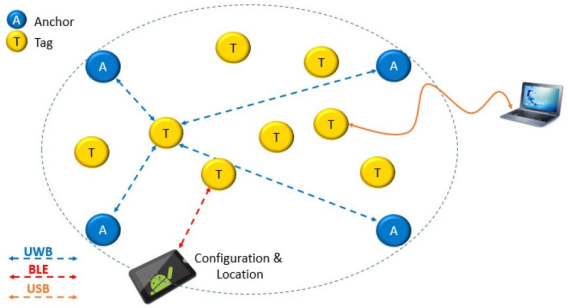


Figure 2.8: 8 tags and 4 anchors configuration.

The DWM1001-DEV unit also allows the user to set up a node as a listener, see Figure 2.9. This advanced mode permits to gain access from a central point to a complete view of all tags within a network. A listener is a tag with the UWB radio in passive mode (the UWB link is used for communication but not for range measurement), and connected to a computer via USB. This is really helpful when the number of tags present in the network is large. A limitation of this access mode is that it can be used to detect the positions of all tags but not their respective measured ranges.

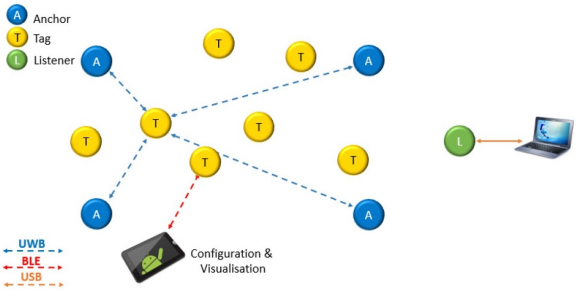


Figure 2.9: 7 tags, 4 anchors and 1 listener configuration.

Another available configuration allows to use nodes as gateways, see Figure 2.10, to route UWB network data over a LAN/WAN network. This can be used to extend the network by deploying anchors over longer distances (with no upper limitation) in order to cover larger buildings.

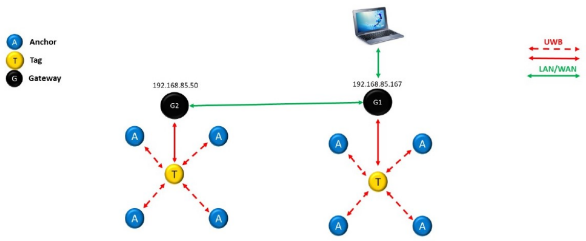


Figure 2.10: 2 tags, 8 anchors and 2 gateway configuration.

In this work, configurations which consider the presence of listener and gateways are discarded, due to the small size of the setup. Positioning information of tags are obtained by directly accessing each device periodically. The selected working area, better described in Section 2.2, is a square of 2x2 meters. Four anchors are placed at the corners of the structure to guarantee correct localization, while a tag is attached on top of each mobile robot. The complete setup is composed by 6 DWM1001-DEV modules, 4 used as anchors and 2 used as tags.

The decawave real time localization system requires that a set of tags and anchors must be associated with the same network (unique PANID) in order to be able to obtain location information for all of them. One of the anchors in the network must be configured as initiator, which is in charge of keeping the timing and synchronization of all modules in the network. The communication and measurement bandwidth of the whole localization network system is 150 Hz, as defined in the PANS (Positioning and Networking Stack) protocol. This means that 750 tags can be located at a rate of 0.2 Hz sampling frequency, 150 tags at 1 Hz, or 15 tags at 10 Hz (maximum sampling frequency). The maximum sampling period is one minute, which would theoretically allow for work with 9000 tags. One hardware limitation set up by the manufacturer is that the master node (initiator) can not see more than 30 anchors simultaneously: this does not mean that there can only be 30 anchors in a network. It is possible to include more anchors if a new initiator joins the network and is placed far enough away and outside the UWB range of the first one. The restriction is related to the fact that each master anchor can only see a maximum of 29 regular anchors.

The PANS protocol by Decawave is a software that defines how the interactions within a DWM1001 node network are orchestrated in order to measure ranges between nodes and communicate information over the UWB channel. The Decawave RTLS network uses time division multiple access (TDMA) channel access, at each instant of time, there is only one tag-anchor pair exchanging messages to measure the round-trip time (TWR). Nodes operate using a 100 ms long “superframe”, whose structure is shown in Figure 2.11. This

TDMA synchronization method is very good to avoid collision, it guarantees a predictable scalability of the system to accommodate hundreds of tags.

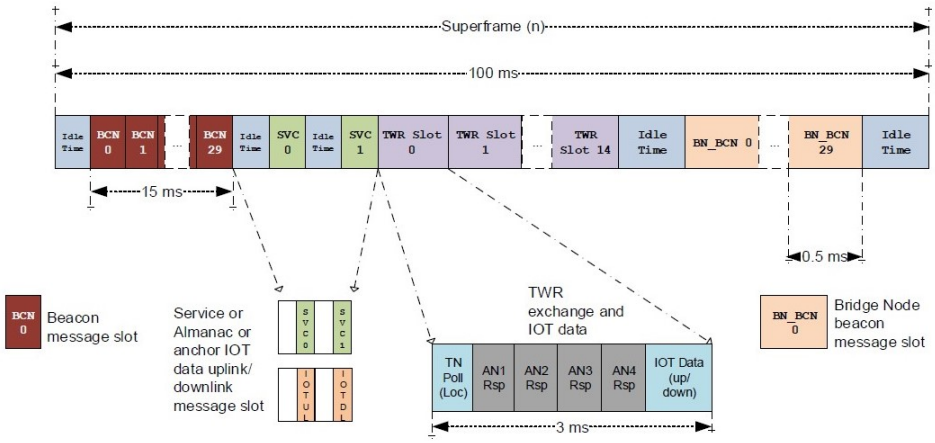


Figure 2.11: Superframe structure.

In each superframe there are 29 BCN slots, allocated for newer anchors to join the network, while the slot BCN0 is reserved by the initiator, who initializes the network and keeps the clocks of all the nodes synchronized. The 15 TWR slots are used to measure the anchor-to-tag ranges. Each of them lasts for three milliseconds: in that interval, measurements of ranges up to a maximum of 4 anchors are performed.
If at least three distances from anchors to a particular tag are available, then the tag position can be privately computed by its microcontroller through the location engine unit, which uses a trilateration algorithm extended with maximum likelihood estimation method and a fixed moving average of the last 3 location results.
The location engine calculates the errors between the estimated positions and the real distance and removes the positions which have high errors. It will also report a quality factor (0-100) together with the computed coordinates.

It is important to highlight that the PANS protocol allows for the coexistence of several networks operating in the same space. In this case, each network has a different identifier (PANID) and the initiating anchor will cooperate to share the time slots and not interfere with each other.

The DWM1001 module firmware provides the Application Programming Interface (API) [4] with which a user can gain access to UWB nodes. There are four APIs available:

- C API, which allows a user to directly program in C language on the microprocessor of the DWM1001-DEV module, and adds custom code that makes use of the existing

API functions

- SPI API, which allows to access the module from an external host device used in SPI master mode, the maximum SPI clock frequency is 8 MHz. It uses a Type-Length-Value (TLV) format
- UART API, designed to communicate through a serial COM port using the UART interface with a baud rate 115200 bps. It has two operation modes: "Generic" (TLV format) and "Shell" mode that generates a terminal like prompt (dwm>) where module's information can be accessed
- BLE API, the wireless mode using Bluetooth Low Energy 4.0 (BLE) that allows for discovering the UWB modules that make periodic BLE broadcasts. Module information can be accessed trough their BLE characteristics (according to protocol "Generic Attribute" - GATT of BLE), which are quite limited respect to other API functions, e.g. accelerometer measurements are not accessible.

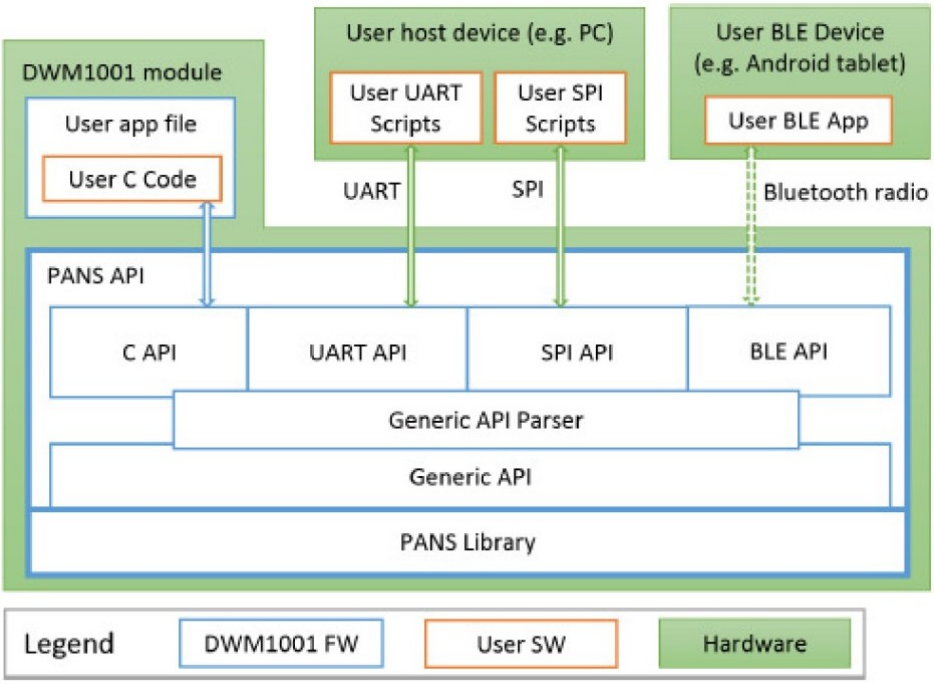


Figure 2.12: Decawave APIs.

In this work, since we are interested just in receiving the information about the position of each tag, we decided to use the BLE API, which is the one that better fits our goals of versatility for the experimental setup. In fact, it does not require any physical connection between tags and computer.

2.1.4. Reference localization system for calibration

As seen in [7], the sensors described in Section 2.1 are influenced by noise coming from the external environment, such as reflections, electromagnetic field, light, and many others. One of the objectives of this thesis will be to characterize the measurement noise affecting the UWB sensors and to develop a strategy to compensate for the biases related to the chosen positioning. To do the calibration, a high precision optical tracking system will be used as reference, the OptiTrack Motion Capture System, with a sub-millimeter positioning accuracy. This device uses reflective tags and a set of high resolution cameras to locate objects in the established area.

OptiTrack is a real time tracking system based on infrared (IR) cameras, offering high precision, low latency, 6 Degrees of Freedom (DoF) tracking for ground and aerial robotics. Manufacturers guarantee position errors smaller than 0.3 mm and rotational errors smaller than 0.05° in optimal conditions, with up to 360 fps acquisition speed. It consists of a set of cameras arranged around a selected volume, and a set of reflective markers to be placed on the devices whose position and orientation must be detected. The OptiTrack system located in the laboratory is equipped with twelve S250e cameras, with a resolution of 832x832 pixels and guaranteeing a maximum sampling frequency of 250 fps.

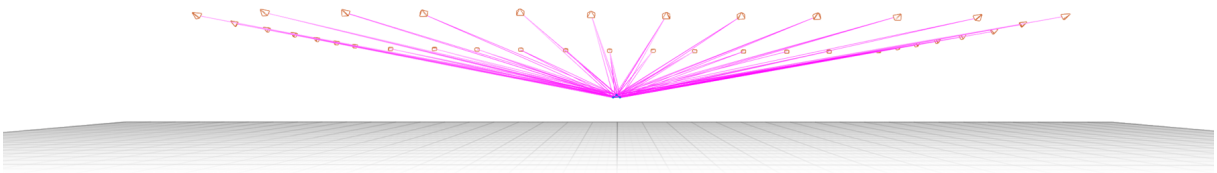


Figure 2.13: OptiTrack setup.

Each camera captures 2D images. Thus, 2D positions are calculated by Motive (proprietary software) and the overlapping position data are compared to compute the 3D positions of each tag via triangulation. The software needs, at least, 3 markers in order to create a valid rigid body, but it is always convenient to put some redundant marker (4 or 5), so that if one (or more) should be lost or not exposed, the tracking would be still valid anyway. The markers must be put together in asymmetric configurations, to allow for the detection of the orientation (roll-pitch-yaw angles) besides of the position (x,y,z). Markers are made of a particular reflective tape that reflects IR rays and make the tracking possible.

Figure 2.14: OptiTrack markers.

2.2. The arena

The choice of the positioning sensors gives us some freedom for the selection of the working area. In fact, the Decawave positioning system can be easily extended to cover large distances. However, for the sake of portability, we limited the region of motion to be a square of 2x2 meters, according to the minimum requirements of spacing among each UWB unit, as stated in [7]. The working area was confined inside a structure of wood, joint together using hinges. In this way it can be easily taken apart and re-assembled at occurrence.

The frame was equipped with 90° clamps, in order to keep the correct alignment between its sides. We positioned, at the vertices of the structure, 4 pillars, in order to place the ranging sensors to work as anchors. The latter must be placed slightly above the units used as tags, since in this way the LoS condition is always granted and performances increase, as specified in previous work [7].

2.3. E-PUCK robots

The carts used in our setup are unicycle mobile robots, whose dynamics is suitably linearized. Among the large variety of mobile robots commercially available, we have selected the E-PUCK robots, developed at the Swiss Federal Institute of Technology in Lausanne (EPFL) and commercially distributed by GCtronic.

Figure 2.15: E-PUCK robot.

The desktop-size and lightweight E-PUCK robots were developed for educational purposes, equipped with a large number of sensors and actuators, and endowed with communication capabilities. They are developed based on an open hardware concept, where all documents are distributed and submitted to a license allowing everybody to freely use them and contribute to the project.
The competitive cost makes it the best solution for projects that require a large number of agents and, thanks to the large amount of tested units, it guarantees a good robustness to wear.

The mechanical structure of the E-PUCK robots is simple: they are differential drive (i.e., unicycle) vehicles, where the wheels, with a diameter of 21 mm, are actuated by two independent stepper motors. The third contact point is directly on the robot case, but it is irrelevant in terms of friction contribution, so it will be neglected in the development of the mathematical model. The robot has a diameter of 75 mm, a height of 55 mm and a weight of 150 g. Most of the weight is due to the 5 Wh LION rechargeable and removable battery, providing 3 hours of autonomy. It is equipped with two stepper motors with a 50:1 reduction gear, resolution of 0.13 mm that can provide a maximum rotational speed of 1000 step/s, corresponding to a complete turn of the wheel per second, while the maximum linear speed is about 80 mm/s.

The processor is a dsPIC30F6014A running at @64 MHz, a CPU of the Microchip family with a 16-bit architecture, providing 16 MIPS of peak processing power. It is equipped with 8 KB of RAM memory and 144 KB of flash memory. It has a 12 bit Analog-to-Digital

converter (ADC), with 12 channels working at a maximum sampling frequency of 200 ksps. This device was developed in 2005 and all the electronics is rather outdated, in view of which the computational power is significantly limited compared with recent processors. In view of this, since the control strategy implemented in this thesis is based on model predictive control, the hardware is not capable to handle such heavy computations, so the control actions were computed by a host computer and then sent to the single entities, where the electronics just manages to actuate the motors and receive measurements. The robot is provided of plenty of different modules to achieve connectivity, like I2C, CAN, and also UART. One of the two available UART modules is connected to the LMX9820A chip, which is a Bluetooth device allowing for wireless communication. This feature is really helpful since it allows for data transmission between computer and robot and also between robots. In fact, each device can be connected with up to 7 other E-PUCK robots, allowing for a fundamental feature of distributed systems, where agents must communicate together.

Figure 2.16 shows all the sensors and actuators available on the robot, together with the connectivity ports, reset button, mode selector, and the multitude of leds distributed all around the robot body, showing the status.

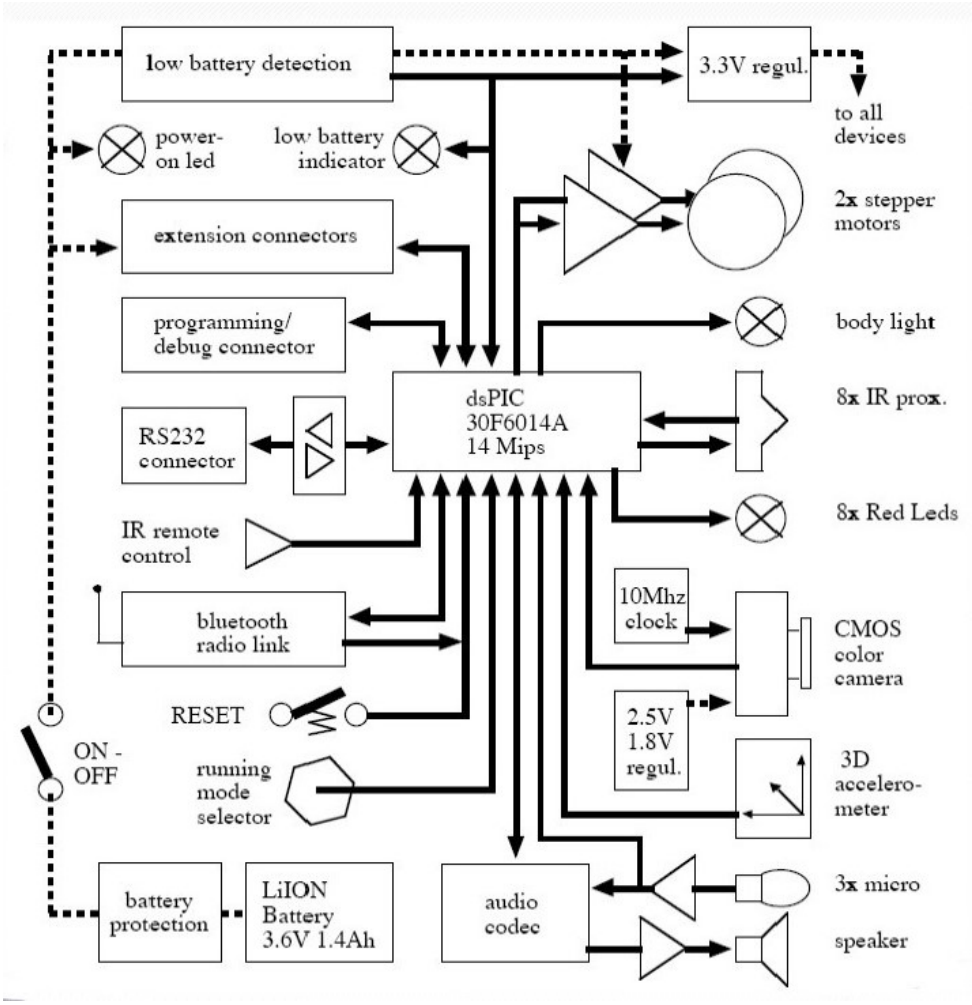


Figure 2.16: E-PUCK connectivity, sensors and actuators.

The robot is equipped with 8 infrared proximity sensors, a 3D accelerometer, 3 omni-directional microphones and a VGA color CMOS camera with 640x480 pixel resolution. It has 2 stepper motors, a speaker, a classic RS232 connector and a connector to interface it with an in-circuit debugger to program the flash memory and debug the code. It is provided with a Bluetooth radio link, and an infrared remote control receiver. The developers equipped the robot with a Bootloader, that allows the user to program the flash of the microcontroller trough the Bluetooth or the serial port, this makes the programming possible without requiring any external device.

2.3.1. Kinematic model

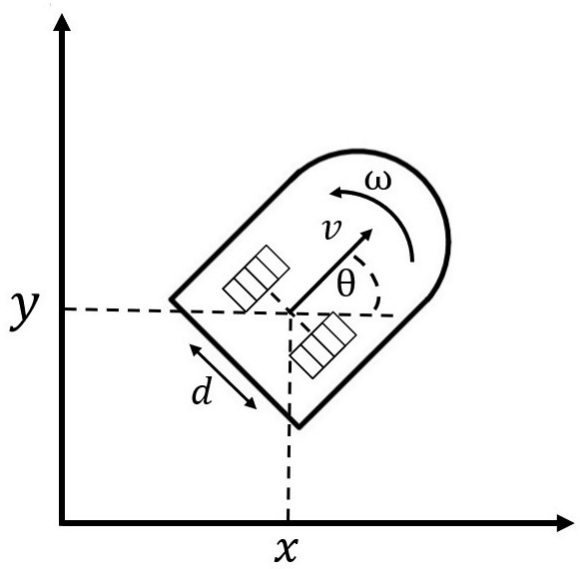


Figure 2.17: E-PUCK kinematic model.

The unicycle model of the robot can be developed considering two contact points of the wheels while neglecting the third, required only for stability but whose friction contribution can be discarded.

With reference to Figure 2.17, the kinematic model of the unicycle is

$$\begin{cases} \dot{x} = v\cos\vartheta \\ \dot{y} = v\sin\vartheta \\ \dot{\vartheta} = \omega \end{cases}$$

Considering that in a differential drive vehicle the two wheels are actuated by different motors, the rotational speed of the right and the left wheels (ω_R and ω_L, respectively) can be different one from the other. Through simple linear transformations, it is possible to compute the linear and angular velocities of the robot from ω_R and ω_L. More specifically, defining with R the radius of the wheels, the tangential speeds v_R and v_L of the two wheels can be computed as

$$\begin{cases} v_R = \omega_R R \\ v_L = \omega_L R \end{cases}$$

$$\begin{cases} v = \dfrac{v_R + v_L}{2} & = R\dfrac{\omega_R + \omega_L}{2} \\ \omega = \dfrac{v_R - v_L}{d} & = R\dfrac{\omega_R - \omega_L}{d} \end{cases}$$

In this thesis we decided to use just the kinematic model, while the dynamics regarding the forces affecting the system are discarded. For example, the electro-mechanical dynamics related to the actuators, like voltage-torque relationships of the stepper motors are not considered.

A rigid body in planar motion has 3 degrees of freedom, the motion along the X,Y axes and the rotation along the Z axis. The kinematics of the robot constrains certain motions: it can't move parallelly along the axle connecting the wheels, because it is subject to the constraint

$$\dot{x}\sin\vartheta - \dot{y}\cos\vartheta = 0 \tag{2.1}$$

The constraint in equation (2.1) is said to be non-holonomic. Since it is non-integrable, it does not reduce the degree of freedom of the system, so any configuration is accessible in $\mathbb{R}^3$. The robot can achieve any movement in that space but is constrained to move strictly forward or to rotate. Any configuration can be reached as a composition of these two primitive movements.

2.3.2. Feedback linearization

The model described in the previous section is nonlinear and, since these nonlinearities significantly affect the dynamics of the system, it is important to find a way to linearize the model without losing information. This is particularly important because predictive control optimization problems can be formulated in a simpler way provided that the model is linear. Therefore, we decided to apply the feedback linearization technique. This approach, by using an internal loop, is capable to reduce the system to a model represented by a set of simple linear integrators.

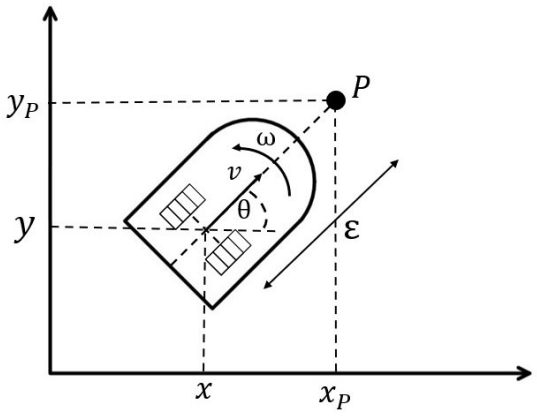


Figure 2.18: Feedback linearization model.

The applied feedback linearization is non-singular and relies on the choice of a point P along the centre line of the robot chassis as shown in Figure 2.18, located at arbitrary distance ε from the axle connecting the wheels. The position of the point P can be described as

$$\begin{cases} x_P = x + \varepsilon \cos\vartheta \\ y_P = y + \varepsilon \sin\vartheta \end{cases}$$

Differentiating with respect to time

$$\begin{cases} \dot{x}_P = \dot{x} - \varepsilon\dot{\vartheta}\sin\vartheta = v\cos\vartheta - \varepsilon\dot{\vartheta}\sin\vartheta & = v_{x_P} \\ \dot{y}_P = \dot{y} + \varepsilon\dot{\vartheta}\cos\vartheta = v\sin\vartheta + \varepsilon\dot{\vartheta}\cos\vartheta & = v_{y_P} \end{cases} \tag{2.2}$$

By opportunely manipulate equation (2.2), the following change of coordinates can be obtained

$$\begin{cases} v = v_{x_P}\cos\vartheta + v_{y_P}\sin\vartheta \\ \omega = \dfrac{v_{y_P}\cos\vartheta - v_{x_P}\sin\vartheta}{\varepsilon} \end{cases} \tag{2.3}$$

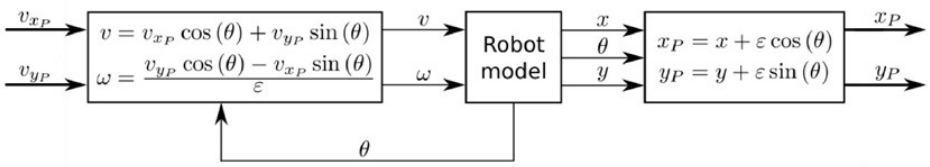


Figure 2.19: Feedback linearization scheme.

Thus, the model can be seen as a pair of decoupled simple linear integrators

$$\begin{cases} \dot{x}_P = v_{x_P} \\ \dot{y}_P = v_{y_P} \end{cases}$$

This technique allows to obtain, from an external point-of-view, a linear mathematical model, more suitable to develop an MPC-based regulator, avoiding to resort to the non-linear theory. The system describes the position of the point P and differs from others feedback linearization methods where the position of the centre of the robot is represented, but this allows the model to be non-singular for null velocities.

This model now can be represented in state-space form, by selecting the X,Y positions of point P as states and the velocities v_{x_P} and v_{y_P} as inputs.

$$\begin{cases} \begin{bmatrix} \dot{x}_P \\ \dot{y}_P \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_P \\ y_P \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_{x_P} \\ v_{y_P} \end{bmatrix} \\ \begin{bmatrix} x_P \\ y_P \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_P \\ y_P \end{bmatrix} \end{cases} \tag{2.4}$$

The model is discretized assuming that inputs are applied in a sample-and-hold fashion, leading to the discretized model

$$\begin{cases} \begin{bmatrix} x_P(k+1) \\ y_P(k+1) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_P(k) \\ y_P(k) \end{bmatrix} + \begin{bmatrix} \tau & 0 \\ 0 & \tau \end{bmatrix} \begin{bmatrix} v_{x_P}(k) \\ v_{y_P}(k) \end{bmatrix} \\ \begin{bmatrix} x_P(k) \\ y_P(k) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_P(k) \\ y_P(k) \end{bmatrix} \end{cases} \tag{2.5}$$

where τ is the sampling time.

2.4. Computing hardware and software

As discussed in Section 2.3, the electronics with which the E-PUCK robots are equipped is not capable to provide enough computational power to run a regulator based on model predictive control. For this reason, the use of a host PC, in charge of running the controllers of all the agents, is necessary. The host PC is connected using the Bluetooth radio link, to all the E-PUCK robots in the setup, and to the Decawave tag modules placed on all agents. The Bluetooth connection is performed using a baud rate of 115200 bps. Once the control actions are computed by the regulator, they are sent back to the robots, that should actuate the motors and achieve the desired positioning.

The Distributed model Predictive Controller (DPC) was implemented using MATLAB, which is the ideal environment to implement such complex control algorithm, providing toolboxes to deal with the Bluetooth and serial communication, performing optimization and operating with sets.

The optimization algorithm is solved using the MATLAB toolbox Yalmip [10], which is a free-to-use software designed to formulate and solve optimization problems. The generation of the Robust Positively Invariant sets (RPI), necessary for the realization of the DPC controller, which will be described in Chapter 4, requires operations among sets, like Minkowsky sum, Pontryagin difference and many more. That is achieved using the Multi-Parametric Toolbox 3 (MPT3) [8]. The latter is an open source, MATLAB-based toolbox for parametric optimization, computational geometry and model predictive control.

The E-PUCK robots are programmed using the property software MPLABX IDE, which allows to write code both in C and assembler languages. The robot manufacturer provides a list of libraries to handle all the sensors and actuators on the device. In Appendix A the C code running on the E-PUCK robots is reported.

3 | Characterization of delays and disturbances in the experimental setup

In this chapter the characterization of delays and disturbances affecting the experimental setup is addressed. Consider the discrete-time unicycle model of the E-PUCK robot introduced in equation (2.5). The states $x_P(k)$ and $y_P(k)$ are both measurable, using the measurements coming from the Decawave positioning system although with some noise *v(k)*. In the real setup, also the dynamics of the system are subject to a disturbance *w(k)*, due to the minor phenomena neglected while developing the mathematical model. The state-space model, including these terms, can be written as

$$\begin{cases} \begin{bmatrix} x_P(k+1) \\ y_P(k+1) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_P(k) \\ y_P(k) \end{bmatrix} + \begin{bmatrix} \tau & 0 \\ 0 & \tau \end{bmatrix} \begin{bmatrix} v_{x_P}(k) \\ v_{y_P}(k) \end{bmatrix} + w(k) \\ \begin{bmatrix} x_P(k) \\ y_P(k) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_P(k) \\ y_P(k) \end{bmatrix} + v(k) \end{cases} \tag{3.1}$$

where the properties of *w(k)* and *v(k)* will be later characterized.

Firstly, a deep analysis of the UWB sensors is necessary. In fact, it is known from past works [7] that the measures provided by those devices are affected by biases, and one of the objectives of this thesis is devise a calibration strategy to compensate for them. Moreover, a characterization of the measurement delays is required since, as we will discuss in the following, most of the lag affecting the system comes from the positioning sensors, while the delays related to Bluetooth transmission, processing, and actuation are negligible.

To characterize the Decawave sensors, the OptiTrack system introduced in Section 2.1.4 was used, mainly as benchmark platform to provide correct measurements, since its accuracy is far better than the one of the adopted positioning system.
We used also the three axis accelerometer MMA7260QT [16] provided by NXP Semicon-

ductor, equipped on the E-PUCK robots, with a sampling frequency of 11 kHz. It was used to estimate the delay affecting the positioning measure, since its update frequency is much larger than the one of the UWB sensor, which is 10 Hz. Therefore, the delay can be estimated by comparison between data coming from the UWB sensors and the changes in the acceleration data coming from the accelerometer.

3.1. Decawave positioning system characterization

In this section a complete characterization of the Decawave UWB sensors is devised. It consists in the evaluation of the delays and the disturbance *v(k)* in equation (3.4) introduced by the positioning system. Then, a strategy to compensate for the bias affecting the measurements is introduced.

3.1.1. Delay characterization

The selected strategy to estimate the measurement delays is based on the use of data obtained from the E-PUCK accelerometer. The mobile robots can be connected through Bluetooth radio link with the PC; in this way information coming from all the sensors arranged on it can be retrieved. Regarding the accelerometer, the robot manufacturer provides a specific library to handle it, including a set of functions to send the measured accelerations to MATLAB using the Bluetooth connectivity.

As stated above, the sampling frequency of the accelerometer is 11 kHz, which is significantly faster than the update rate Decawave sensors (10 Hz), and so it is ideal for our purposes. Unfortunately, the Bluetooth transmission isn't that fast and acts as bottleneck.

The delay affecting the transmission was evaluated by measuring the time interval between two consecutive acceleration readings in the MATLAB environment. This is justified by the fact that the code running on the E-PUCK robots was set up to send the accelerations measurements as soon as they were available, so an update rate of 11 kHz was expected. This was however not possible due to the transmission lag. Therefore the frequency with which accelerations readings were obtained in the MATLAB environment provides an estimation of the delays affecting the Bluetooth connectivity. This measure was averaged among multiple readings, specifically 500 and 1000; the same tests were repeated multiple times, to obtain more reliable data. In particular, regarding the Bluetooth connection, the average delay is 10.33 ms, with a standard deviation of 6.11 ms.

This delay estimation was validated using an alternative strategy. This technique consists of evaluating the time that it takes for data to be sent from the PC, received by the robot,

sent back and then received by the PC. The latter is really similar to the time-of-flight technique introduced in Chapter 2 to achieve ranging measurements. The timing was managed by MATLAB, which started a timer when data were transmitted for the first time and then arrested it when data were received back. This delay estimation considers also the time taken by the microcontroller equipped on the E-PUCK robots to process the data and then send them back. However, since the microcontroller works at high frequency and the specific operation complexity was small, we assumed this time to be negligible compared to the one related to data transmission. The measured time was averaged among multiple tests and divided by 2, since it measured the time taken for two complete Bluetooth transmissions. Results show a delay estimation compatible with the one obtained with the first strategy. More specifically, in this way we estimated an average delay of 15.19 ms, with a standard deviation of 8.65 ms.

In view of this result, we opted to set, for the estimation of the measurement delay, an accelerometer reading frequency of 100 Hz. As shown in Figure 3.1, the acceleration readings at 100 Hz (above), and the position measurements at 10 Hz (below) were compared.

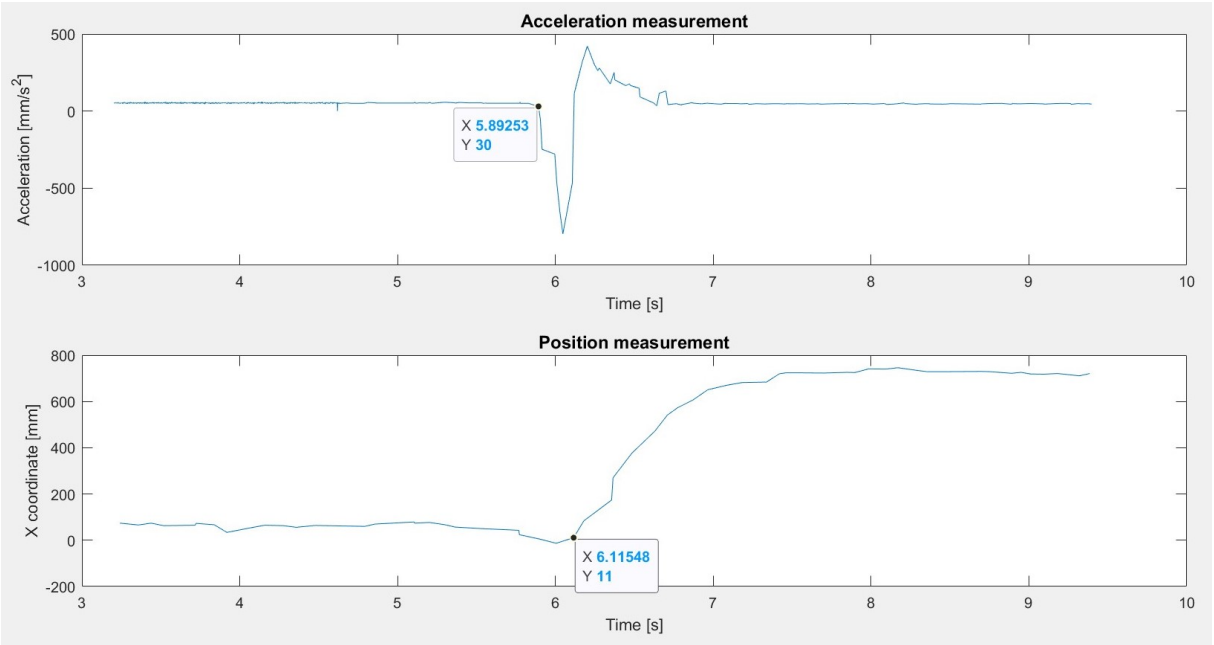


Figure 3.1: Acceleration and position measurements.

The initial time was selected as the time instant in which the acceleration had a sudden change, and the delay was computed by selecting the time instant in which also the position measurement detected a variation. This test was repeated 50 times and obtained data are shown in Table 3.1.

test	delay ms	test	delay ms	test	delay ms	test	delay ms	test	delay ms
1	20	11	23	21	20	31	24	41	24
2	15	12	28	22	30	32	15	42	22
3	23	13	22	23	19	33	25	43	26
4	22	14	19	24	17	34	23	44	18
5	27	15	16	25	15	35	18	45	25
6	25	16	22	26	22	36	15	46	22
7	21	17	17	27	24	37	24	47	26
8	21	18	15	28	16	38	23	48	18
9	11	19	24	29	24	39	24	49	25
10	14	20	16	30	16	40	19	50	16

Table 3.1: Decawave sensors delay data

We obtained an average measurement delay of 206.2 ms, with a standard deviation of 41.4 ms. This delay is far larger than the one related to data transmission, processing, and actuation. Due to the presence of this relatively large delay, also the choice of the sampling time for the system discretization and the design of the controllers was constrained. In particular, it was selected to be $\tau = 0.2$ s.

3.1.2. Disturbance characterization

The position measurements provided by the Decawave positioning system are affected by significant noise and biases. In fact, during preliminary tests, it was discovered that bias and variance of the tags measures completely change with respect to their position in the experimental area, particularly when they are far from the anchors or close to a stronger source of light, e.g., windows.
To correctly characterize this behaviour, data were acquired from several positions, uniformly distributed all over the whole accessible surface. We decided to take 50 sample positions. In this way a complete mapping of the area was obtained, with sample points spaced of $\sim$ 30 cm along both the X and Y axes one form the others.

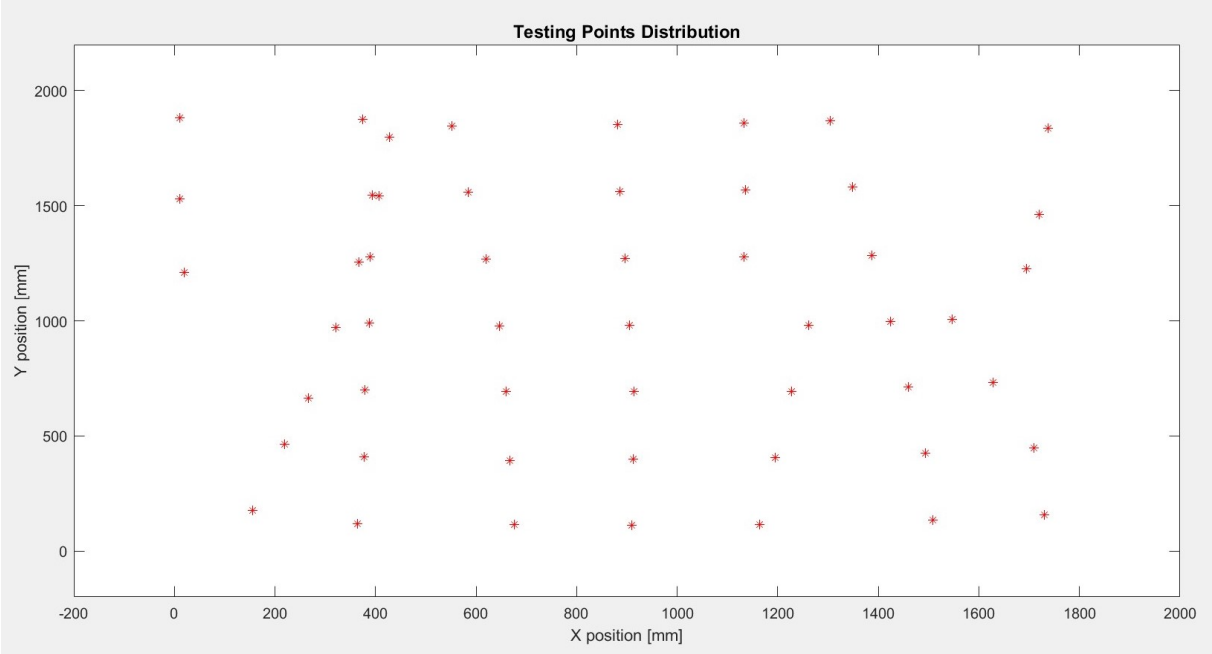


Figure 3.2: Testing points distribution.

For each testing point, a correct tag position measure has been collected using the Opti-Track system.
Positioning measurements from the Decawave sensors were collected for each testing point. We took 500 samples for each position. Recall that, as discussed in Chapter 2, the UWB measurements are equipped with a quality factor ranging from 1 to 100, inducting the estimated quality of the measure. In order to remove outliers we decided to discard data which have a quality factor lower that 30, which can contribute to deteriorate the performances.

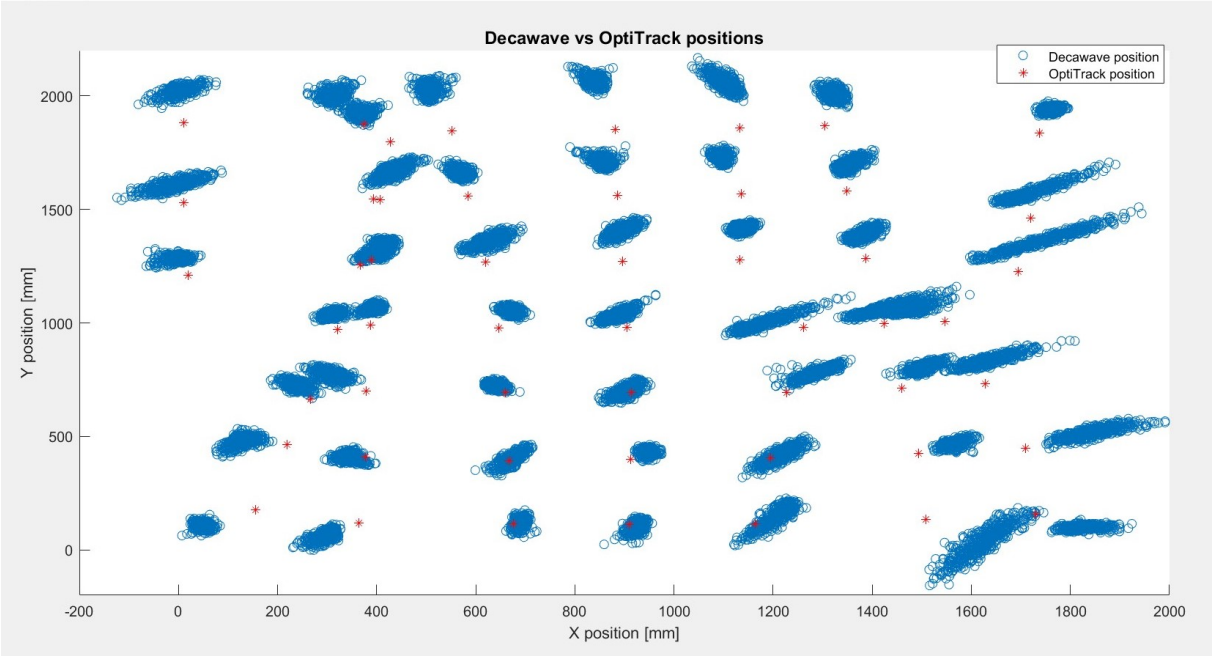


Figure 3.3: Decawave vs OptiTrack positions.

As it can be seen from Figure 3.3, the point clouds of the measurements coming from the UWB sensors are particularly critical. Firstly note that, in the right side of the working area ($1000 < x < 2000$), the measurements deteriorate due to the presence of a window. It can be observed that, for most of the testing positions, the point clouds of the measures are not centered in the correct position, but are non uniformly biased.

As a result, the measurements of the Decawave sensors, without any manipulation, provide unsatisfactory performances. In fact, the average standard deviation among all the tested points was about 54 mm for the X axis and 101 mm for the Y axis. This motivate the need to develop a suitable compensation scheme.

Using a polynomial regression technique, it is possible to remove the biases and reduce the variance of such measures. This consists of finding polynomial functions which are capable to provide a good estimation of the real position $x_{estimated}(k), y_{estimated}(k)$ from the measures provided by the Decawave sensors $x_m(k)$ and $y_m(k)$.

The polynomials are of the type

$$x_{estimated}(k) = C_0 + C_1 x_m(k) + C_2 y_m(k) + C_3 x_m^2(k) + C_4 y_m^2(k) + C_5 x_m(k) y_m(k) + ...$$
$$y_{estimated}(k) = D_0 + D_1 y_m(k) + D_2 x_m(k) + D_3 y_m^2(k) + D_4 x_m^2(k) + D_5 y_m(k) x_m(k) + ... \tag{3.2}$$

We collected 500 samples of the X and Y coordinates for 50 tested points, thus we could define a system composed by (50 x 500)-pairs of equations like the ones in (3.2), that is a over-determined system, whose solution can be approximated thanks to the Least Squares (LS) algorithm.

Considering for example the measurement in the X direction, let ϑ be the parameter vector and let $\varphi(k)$ be the regressor vector

$$\vartheta = \begin{bmatrix} C_0 & C_1 & C_2 & C_3 & C_4 & ... \end{bmatrix}^T$$

$$\varphi(k) = \begin{bmatrix} 1 & x_m(k) & y_m(k) & x_m^2(k) & y_m^2(k) & x_m(k) y_m(k) & ... \end{bmatrix}^T$$

The least squares algorithm consists of minimizing the sum of the squares of the residuals. The cost function considering all the available data can be written as

$$J_x = \sum_{k=1}^{25000} \left\| x_{OptiTrack}(k) - \varphi(k)^T \vartheta \right\|^2 = \left\| X_{OptiTrack} - F^T \vartheta \right\|^2$$

where

$$X_{OptiTrack} = \begin{bmatrix} x_{OptiTrack(1)} & x_{OptiTrack(2)} & x_{OptiTrack(3)} & ... & x_{OptiTrack(N)} \end{bmatrix}^T$$

and

$$F = \begin{bmatrix} \varphi(1) & \varphi(2) & \varphi(3) & ... & \varphi(N) \end{bmatrix}$$

The optimal parameter vector ϑ, minimizing the sum of the residual J_x is

$$\vartheta^{min} = (FF^T) F X_{OptiTrack}$$

Multiple tests were carried out, starting from simpler polynomials e.g. first order, up to far more complex ones.

In data fitting, one of the challenges consists of selecting the model level of complexity

which properly characterizes the existing relationships between dependent and independent variables.
The adopted criterion used to detect a suitable complexity level consisted of checking the evolution the cost function as the complexity increases.
As it can be seen in Figure 3.4, the estimation quality of $x_{estimated}(k)$ and $y_{estimated}(k)$ increases significantly as the complexity raises. We opted to choose the model indicated with 23, which is a good compromise between model complexity and estimation quality.

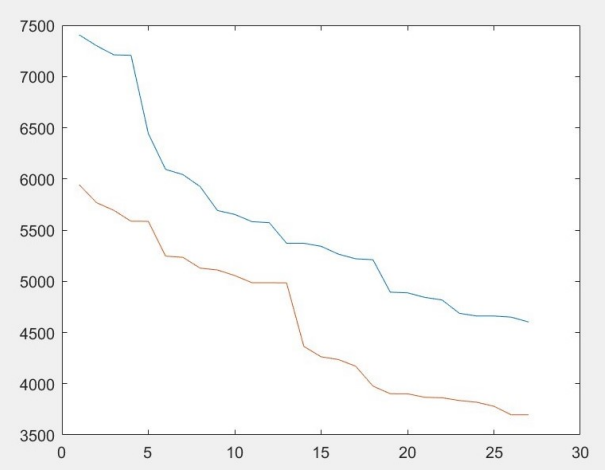


Figure 3.4: Evolution of the cost functions.

The calibration algorithm allows to improve the tracking performances, both in terms of bias, which is completely removed, and in terms of variance, which is significantly reduced with respect to the original measurements shown in Figure 3.3.

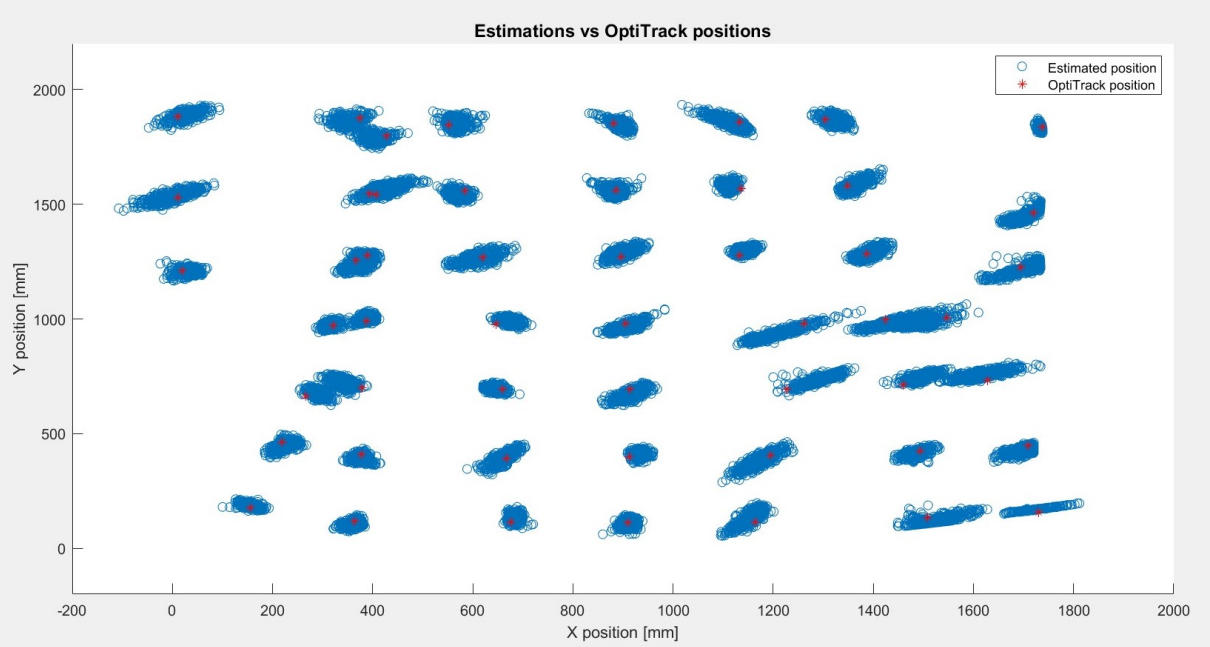


Figure 3.5: Estimated vs OptiTrack positions.

The disturbances affecting the manipulated data can then be characterized, the mean values of the estimations, averaged among 500 samples for each tested position, is close to zero (10^{-7} mm precisely). As it can be seen from Figure 3.5, the point clouds of the manipulated data have a smaller dispersion if compared to the original measurements. The standard deviation is 29 mm along the X axis and 24 mm along the Y axis. It results in a covariance matrix R of the disturbance $v(k)$ equal to

$$R = \begin{bmatrix} 845 & 360 \\ 360 & 566 \end{bmatrix}$$

From tests it was observed that this disturbance $v(k)$ is bounded. It can be confined in the set $\mathbb{V}$, selected as an hyperrectangle which limits are

$$\begin{cases} v_x(k) \in [-95; 90] \\ v_y(k) \in [-75; 80] \end{cases} \tag{3.3}$$

3.2. E-Puck robot disturbance characterization

In this section the procedure to achieve a characterization of the disturbance $w(k)$ acting on the unicycle model of the E-PUCK robot is described, this was possible thanks to the

usage of the OptiTrack system. This disturbance is influenced by many factors, including the following.

- Errors coming from the missing use of a dynamical model, e.g., the electro-mechanical interactions, friction, and other unmodeled dynamics
- Errors related to the wheel sliding phenomena.

For the identification of the disturbance *w(k)*, dynamic tests have been performed. In particular, the mobile robot was required to move at constant speed along different trajectories. Tests were performed in open loop: trough the MATLAB environment, it was possible to send information to the E-PUCK robot, concerning the linear speeds along the X and Y axes that must be actuated, in order to track linear segments.

We decided to perform multiple experiments. In the tests the robot was oriented along the Y direction ($\vartheta = \frac{\pi}{2}$), along the X direction ($\vartheta = 0$), and along the diagonals ($\vartheta = \pm\frac{\pi}{4}$) of the arena. Each test was repeated two times, one with the robot moving forward and then moving backward. The robot was forced to move at constant linear speed of 80 mm/s.

Considering the feedback linearization applied at the E-PUCK robot (equation (2.3)), the inputs that must be given to the robot were the linear velocities of the point P, along the X and Y axis. For the first two experiments where the robot had to move along the Y direction

$$\begin{cases} v_{x_P} = 0 \\ v_{y_P} = 80 \end{cases} \longrightarrow \begin{cases} v = 80 \sin \dfrac{\pi}{2} = 80 \\ \omega = \dfrac{80 \cos \frac{\pi}{2}}{\varepsilon} = 0 \end{cases}$$

For the second pair of experiments, where the robot had to move along the X direction, with $\vartheta = 0$ and $\vartheta = -\pi$

$$\begin{cases} v_{x_P} = 80 \\ v_{y_P} = 0 \end{cases} \longrightarrow \begin{cases} v = 80 \cos 0 = 80 \\ \omega = -\dfrac{80 \sin 0}{\varepsilon} = 0 \end{cases}$$

In the last two experiments the robot had to move always forward, but along the two diagonals of the arena, so with $\vartheta = \pm\frac{\pi}{4}$

$$\begin{cases} v_{x_P} = 40 \\ v_{y_P} = 40 \end{cases} \longrightarrow \begin{cases} v = 40\cos\frac{\pi}{4} 40\sin\frac{\pi}{4} = 80 \\ \omega = \dfrac{40cos\frac{\pi}{4} - 40\sin\frac{\pi}{4}}{\varepsilon} = 0 \end{cases}$$

We changed the initial position of the robot during each tests, to retrieve data distributed among the complete reachable area.

In Figure 3.6 the described testing trajectories are illustrated. All the reported data was obtained with the OptiTrack positioning system.

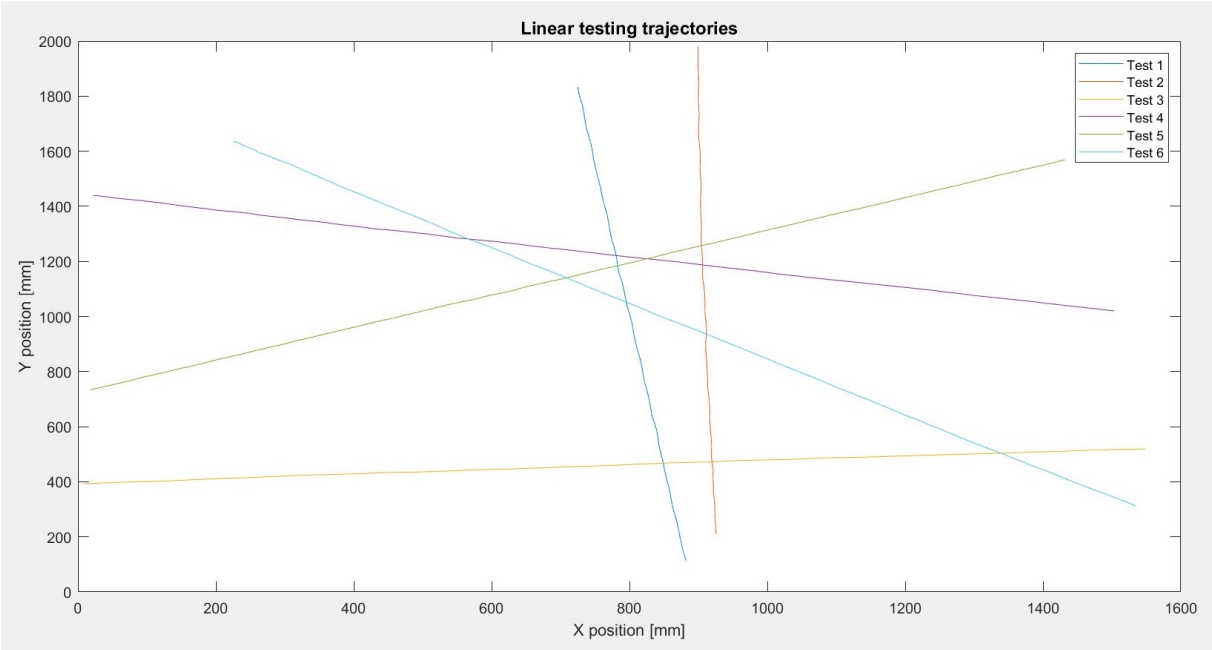


Figure 3.6: Linear testing trajectories.

The tests were performed in open loop and, as expected, the motion of the robot was affected by drifts, as it can be seen in Figure 3.6, by effect of the disturbance *w(k)*.

It is possible to rewrite the model described in equation (3.4) in the compact way

$$\begin{cases} x_{epuck}(k+1) = A_{epuck}x_{epuck}(k) + B_{epuck}u_{epuck}(k) + w(k) \\ x_{epuck}(k) = C_{epuck}x_{epuck}(k) + v(k) \end{cases} \tag{3.4}$$

where

$$x_{epuck}(k) = \begin{bmatrix} x_P \\ y_P \end{bmatrix}, u_{epuck}(k) = \begin{bmatrix} v_{x_P} \\ v_{y_P} \end{bmatrix}, w(k) = \begin{bmatrix} w_x(k) \\ w_y(k) \end{bmatrix}$$

and

$$A_{epuck} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad B_{epuck} = \begin{bmatrix} \tau & 0 \\ 0 & \tau \end{bmatrix}, \quad C_{epuck} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The one step ahead predictor based on the knowledge of the system at time instant k can be written as

$$\hat{x}_{epuck}(k+1|k) = A_{epuck}x_{epuck}(k) + B_{epuck}u_{epuck}(k)$$

Thus, the disturbance *w(k)* can be estimated at each time instant as

$$w(k) = x_{OptiTrack}(k+1) - \hat{x}_{epuck}(k+1|k) \tag{3.5}$$

This evaluation of the disturbance is related to the value of the selected sampling time. Due to the presence of the delay associated to the positions acquisition with the Decawave sensors, we decided to set the sampling time $\tau = 0.2$ s.

Figures 3.7 and 3.8 show the evolution of the disturbance along the X and Y directions, computed according to equation (3.5).

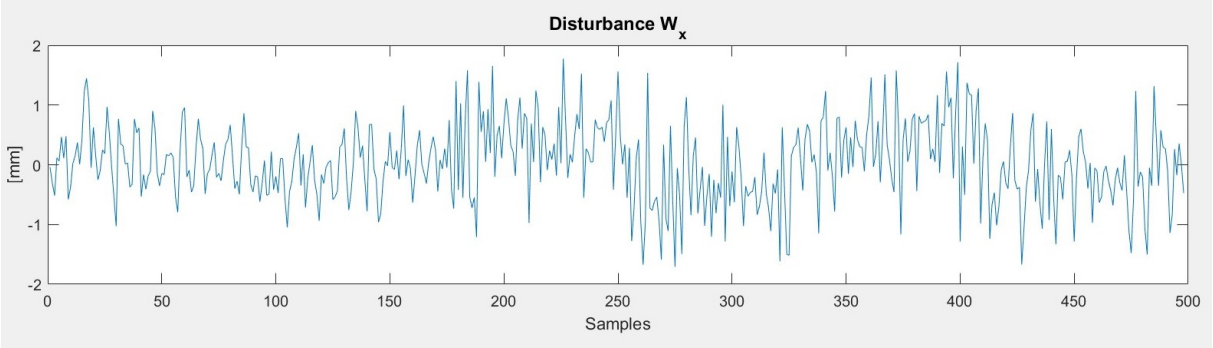


Figure 3.7: Disturbance w along the X direction.

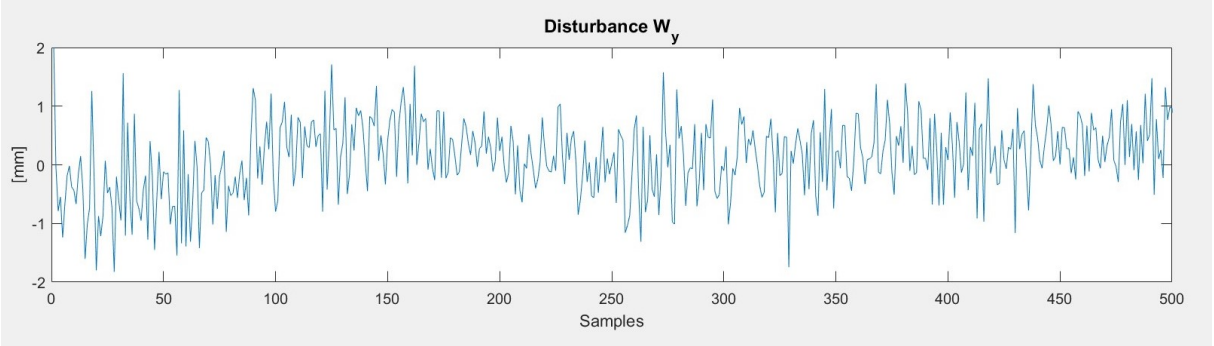


Figure 3.8: Disturbance w along the Y direction.

The disturbance mean along both the directions is equal to zero, while the covariance matrix Q is computed according to

$$Q = \frac{1}{N}\sum_{i=1}^{N}(w - \bar{w})(w - \bar{w})^T$$

where $\bar{w} = 0$, obtaining

$$Q = \begin{bmatrix} 0.4205 & -0.0015 \\ -0.0015 & 0.4207 \end{bmatrix}$$

Note that the off-diagonal terms are much smaller compared to the diagonal ones, which represent the variance of the noise along the X and Y directions.
From tests it was observed that this disturbance is bounded. It can be confined in the set $\mathbb{W}$, selected as an hyperrectangle which limits are

$$\begin{cases} w_x(k) \in [-2; 2] \\ w_y(k) \in [-2; 2] \end{cases} \tag{3.6}$$

The results can be plotted on histograms, where the affinity with a Gaussian distribution can clearly be noticed. Thus we can make the approximation that the system is subject to Gaussian noise with zero mean.

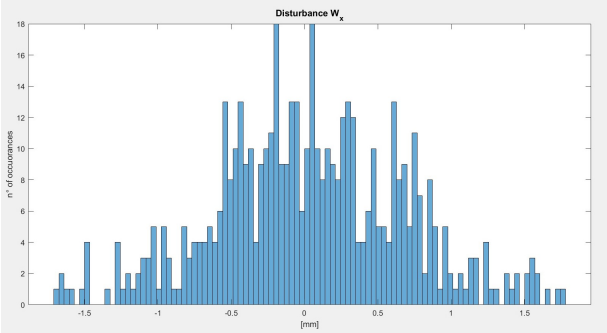


Figure 3.9: Disturbance w along the X direction.

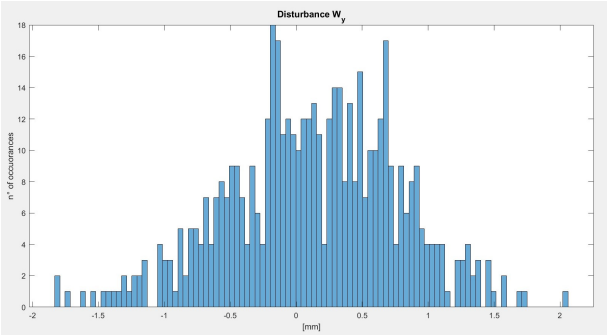


Figure 3.10: Disturbance w along the Y direction.

By comparing the covariance matrices Q and R, we can notice that the noise affecting the measurements is very large if compared to the one affecting the states.

3.3. The case study

In this thesis work we want to test and develop distributed algorithms applicable to systems composed of different subsystems, physically coupled between each others. Since the E-PUCK robots presented in the previous chapter are notable examples of independent agents, we opted to create a virtual connection between different robots. In particular, the setup defined in this thesis will be able to behave as a system, composed of two carts, moving in the 2D plane and connected through a spring and a damper.

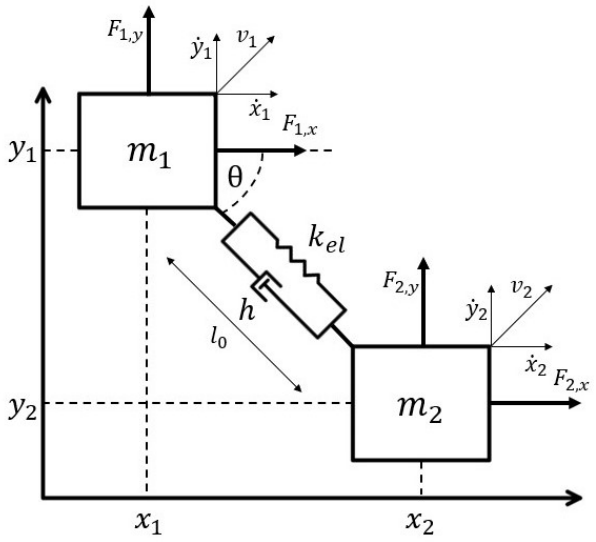


Figure 3.11: 2D Carts-Spring-Damper system.

Considering the scheme depicted in Figure 3.11, the kinematic model of the system can be easily devised

$$\begin{cases} \ddot{x}_1 = \frac{1}{m_1}\left(\left(k_{el}(l-l_0)+\frac{h}{l}((x_2-x_1)(\dot{x}_2-\dot{x}_1)+(y_2-y_1)(\dot{y}_2-\dot{y}_1))\right)\frac{(x_2-x_1)}{l}+F_{1,x}\right) \\ \ddot{y}_1 = \frac{1}{m_1}\left(\left(k_{el}(l-l_0)+\frac{h}{l}((x_2-x_1)(\dot{x}_2-\dot{x}_1)+(y_2-y_1)(\dot{y}_2-\dot{y}_1))\right)\frac{y_2-y_1)}{l}+F_{1,y}\right) \\ \ddot{x}_2 = \frac{1}{m_2}\left(\left(-k_{el}(l-l_0)-\frac{h}{l}((x_2-x_1)(\dot{x}_2-\dot{x}_1)+(y_2-y_1)(\dot{y}_2-\dot{y}_1))\right)\frac{(x_2-x_1)}{l}+F_{2,x}\right) \\ \ddot{y}_2 = \frac{1}{m_2}\left(\left(-k_{el}(l-l_0)-\frac{h}{l}((x_2-x_1)(\dot{x}_2-\dot{x}_1)+(y_2-y_1)(\dot{y}_2-\dot{y}_1))\right)\frac{(y_2-y_1)}{l}+F_{2,y}\right) \end{cases} \tag{3.7}$$

where $l = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$ and $F_{i,j}$ is the external force vector applied to subsystem i along direction j.

Here the coordinate along the X axis of the first and the second carts are represented with x_1 and x_2, respectively. The same notation is used for the coordinate along the Y axis y_1 and y_2. The equilibrium elongation of the spring is denoted as l_0, while the current elongation is indicated as l. Letter k_{el} is used to denote the spring constant, while the friction constant is denoted with letter h. m_1 and m_2 are the masses of the two carts. All the quantities are represented according to the International System of units (SI).

The model used in the following is a linearized one, obtained around the following equi-

librium point

$$\begin{cases} \bar{x}_1 = 0, & \bar{x}_2 = l_0 \cos\dfrac{\pi}{4} \\ \bar{y}_1 = 0, & \bar{y}_2 = l_0 \sin\dfrac{\pi}{4} \end{cases}$$

Defining

$$\begin{cases} \delta x_1 = x_1 - \bar{x}_1 \\ \delta x_2 = x_2 - \bar{x}_2 \\ \delta y_1 = y_1 - \bar{y}_1 \\ \delta y_2 = y_2 - \bar{y}_2 \end{cases} \quad \begin{cases} \delta \dot{x}_1 = \dot{x}_1 - \dot{\bar{x}}_1 \\ \delta \dot{x}_2 = \dot{x}_2 - \dot{\bar{x}}_2 \\ \delta \dot{y}_1 = \dot{y}_1 - \dot{\bar{y}}_1 \\ \delta \dot{y}_2 = \dot{y}_2 - \dot{\bar{y}}_2 \end{cases} \quad \begin{cases} \delta \ddot{x}_1 = \ddot{x}_1 - \ddot{\bar{x}}_1 \\ \delta \ddot{x}_2 = \ddot{x}_2 - \ddot{\bar{x}}_2 \\ \delta \ddot{y}_1 = \ddot{y}_1 - \ddot{\bar{y}}_1 \\ \delta \ddot{y}_2 = \ddot{y}_2 - \ddot{\bar{y}}_2 \end{cases}$$

the linearized model can be written as

$$\begin{cases} \delta \ddot{x}_1 = \dfrac{1}{m_1}\Big(-\dfrac{1}{2}k\delta x_1 + \dfrac{1}{2}k\delta x_2 - \dfrac{1}{2}k\delta y_1 + \dfrac{1}{2}k\delta y_2 - \dfrac{1}{2}h\delta \dot{x}_1 + \dfrac{1}{2}h\delta \dot{x}_2 - \dfrac{1}{2}h\delta \dot{y}_1 + \dfrac{1}{2}h\delta \dot{y}_2 + F_{1,x}\Big) \\ \delta \ddot{y}_1 = \dfrac{1}{m_1}\Big(-\dfrac{1}{2}k\delta x_1 + \dfrac{1}{2}k\delta x_2 - \dfrac{1}{2}k\delta y_1 + \dfrac{1}{2}k\delta y_2 - \dfrac{1}{2}h\delta \dot{x}_1 + \dfrac{1}{2}h\delta \dot{x}_2 - \dfrac{1}{2}h\delta \dot{y}_1 + \dfrac{1}{2}h\delta \dot{y}_2 + F_{1,y}\Big) \\ \delta \ddot{x}_2 = \dfrac{1}{m_2}\Big(+\dfrac{1}{2}k\delta x_1 - \dfrac{1}{2}k\delta x_2 + \dfrac{1}{2}k\delta y_1 - \dfrac{1}{2}k\delta y_2 + \dfrac{1}{2}h\delta \dot{x}_1 - \dfrac{1}{2}h\delta \dot{x}_2 + \dfrac{1}{2}h\delta \dot{y}_1 - \dfrac{1}{2}h\delta \dot{y}_2 + F_{2,x}\Big) \\ \delta \ddot{y}_2 = \dfrac{1}{m_2}\Big(+\dfrac{1}{2}k\delta x_1 - \dfrac{1}{2}k\delta x_2 + \dfrac{1}{2}k\delta y_1 - \dfrac{1}{2}k\delta y_2 + \dfrac{1}{2}h\delta \dot{x}_1 - \dfrac{1}{2}h\delta \dot{x}_2 + \dfrac{1}{2}h\delta \dot{y}_1 - \dfrac{1}{2}h\delta \dot{y}_2 + F_{2,y}\Big) \end{cases} \tag{3.8}$$

The system in (3.7) can be rewritten in state-space form as follows

$$\begin{aligned} \dot{\boldsymbol{x}} &= A^c \boldsymbol{x} + B^c \boldsymbol{u} \\ \boldsymbol{y} &= C^c \boldsymbol{x} + D^c \boldsymbol{u} \end{aligned} \tag{3.9}$$

where

$$\boldsymbol{x} = \begin{bmatrix} \delta x_1 \\ \delta \dot{x}_1 \\ \delta y_1 \\ \delta \dot{y}_1 \\ \delta x_2 \\ \delta \dot{x}_2 \\ \delta y_2 \\ \delta \dot{y}_2 \end{bmatrix}, \qquad \boldsymbol{u} = \begin{bmatrix} F_{1,x} \\ F_{1,y} \\ F_{2,x} \\ F_{2,y} \end{bmatrix}$$

3.3.1. Partitioned models

The state-space model can be partitioned in 2 low-order interconnected non overlapping subsystems, each one corresponding to a cart. Consistently, it is possible to rewrite the state space model (3.9) decomposing the state vector $\boldsymbol{x}$, the input vector $\boldsymbol{u}$, and consequently the system matrices in a non-overlapping way. For subsystem 1:

$$\begin{bmatrix} \delta\dot{x}_1 \\ \delta\ddot{x}_1 \\ \delta\dot{y}_1 \\ \delta\ddot{y}_1 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{1}{2m_1}k & -\frac{1}{2m_1}h & -\frac{1}{2m_1}k & -\frac{1}{2m_1}h \\ 0 & 0 & 0 & 1 \\ -\frac{1}{2m_1}k & -\frac{1}{2m_1}h & -\frac{1}{2m_1}k & -\frac{1}{2m_1}h \end{bmatrix}}_{A_{11}^c} \begin{bmatrix} \delta x_1 \\ \delta\dot{x}_1 \\ \delta y_1 \\ \delta\dot{y}_1 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & 0 \\ \frac{1}{m_1} & 0 \\ 0 & 0 \\ 0 & \frac{1}{m_1} \end{bmatrix}}_{B_{11}^c} \begin{bmatrix} F_{1,x} \\ F_{1,y} \end{bmatrix}$$
$$+ \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2m_1}k & \frac{1}{2m_1}h & \frac{1}{2m_1}k & \frac{1}{2m_1}h \\ 0 & 0 & 0 & 0 \\ \frac{1}{2m_1}k & \frac{1}{2m_1}h & \frac{1}{2m_1}k & \frac{1}{2m_1}h \end{bmatrix}}_{A_{12}^c} \begin{bmatrix} \delta x_2 \\ \delta\dot{x}_2 \\ \delta y_2 \\ \delta\dot{y}_2 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}}_{B_{12}^c} \begin{bmatrix} F_{2,x} \\ F_{2,y} \end{bmatrix} \tag{3.10}$$

By defining for $i = 1, 2$ $\boldsymbol{x}^{[i]} = \begin{bmatrix} \delta x_i & \delta\dot{x}_i & \delta y_i & \delta\dot{y}_i \end{bmatrix}^T$ and $\boldsymbol{u}^{[i]} = \begin{bmatrix} F_{i,x} & F_{i,y} \end{bmatrix}^T$, the model can be rewritten as

$$\dot{\boldsymbol{x}}^{[1]} = A_{11}^c \boldsymbol{x}^{[1]} + B_{11}^c \boldsymbol{u}^{[1]} + A_{12}^c \boldsymbol{x}^{[2]} + B_{12}^c \boldsymbol{u}^{[2]}$$

The same reasoning is used to derive also the state-space model of the second subsystem.

$$\dot{\boldsymbol{x}}^{[2]} = A_{22}^c \boldsymbol{x}^{[2]} + B_{22}^c \boldsymbol{u}^{[2]} + A_{21}^c \boldsymbol{x}^{[1]} + B_{21}^c \boldsymbol{u}^{[1]}$$

The discrete-time system is obtained by applying the approximated Mixed Euler-Zero-Order-Hold (ME-ZOH) discretization method [6]. It is capable to maintain the sparse zero-nonzero pattern of the system (zero-nonzero matrices A_{ij}, B_{ij}), which is lost when the exact ZOH, Backward Euler, or bilinear transformations are used.

The ME-ZOH discretization method consists of setting

$$\begin{cases} A_{ii}^d = e^{A_{ii}^c \tau} \\ A_{ij}^d = \int_0^\tau e^{A_{ii}^c \tau}\, d\tau\, A_{ij}^c \quad i \neq j \\ B_{ij}^d = \int_0^\tau e^{A_{ii}^c \tau}\, d\tau\, B_{ij}^c \quad \forall i, j \end{cases}$$

where τ is the sampling time.
In this way we obtain

$$\boldsymbol{x}^{[i]}(k+1) = A_{ii}^d \boldsymbol{x}^{[i]}(k) + B_{ii}^d \boldsymbol{u}^{[i]}(k) + \sum_{j \in \mathcal{N}_i} A_{ij}^d \boldsymbol{x}^{[j]}(k) + B_{ij}^d \boldsymbol{u}^{[j]}(k) \tag{3.11}$$

The complete, linear, discrete-time model, composed by the two subsystems (carts) can then be devised

$$\boldsymbol{x}(k+1) = \mathbf{A}^d \boldsymbol{x}(k) + \mathbf{B}^d \boldsymbol{u}(k) \tag{3.12}$$

where

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}^{[1]} \\ \boldsymbol{x}^{[2]} \end{bmatrix}, \qquad \boldsymbol{u} = \begin{bmatrix} \boldsymbol{u}^{[1]} \\ \boldsymbol{u}^{[2]} \end{bmatrix}$$

and

$$\mathbf{A}^d = \begin{bmatrix} A_{11}^d & A_{12}^d \\ A_{21}^d & A_{22}^d \end{bmatrix}, \qquad \mathbf{B}^d = \begin{bmatrix} B_{11}^d & B_{12}^d \\ B_{21}^d & B_{22}^d \end{bmatrix}$$

From now on, the superscript d representing the discrete-time notation will be omitted for brevity of representation.

3.3.2. Definition of the virtual connection between the E-PUCK robots

As specified in Chapter 1, for simplicity and to allow for more versatility, the experimental setup consists of two unicycle robots not physically connected the one with the other. In order to make these systems behave according to equation (3.10), the feedback-linearized unicycles are virtually connected through a spring and a damper. To obtain the virtual connection it is necessary to modify the input vector of the unicycle model.

Consider, for example, the motion of the feedback linearized E-PUCK robot 1 on the X direction, i.e. $v_{x_P 1} = \dot{x}_{P1}$. The term v_P, which is the E-PUCK robot input, will be assigned according to the following equation, derived from (3.7)

$$\begin{aligned}\dot{v}_{x_P1} =& \frac{1}{m_1}\Bigg(\Big(k(l-l_0)+\frac{h}{l}((x_{P_2}-x_{P_1})(v_{x_P2}-v_{x_P1}) \\ &+(y_{P_2}-y_{P_1})(v_{y_P2}-v_{y_P1})\Big)\frac{(x_{P_2}-x_{P_1})}{l}+F_{1,x}\Bigg)\end{aligned} \tag{3.13}$$

A similar procedure must be carried out for all the velocity vectors and for all the agents. The new modified inputs for the mobile robots are $F_{1,x}, F_{1,y}, F_{2,x}, F_{2,y}$, as clear from Figure 3.12. In this way the experimental setup will mimic the behaviour of the desired case study with carts connected trough a spring and a damper.

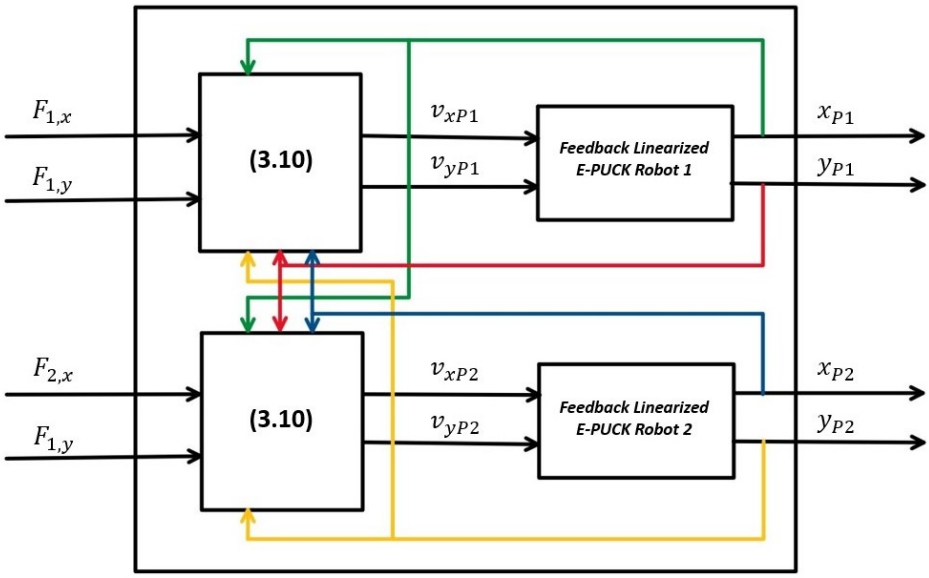


Figure 3.12: Subsystems division.

4 | Distributed model predictive control algorithm

4.1. Model predictive control

Starting from the 80's, Model Predictive Control (MPC) has became one of the most widely used control methods in industry, specifically in the process manufacturing sector. Nowadays it is by far the most popular advanced control method, adopted in all the main automation companies. This control technique was developed to deal with multivariable systems, to cope with constraints on the process variables and time varying reference signals.

This method is optimal. In fact, it consists in transforming the control synthesis problem into an optimization one, where a finite-horizon problem is solved online. In the past years it was used just with processes characterized by slow dynamics, such as chemical or petrochemical plants, since the online optimization of high-dimensional problems was quite time demanding. Nowadays, due to the evolution of the electronics, this is no more a limitation and also processes with faster dynamics can benefit from the usage of this control method.
This technique is usually used with linear models or empirical ones (impulse/step response), to have a reasonable level of complexity, but extensions that handle also non linear models exist.
The main difference with respect to other control approaches is the possibility to take into consideration also constraints on the process variables. This means that the working point of plants can be set close the operating limits, maximizing the economical profit.

MPC-based regulators rely on the knowing of the dynamical model of the system, in order to compute the future evolution of the controlled variables as a function of the control inputs. The input sequence is computed minimizing a cost function under state, input and output constraints.

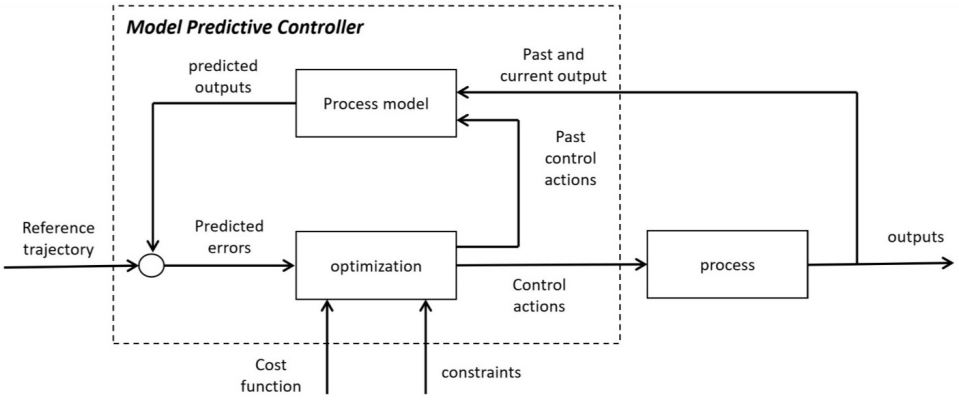


Figure 4.1: Model predictive control architecture.

The optimal input sequence, computed by minimizing a constrained cost function, is defined over a finite-horizon (t_r), from the actual time instant (t) to the end of the horizon ($t + t_r$). In model predictive control, just the first element of the optimal sequence is applied to the real process ($\hat{u}(t)$). At the next time instant ($t + 1$), a new optimization problem, defined over a shifted finite horizon $t + t_r + 1$ is solved. Thus, a new optimal input sequence is obtained, of which just the first element will be actuated ($\hat{u}(t + 1)$).

This technique relies on the Receding Horizon (RH) principle, in Figure 4.2, which permits to obtain a time invariant control law.

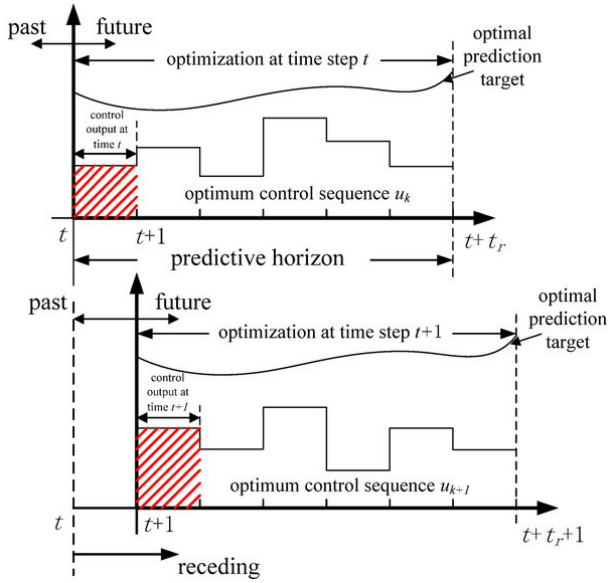


Figure 4.2: Receding horizon principle. from [13]

4.2. Introduction to distributed predictive control

In the following sections the selected distributed model predictive control strategy is deeply described and the fundamental implementation details are discussed.

The adopted state-of-the-art algorithm, named Distributed Predictive Control (DPC) [5], was specifically developed for nominal, linear discrete-time systems. It is a non-cooperative, non-iterative algorithm, where a neighbor-to-neighbor communication network and partial structural information are necessary.

Within the wide set of distributed MPC algorithms proposed in the literature, a classification can be made depending on the needed information exchange protocol (i.e. non-iterative or iterative algorithms), on the type of cost function which is optimized (i.e. cooperative or non-cooperative algorithms), and on the topology of the transmission network. The selected algorithm is non-iterative, i.e., some information is transmitted (and received) by the local regulators only once within each sampling time. It is non-cooperative, i.e., each local regulator minimizes a local performance index. Further details on the classification adopted can be found in [15].

This algorithm relies on the fact that, at each sampling time, each subsystem sends to its neighbors (i.e. the subsystems which actually affect its dynamics and the ones which share one's constraints) some information about its future reference trajectory. It requires a limited transmission of information: in fact, each subsystem needs to know the reference trajectories just of its neighbors and it does not necessitate to know their dynamical model.

The DPC algorithm handles state and input constraints and, under the assumption of existence of a suitable decentralized auxiliary control law, convergence of the closed loop control system is proven, see [5]. The control algorithm is state-feedback, so the states of each subsystem must be available or estimated.

4.3. The system and the constraints

Recall the case study discussed in Section 3.3, represented with the discrete-time, linear model (3.12).

$$\boldsymbol{x}(k+1) = \mathbf{A}\boldsymbol{x}(k) + \mathbf{B}\boldsymbol{u}(k)$$

The model can be partitioned into 2 low-order interconnected non overlapping subsystems, where the i-th subsystem (with $i = 1, 2$) model is depicted in equation (3.11).

In general we write, for all i

$$\boldsymbol{x}^{[i]}(k+1) = A_{ii}\boldsymbol{x}^{[i]}(k) + B_{ii}\boldsymbol{u}^{[i]}(k) + \sum_{j\in\mathcal{N}_i} A_{ij}\boldsymbol{x}^{[j]}(k) + B_{ij}\boldsymbol{u}^{[j]}(k)$$

Equations (3.12) and (3.11) are here reported for sake of clarity, without the superscript d (denoting the discrete time matrices) for conciseness of notation.

The matrices A_{ij} with $i \neq j$ define the dynamic coupling terms between agents and the matrices B_{ij} with $i \neq j$ define the influence of the j-th subsystem input, on the i-th subsystem state. We denote with $\mathcal{N}_i$ the set containing the dynamic neighbors of the i-th subsystem, where a subsystem j is defined as a dynamic neighbor of subsystem i if and only the state and/or the input of j affect the dynamics of subsystem i (i.e. iff $A_{ij} \neq 0$ or $B_{ij} \neq 0$).
In our framework, we have just two subsystems. It follows that each one is a dynamic neighbor of the other. In fact matrices A_{12} and A_{21} are different from zero, while matrices B_{12} and B_{21} are equal to zero.

As discussed before, the development of the DPC algorithm relies on the fulfillment of the assumption on decentralized stability.

Assumption 1. There exists a block-diagonal matrix $\mathbf{K}^{aux} = \begin{bmatrix} K_1^{aux} & 0 \\ 0 & K_2^{aux} \end{bmatrix}$, with $K_i^{aux} \in \mathbb{R}^{2\times 4}, i = 1, 2$ such that:
(i) $\mathbf{F} = \mathbf{A} + \mathbf{B}\mathbf{K}^{aux}$ is Schur.
(ii) $F_{ii} = (A_{ii} + B_{ii}K_i^{aux})$ is Schur, $i = 1, 2$.

A suitable control gain matrix $\mathbf{K}^{aux}$ is obtained solving the following Linear Matrix Inequality (LMI) problem, which allows to properly select the structure of the gain matrix. The closed loop system matrix $\mathbf{F}$ is Schur if and only if there exists a symmetric matrix $P > 0$ such that $(\mathbf{A} + \mathbf{B}\mathbf{K}^{aux})P(\mathbf{A} + \mathbf{B}\mathbf{K}^{aux})^T - P < 0$.
This inequality is not linear in its unknowns, P and $\mathbf{K}^{aux}$, but by defining $L = \mathbf{K}^{aux}P$, it can be reformulated as:

$$\begin{bmatrix} P - \mathbf{A}P\mathbf{A}^T - \mathbf{A}L^T\mathbf{B}^T - \mathbf{B}L\mathbf{A}^T & \mathbf{B}L \\ L^T\mathbf{B}^T & P \end{bmatrix} > 0$$

which is linear in P and L.
By imposing to matrix L to have the same structure of the desired gain matrix, and to matrix P to have a block-diagonal structure (can be easily achieved using YALMIP

[10]). The block-diagonal gain matrix satisfying Assumption 1, can be obtained as $\mathbf{K}^{aux} = LP^{-1}$.

It is possible to define, for each subsystem, the sets $\mathbb{X}_i$ and $\mathbb{U}_i$, as convex neighborhoods of the origin, where the i-th state and input vectors are confined. $\boldsymbol{x}^{[i]}(k) \in \mathbb{X}_i \subseteq \mathbb{R}^4$ and $\boldsymbol{u}^{[i]}(k) \in \mathbb{U}_i \subseteq \mathbb{R}^2$.

These sets are defined according to the physical limitations of the experimental setup. Such as the dimension of the arena, which limits the range of the X and Y coordinates reachable by the robot. Also, we should consider the saturation of the E-PUCK robot velocities and accelerations, described in Section 2.3.
The origin of the global reference system has been set at the center of the arena, so that the constraints on the X and Y coordinates are symmetric on both axis and limited in module at 1000 mm. The maximal velocity of the agents is constrained to be 80 mm/s. The constraint on the acceleration, whose upper bound is 800 mm/s^2, is defined in terms of maximum achievable force. In fact, recalling the weight of the mobile robots, 200 g, and the force expression $F = ma$, the maximum force that can be generated can be computed, i.e., 160 mN.
From now on all the quantities related to distances will be expressed in mm, the velocities in mm/s and mN will be used for the forces.

$$\begin{cases} \delta x^{[i]} \in [-1000, 1000] \\ \delta y^{[i]} \in [-1000, 1000] \\ \delta \dot{x}^{[i]} \in [-80, 80] \\ \delta \dot{y}^{[i]} \in [-80, 80] \end{cases} \qquad \begin{cases} F_{i,X} \in [-160, 160] \\ F_{i,Y} \in [-160, 160] \end{cases}$$

4.4. The control law and the reference trajectories

The DPC algorithm requires that, at any time instant k, each subsystem transmits to its neighbor its future state and input reference trajectories $\tilde{\boldsymbol{x}}^{[i]}(k+h)$ and $\tilde{\boldsymbol{u}}^{[i]}(k+h)$, $h = 0, 1, ..., N-1$ respectively, predicted over the whole prediction horizon N. While doing that, the distributed predictive control algorithm guarantees that the actual i-th subsystem trajectory lies in a specified time-invariant neighborhood of the reference one, i.e. $\boldsymbol{x}^{[i]}(k) \in \tilde{\boldsymbol{x}}^{[i]}(k) \oplus \mathcal{E}_i$ and $\boldsymbol{u}^{[i]}(k) \in \tilde{\boldsymbol{u}}^{[i]}(k) \oplus \mathcal{U}_i$, where $0 \in \mathcal{E}_i$ and $0 \in \mathcal{U}_i$.
In this way equation (3.11), recalling that matrices B_{ij} are equal to zero, can be rewritten as

$$\boldsymbol{x}^{[i]}(k+1) = A_{ii}\boldsymbol{x}^{[i]}(k) + B_{ii}\boldsymbol{u}^{[i]}(k) + \sum_{j \in \mathcal{N}_i} A_{ij}\tilde{\boldsymbol{x}}^{[j]}(k) + \boldsymbol{w}^{[i]}(k) \tag{4.1}$$

The term

$$\boldsymbol{w}^{[i]}(k) = \sum_{j\in\mathcal{N}_i}\Big(A_{ij}\big(\boldsymbol{x}^{[j]}(k) - \tilde{\boldsymbol{x}}^{[j]}(k)\big)\Big) \in \mathbb{W}_i$$

is a bounded disturbance to be rejected, where $\mathbb{W}_i = \bigoplus_{j\in\mathcal{N}_i} A_{ij}\mathcal{E}_j$. Also, the term $\sum_{j\in\mathcal{N}_i} A_{ij}\tilde{\boldsymbol{x}}^{[j]}(k)$ can be interpreted as an input, known in advance over the prediction horizon $h = 0, ..., N-1$.
The symbol $\oplus$ denotes the Minkowski sum, defined as $\mathbb{A}\oplus\mathbb{B} := \{a+b | a\in\mathbb{A}, b\in\mathbb{B}\}$, and $\bigoplus_{i=1}^{M}\mathbb{A}_i = \mathbb{A}_1\oplus ...\oplus\mathbb{A}_M$.

For each subsystem, a robust MPC problem for constrained linear systems subject to bounded disturbance must be solved, as the one stated in [12]. The nominal model of the i-th subsystem is

$$\hat{\boldsymbol{x}}^{[i]}(k+1) = A_{ii}\hat{\boldsymbol{x}}^{[i]}(k) + B_{ii}\hat{\boldsymbol{u}}^{[i]}(k) + \sum_{j\in\mathcal{N}_i} A_{ij}\tilde{\boldsymbol{x}}^{[j]}(k) \tag{4.2}$$

and the control law for $k \geq 0$, is given by

$$\boldsymbol{u}^{[i]}(k) = \hat{\boldsymbol{u}}^{[i]}(k) + K_i^{aux}\big(\boldsymbol{x}^{[i]}(k) - \hat{\boldsymbol{x}}^{[i]}(k)\big) \tag{4.3}$$

where K_i^{aux} satisfies Assumption 1.
Letting $\boldsymbol{z}^{[i]}(k) = \big(\boldsymbol{x}^{[i]}(k) - \hat{\boldsymbol{x}}^{[i]}(k)\big)$, from (4.1) and (4.3) we obtain

$$\boldsymbol{z}^{[i]}(k+1) = F_{ii}\boldsymbol{z}^{[i]}(k) + \boldsymbol{w}^{[i]}(k) \tag{4.4}$$

Since $\boldsymbol{w}^{[i]}(k)$ is bounded and F_{ii} is Schur, there exists a Robust Positively Invariant (RPI) set $\mathbb{Z}_i$ for (4.4) such that, for all $\boldsymbol{z}^{[i]}(k)\in\mathbb{Z}_i$, then $\boldsymbol{z}^{[i]}(k+1)\in\mathbb{Z}_i$.

The computation of the minimal RPI sets $\mathbb{Z}_i$ is performed as discussed in [14].
Specifically the i-th minimal RPi set $\mathbb{Z}_i^0$ is given by

$$\mathbb{Z}_i^0 = \bigoplus_{k=0}^{\infty} F_{ii}^k\mathbb{W}_i$$

In general, it is not possible to obtain an explicit characterization of $\mathbb{Z}_i^0$, and also $\mathbb{Z}_i^0$ may not be a polytope even if $\mathbb{W}_i$ is a polytope, see [14]. However, if $\mathbb{W}_i$ is a neighbor of the origin, (as in this application), given a scalar $\delta > 0$, there exist $\alpha_i\in\mathbb{R}, s_i\in\mathbb{I}$ such that the set

$$\mathbb{Z}_i = (1-\alpha_i)^{-1} \bigoplus_{k=0}^{s_i-1} F_{ii}^k \mathbb{W}_i$$

is an outer δ-approximation of the minimal RPI set for (4.4) which, in turn, is a polytopic RPI set. In this work, we selected the parameters $\alpha_1 = \alpha_2 = 0.01$. This is a good trade-off between computational burden and approximation quality of the RPI sets.

4.5. Optimization problem

Given $\mathbb{Z}_i$, it is possible to define the neighborhoods of the origin E_i and U_1, such that $E_i \oplus \mathbb{Z}_i \subseteq \mathcal{E}_i$ and $U_i \oplus K_i^{aux}\mathbb{Z}_i \subseteq \mathcal{U}_i$, respectively. At any time instant k, given the future reference trajectories $\tilde{\boldsymbol{x}}^{[j]}(k+h)$ and $\tilde{\boldsymbol{u}}^{[j]}(k+h)$, $h = 0, ..., N-1$, $j \in \mathcal{N}_i$, it is possible to solve the following *i-DPC problem*

$$\begin{aligned} &\min_{\hat{\boldsymbol{x}}^{[i]}(k), \hat{\boldsymbol{u}}^{[i]}(k:k+N-1)} V_i^N\big(\hat{\boldsymbol{x}}^{[i]}(k), \hat{\boldsymbol{u}}^{[i]}(k:k+N-1)\big) \\ &V_i^N = \sum_{h=0}^{N-1} l_i\big(\hat{\boldsymbol{x}}^{[i]}(k+h), \hat{\boldsymbol{u}}^{[i]}(k+h)\big) + V_i^F\big(\hat{\boldsymbol{x}}^{[i]}(k+N)\big) \end{aligned} \tag{4.5}$$

subject to (4.2) and, for $h = 0, ..., N-1$,

$$\boldsymbol{x}^{[i]}(k) - \hat{\boldsymbol{x}}^{[i]}(k) \in \mathbb{Z}_i \tag{4.6}$$

$$\hat{\boldsymbol{x}}^{[i]}(k+h) - \tilde{\boldsymbol{x}}^{[i]}(k+h) \in E_i \tag{4.7}$$

$$\hat{\boldsymbol{u}}^{[i]}(k+h) - \tilde{\boldsymbol{u}}^{[i]}(k+h) \in U_i \tag{4.8}$$

$$\hat{\boldsymbol{x}}^{[i]}(k+h) \in \hat{\mathbb{X}}_i \subseteq \mathbb{X}_i \ominus \mathbb{Z}_i \tag{4.9}$$

$$\hat{\boldsymbol{u}}^{[i]}(k+h) \in \hat{\mathbb{U}}_i \subseteq \mathbb{U}_i \ominus K_i^{aux}\mathbb{Z}_i \tag{4.10}$$

and to the terminal constraint

$$\boldsymbol{x}^{[i]}(k+N) \in \hat{\mathbb{X}}_i^F \tag{4.11}$$

$\hat{\mathbb{X}}_i^F$ is a nominal subsystem terminal set, whose properties will be later specified.
The symbol $\ominus$ denotes the Pontryagin difference, defined as $\mathbb{A} \ominus \mathbb{B} := \{a | a+b \in \mathbb{A}, \forall b \in \mathbb{B}\}$.

At time k, let the pair $\hat{\boldsymbol{x}}^{[i]}(k|k), \hat{\boldsymbol{u}}^{[i]}(k:k+N-1|k)$ be the solution to the i-DPC problem and define by $\hat{\boldsymbol{u}}^{[i]}(k|k)$ the input to the nominal system (4.2). Then, according to (4.3), the input to the system (3.11) is

$$\boldsymbol{u}^{[i]}(k) = \hat{\boldsymbol{u}}^{[i]}(k|k) + K_i^{aux}\big(\boldsymbol{x}^{[i]}(k) - \hat{\boldsymbol{x}}^{[i]}(k|k)\big) \tag{4.12}$$

The optimization problem for the i-th subsystem is referred to the nominal model (4.2), whose initial state $\hat{\boldsymbol{x}}^{[i]}(k|k)$ is also an argument of the optimization.
In (4.5), the final cost V_i^F is selected to guarantee suitable decreasing properties of the cost function.
The main assumption regarding the definition of the terminal set $\hat{\mathbb{X}}^F$ and the final cost V_i^F is stated below.

Assumption 2. Letting $\hat{\mathbb{X}} = \prod_{i=1}^{2} \hat{\mathbb{X}}_i$, $\hat{\mathbb{U}} = \prod_{i=1}^{2} \hat{\mathbb{U}}_i$ and $\hat{\mathbb{X}}^F = \prod_{i=1}^{2} \hat{\mathbb{X}}_i^F$, it holds that:
(i) $\hat{\mathbb{X}}^F \subseteq \hat{\mathbb{X}}$ is an invariant set for $\hat{\boldsymbol{x}}^+ = (\mathbf{A} + \mathbf{B}\mathbf{K}^{aux})\hat{\boldsymbol{x}}$;
(ii) $\hat{\boldsymbol{u}} = \mathbf{K}^{aux}\hat{\boldsymbol{x}} \in \hat{\mathbb{U}}$ for any $\hat{\boldsymbol{x}} \in \hat{\mathbb{X}}^F$;
(iii) for all $\hat{\boldsymbol{x}} \in \hat{\mathbb{X}}^F$ and, for a given constant κ,

$$V^F(\hat{\boldsymbol{x}}^+) - V^F(\hat{\boldsymbol{x}}) \leq -(1-\kappa) l(\hat{\boldsymbol{x}}, \hat{\boldsymbol{u}}) \tag{4.13}$$

where $V^F(\hat{\boldsymbol{x}}^+) = \sum_{i=1}^{2} V_i^F(\hat{\boldsymbol{x}}^{[i]})$ and $l(\hat{\boldsymbol{x}}, \hat{\boldsymbol{u}}) = \sum_{i=1}^{2} l_i(\hat{\boldsymbol{x}}^{[i]}, \hat{\boldsymbol{u}}^{[i]})$.

The stage and final costs $l_i(\hat{\boldsymbol{x}}^{[i]}, \hat{\boldsymbol{u}}^{[i]})$ and $V_i^F(\hat{\boldsymbol{x}}^{[i]})$, according to suggestions in [5] are selected as follows, in order to satisfy point *(iii)* of Assumption 2.

$$\begin{aligned} V_i^F(\hat{\boldsymbol{x}}^{[i]}) &= \left\|\hat{\boldsymbol{x}}^{[i]}\right\|_{P_i^0}^2 \\ l_i(\hat{\boldsymbol{x}}^{[i]}, \hat{\boldsymbol{u}}^{[i]}) &= \left\|\hat{\boldsymbol{x}}^{[i]}\right\|_{Q_i^0}^2 + \left\|\hat{\boldsymbol{u}}^{[i]}\right\|_{R_i^0}^2 \end{aligned} \tag{4.14}$$

Considering the state feedback control law $\hat{\boldsymbol{u}} = \mathbf{K}^{aux}\hat{\boldsymbol{x}}$, and recalling that $\hat{\boldsymbol{x}}^+ = \mathbf{F}\hat{\boldsymbol{x}} = (\mathbf{A} + \mathbf{B}\mathbf{K}^{aux})\boldsymbol{x}$, equation (4.13) can be rewritten as follows

$$P^0 - (\mathbf{A} + \mathbf{B}\mathbf{K}^{aux})^T P^0 (\mathbf{A} + \mathbf{B}\mathbf{K}^{aux}) - (Q^0 + \mathbf{K}^{aux^T} R^0 \mathbf{K}^{aux})(1+\kappa) > 0 \tag{4.15}$$

where

$$P^0 = \begin{bmatrix} P_1^0 & 0 \\ 0 & P_2^0 \end{bmatrix}, \quad Q^0 = \begin{bmatrix} Q_1^0 & 0 \\ 0 & Q_2^0 \end{bmatrix} \text{ and } R^0 = \begin{bmatrix} R_1^0 & 0 \\ 0 & R_2^0 \end{bmatrix}$$

Equation (4.15) is an LMI in the unknown P^0, that has been solved using Yalmip. Matrices Q_i^0 and R_i^0 are tuning parameters, that influence the dynamics of the closed loop system. In particular, larger matrices Q_i^0 the more reactive the controllers, while larger matrices R_i^0 will result in less stressful controllers on the control variables.
In this application they have been both selected as identity matrices of proper dimensions.

Denoting by $\hat{\boldsymbol{x}}^{[i]}(k+h|k)$ the state trajectory of system (4.2) stemming from $\hat{\boldsymbol{x}}^{[i]}(k|k)$ and $\hat{\boldsymbol{u}}^{[i]}(k : k+N-1|k)$, at time k it is also possible to compute $\hat{\boldsymbol{x}}^{[i]}(k+N|k)$ and

$K_i^{aux}\hat{\boldsymbol{x}}^{[i]}(k+N|k)$. In DPC, these values incrementally define the trajectories of the reference state and input variables to be used at the next time instant $k+1$, that are

$$\tilde{\boldsymbol{x}}^{[i]}(k+N) = \hat{\boldsymbol{x}}^{[i]}(k+N|k), \qquad \tilde{\boldsymbol{u}}^{[i]}(k+N) = K_i^{aux}\hat{\boldsymbol{x}}^{[i]}(k+N|k)$$

Convergence results are proved thanks to the definition of the sets of admissible initial conditions $\boldsymbol{x}(0) = (\boldsymbol{x}^{[1]}(0), \boldsymbol{x}^{[2]}(0))^T \in \mathbb{X}^N$ and of initial reference trajectories $\tilde{\boldsymbol{x}}^{[j]}(0:N-1), \tilde{\boldsymbol{u}}^{[j]}(0:N-1) \in \tilde{\mathbb{X}}_{\boldsymbol{x}(0)}$ for all $j \in \mathcal{N}_i$, for their definition see [5].

4.6. Remarks on the set definitions

Once the RPI sets for equation (4.4) have been computed, it is necessary to define the sets $\mathcal{E}_i$ fulfilling the following main assumption.

Assumption 3. Given the sets $\mathcal{E}_i, \mathcal{U}_i$, and the RPI sets $\mathbb{Z}_i$ for equation (4.4), there exists a real positive constant $\bar{\rho}_E > 0$ such that $\mathbb{Z}_i \oplus \mathcal{B}^{(2)}_{\bar{\rho}_E}(0) \subseteq \mathcal{U}_i$ for all $i = 1, 2$.
$\mathcal{B}^{(2)}_{\bar{\rho}_E}(0)$ is a ball of radius $\bar{\rho}_E > 0$ centered at the origin in the $\mathbb{R}^2$ space.

The selection of proper polytopic sets $\mathcal{E}_i$ in order to fulfill Assumption 3, is carried out considering that they can be equivalently represented in one of the following ways:

$$\begin{aligned}\mathcal{E}_i &= \{\mathcal{E}_i \in \mathbb{R}^4 | \mathcal{E}_i = \Xi_i d_i \text{ where } \|d_i\|_\infty \leq l_i\} \\ &= \{\mathcal{E}_i \in \mathbb{R}^4 | f_{i,r}^T \mathcal{E}_i \leq l_i \text{ for all r}\}\end{aligned}$$

where $d_i \in \mathbb{R}^4$, $\Xi_i \in \mathbb{R}^{4\times 4}$, $f_{i,r} \in \mathbb{R}^4$ and $r = 1, ..., 8$; for $i = 1, 2$.

The constants l_i appearing in both the equivalent definitions, can be regarded as scaling factors.
Matrices Ξ_i and vectors $f_{i,r}$ have been selected in order to obtain hyperrectangles shapes for the polyhedra $\mathcal{E}_i$.
For the computation of the scaling factors l_i, for which Assumption 3 is satisfied, it is necessary to define

$$(T_i^{(k)})^T = \begin{bmatrix} f_{i,1}^T \\ \vdots \\ f_{i,8}^T \end{bmatrix} F_{ii}^k$$

and the matrix $\mathcal{M}^P \in \mathbb{R}^{2\times 2}$ whose entries μ_{ij}^P are

$$\mu_{ii}^P = -1, \qquad i = 1,2$$
$$\mu_{ij}^P = \sum_{k=0}^{\infty}\Big(\left\|(T_i^{(k)})^T A_{ij}\Xi_j\right\|_\infty + \left\|(T_i^{(k)})^T B_{ij}K_j^{aux}\Xi_j\right\|_\infty\Big), \qquad i,j = 1,2 \text{ with } i \neq j$$

In our framework, with the selection made of the gain matrix $\mathbf{K}^{aux}$ (fulfilling Assumption 1), and since matrix $\mathcal{M}^P$ is Hurwitz, then there exist values of l_i, $i = 1,2$ such that Assumption 3 is satisfied for a sufficiently small value of δ. The constants l_i are selected as the entries of the strictly positive vector $\mathbf{l}$ satisfying $\mathcal{M}^P\mathbf{l} < 0$, obtained solving an LMI problem with Yalmip.

The sets $\mathcal{E}_i$ have been selected as hyperrectangles. This choice leads to sets $\mathbb{W}_i = \bigoplus_{j\in\mathcal{N}_i} A_{ij}\mathcal{E}_j$ which are characterized by equality constraints. This is limiting the performances of the controllers, since the RPI sets generated would allow stabilization just of points that fulfill these constraints, i.e., equal displacement among the X,Y axes. For this reason we decided to take larger sets $\mathbb{W}_i$, as hyperrectangles containing the $\bigoplus_{j\in\mathcal{N}_i} A_{ij}\mathcal{E}_j$. For the computation of the RPI sets $\mathbb{Z}_i$ we decided to use the MPT3 toolbox [8], which allows operations among polyhedra, and to carry out the development depicted previously, as in [14].

Once the RPI sets for (4.4) have been specified, then the sets E_i can be selected as any polytope satisfying $E_i \oplus \mathbb{Z}_i \subseteq \mathcal{E}_i$, and the sets U_i as any polytope satisfying $U_i = K_i^{aux}E_i$. The constraint sets $\hat{\mathbb{X}}_i$, $\hat{\mathbb{U}}_i$ are defined as $\hat{\mathbb{X}}_i \subseteq \mathbb{X}_i \ominus \mathbb{Z}_i$ and $\hat{\mathbb{U}}_i \subseteq \mathbb{U}_i \ominus K_i^{aux}\mathbb{Z}_i$ respectively, as stated in the definition of the constraints (4.9) and (4.10).
The sets $\hat{\mathbb{X}}_i^F$ are obtained by suitably scaling the sets $\mathbb{Z}_i$ by a common factor, which should be sufficiently small to not violate points (i) and (ii) of Assumption 2, so $\hat{\mathbb{X}}_i^F \subseteq \hat{\mathbb{X}}_i$ and $K_i^{aux}\hat{\mathbb{X}}_i^F \subseteq \hat{\mathbb{U}}_i$.

4.7. Definition of the initial reference trajectories

It is also necessary to define the initial reference trajectories, since they strongly affect the initial feasibility. Note that feasibility can be guaranteed by setting a sufficiently long prediction horizon N and large sets $\mathcal{E}_i$. In this work, the choice of a suitable value for N, and the definition of the initial reference trajectories is obtained using the following algorithm

Algorithm 4.1 Initial reference trajectories definition and choice of the prediction horizon N

1: For each subsystem i, at time instant $k = 0$, set $N = 1$, $\tilde{\boldsymbol{x}}^{(i)}(0) = \boldsymbol{x}^{(i)}(0)$ and a feasible value of $\tilde{\boldsymbol{u}}^{(i)}(0) = K_i^{aux}\boldsymbol{x}^{(i)}(0)$.
2: **while** $\tilde{\boldsymbol{x}}^{(i)}(N-1) \notin \hat{\mathbb{X}}_i^F$ **do**
3: Solve the i-DPC problem (4.5), subject to constraints (4.6)-(4.10) and additional constraints (4.9) and (4.10) for $h = N$.
4: Set the reference state and input, at time N, as $\tilde{\boldsymbol{x}}^{(i)}(N) = \hat{\boldsymbol{x}}^{(i)}(N|0)$ and $\tilde{\boldsymbol{u}}^{(i)}(N) = K_i^{aux}\hat{\boldsymbol{x}}^{(i)}(N|0)$, respectively.
5: Transmit $\tilde{\boldsymbol{x}}^{(i)}(N)$ and $\tilde{\boldsymbol{u}}^{(i)}(N)$ and receive $\tilde{\boldsymbol{x}}^{(j)}(N)$ and $\tilde{\boldsymbol{u}}^{(j)}(N)$ from the neighbors $j \in \mathcal{N}_i$.
6: set $N = N + 1$.
7: **end while**

Once the initial reference trajectories, the sets, and the cost function have been defined in order to fulfill the previously cited assumptions, it is possible to state the main Theorem.
Theorem. Let Assumptions 1-3 be satisfied and let $\mathcal{E}_i$ and $\mathcal{U}_i$ be neighborhoods of the origin satisfying $E_i \oplus \mathbb{Z}_i \subseteq \mathcal{E}_i$ and $U_i \oplus K_i^{aux}\mathbb{Z}_i \subseteq \mathcal{U}_i$. Then, for any initial reference trajectories in $\tilde{\mathbb{X}}_{\boldsymbol{x}(0)}$, the trajectory $\boldsymbol{x}(k)$, starting from any initial condition $\boldsymbol{x}(0) \in \mathbb{X}^N$, asymptotically converges to the origin.

4.8. Numerical implementation of the DPC algorithm

We decided to formulate the complete set of i-DPC problems as a unique (centralized) optimization one. In fact, as already mentioned, we are using just a host PC to run all the controllers. The regulators of the two carts have been implemented together in centralized fashion, formulating the complete DPC problem in order to preserve the division among the i-th controllers. In this way, solving the complete (centralized) scheme, is exactly equivalent to solve, the single i-th DPC problems in parallel.

This optimization problem can be solved using the function *quadprog* provided by MATLAB. To achieve that, it is required to reformulate the i-th problem (4.5), according to the case study model of the i-th subsystem (3.11).

The nominal model with which the i-DPC problem is referred to (4.2), can be rewritten in a centralized fashion as

$$\hat{\boldsymbol{x}}(k+1) = \mathbf{A}_D\hat{\boldsymbol{x}}(k) + \mathbf{B}\hat{\boldsymbol{u}}(k) + \mathbf{A}_{ND}\tilde{\boldsymbol{x}}(k) \tag{4.16}$$

where $\mathbf{A}_D = \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \end{bmatrix}$, and $\mathbf{A}_{ND} = \begin{bmatrix} 0 & A_{12} \\ A_{21} & 0 \end{bmatrix}$.

Since the optimization problem (4.5) is defined over the whole prediction horizon N, it is possible to rewrite equation (4.16) as

$$\underbrace{\begin{bmatrix} \hat{\boldsymbol{x}}(0) \\ \hat{\boldsymbol{x}}(1) \\ \hat{\boldsymbol{x}}(2) \\ \vdots \\ \hat{\boldsymbol{x}}(N) \end{bmatrix}}_{\hat{\mathbf{x}}} = \underbrace{\begin{bmatrix} I \\ \mathbf{A}_D \\ \mathbf{A}_D^2 \\ \vdots \\ \mathbf{A}_D^N \end{bmatrix}}_{\mathcal{A}_D} \hat{\boldsymbol{x}}(0|0) + \underbrace{\begin{bmatrix} 0 & 0 & \dots & \dots & 0 \\ \mathbf{B} & 0 & \dots & \dots & 0 \\ \mathbf{A}_D\mathbf{B} & \mathbf{B} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \mathbf{A}_D^{N-1}\mathbf{B} & \mathbf{A}_D^{N-2}\mathbf{B} & \dots & \mathbf{A}_D\mathbf{B} & \mathbf{B} \end{bmatrix}}_{\mathcal{B}} \underbrace{\begin{bmatrix} \hat{\boldsymbol{u}}(0) \\ \hat{\boldsymbol{u}}(1) \\ \vdots \\ \hat{\boldsymbol{u}}(N-1) \end{bmatrix}}_{\hat{\mathbf{u}}}$$

$$+ \underbrace{\begin{bmatrix} 0 & 0 & \dots & \dots & 0 \\ \mathbf{A}_{ND} & 0 & \dots & \dots & 0 \\ \mathbf{A}_D\mathbf{A}_{ND} & \mathbf{A}_{ND} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \mathbf{A}_D^{N-1}\mathbf{A}_{ND} & \mathbf{A}_D^{N-2}\mathbf{A}_{ND} & \dots & \mathbf{A}_D\mathbf{A}_{ND} & \mathbf{A}_{ND} \end{bmatrix}}_{\mathcal{A}_{ND}} \underbrace{\begin{bmatrix} \tilde{\boldsymbol{x}}(0) \\ \tilde{\boldsymbol{x}}(1) \\ \vdots \\ \tilde{\boldsymbol{x}}(N-1) \end{bmatrix}}_{\tilde{\mathbf{x}}}$$

This in compact form, can be rewritten as

$$\hat{\mathbf{x}} = \mathcal{A}_D\hat{\boldsymbol{x}}(0|0) + \mathcal{B}\hat{\mathbf{u}} + \mathcal{A}_{ND}\tilde{\mathbf{x}}$$

According to equation (4.12), the control action for subsystem (3.11) is computed based on the input sequence $\hat{\mathbf{u}}^{[i]}$ (defined over the whole prediction horizon) and on its current state $\hat{\boldsymbol{x}}(0|0)$.

For this reason the optimization variable is defined as $\chi = \begin{bmatrix} \hat{\boldsymbol{x}}(0|0) \\ \hat{\mathbf{u}} \end{bmatrix}$, and

$$\hat{\mathbf{x}} = \underbrace{\begin{bmatrix} \mathcal{A}_D & \mathcal{B} \end{bmatrix}}_{\mathcal{A}_\chi} \chi + \mathcal{A}_{ND}\tilde{\mathbf{x}} \tag{4.17}$$

According to the selected stage and final cost (4.14), the optimization problem (4.5) can be reformulated, and the associated cost function J_{i-DPC} to be minimized is

$$J_{i-DPC} = \sum_{k=0}^{N-1} \left(\left\| \hat{\boldsymbol{x}}^{[i]}(k) \right\|_{Q_i^0}^2 + \left\| \hat{\boldsymbol{u}}^{[i]}(k) \right\|_{R_i^0}^2 \right) + \left\| \hat{\boldsymbol{x}}^{[i]}(N) \right\|_{P_i^0}^2 \tag{4.18}$$

Recalling that the MATLAB function *quadprog* is capable to find a vector η that minimizes a quadratic function in the form $J_{quadprog} = \frac{1}{2}\left(\eta^T H \eta + f^T \eta\right)$.
The cost function (4.18), can be rearranged in centralized fashion, according to (4.17), neglecting the terms that do not depend on the optimization variable χ.

$$J_{DPC} = \chi^T \underbrace{\left(\mathcal{A}_\chi^T Q^0 \mathcal{A}_\chi + \begin{bmatrix} 0 \\ I \end{bmatrix} R^0 \begin{bmatrix} 0 & I \end{bmatrix} \right)}_{H} \chi + \underbrace{2\tilde{\mathbf{x}} \mathcal{A}_{ND}^T Q^0 \mathcal{A}_{ND}}_{f} \chi$$

The constraints (4.6)-(4.11) must be expressed in the form $H_{ineq}\eta \leq b_{ineq}$ and with respect to the optimization variable χ.

The constraint (4.6), defining $\mathbb{Z} = \prod_{i=1}^{2} \mathbb{Z}_i$, and $H_Z \in \mathbb{R}^{16\times 8}, b_Z \in \mathbb{R}^{16\times 1}$ as the matrices of the half-space representation (H-representation) $\{z \in \mathbb{Z} | H_Z z \leq b_Z\}$, can be expressed as

$$\boldsymbol{x}(0) - \hat{\boldsymbol{x}}(0) \in \mathbb{Z} \longrightarrow \begin{bmatrix} -H_Z & 0 \end{bmatrix} \chi \leq b_Z - H_Z \boldsymbol{x}(0)$$

The constraints (4.7), defining $E = \prod_{i=1}^{2} E_i$, and $H_E \in \mathbb{R}^{16\times 8}, b_E \in \mathbb{R}^{16\times 1}$ as the matrices of the H-representation $\{e \in E | H_E e \leq b_E\}$, can be expressed as

$$\begin{cases} \hat{\boldsymbol{x}}(0) - \tilde{\boldsymbol{x}}(0) \in E \\ \hat{\boldsymbol{x}}(1) - \tilde{\boldsymbol{x}}(1) \in E \\ \vdots \\ \hat{\boldsymbol{x}}(N-1) - \tilde{\boldsymbol{x}}(N-1) \in E \end{cases} \longrightarrow \underbrace{\begin{bmatrix} \hat{H}_E & 0 & \dots & \dots & 0 \\ 0 & \hat{H}_E & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \hat{H}_E & 0 \\ 0 & \dots & \dots & 0 & 0 \end{bmatrix}}_{\mathcal{H}_E} \hat{\mathbf{x}} \leq \underbrace{\begin{bmatrix} \hat{b}_E \\ \hat{b}_E \\ \vdots \\ \hat{b}_E \\ 0 \end{bmatrix}}_{\mathcal{B}_E} + \mathcal{H}_E \begin{bmatrix} \tilde{\mathbf{x}} \\ 0 \end{bmatrix}$$

$$\longrightarrow \mathcal{H}_E \mathcal{A}_\chi \chi \leq \mathcal{B}_E + \mathcal{H}_E \begin{bmatrix} \tilde{\mathbf{x}} \\ 0 \end{bmatrix} - \mathcal{H}_E \mathcal{A}_{ND} \tilde{\mathbf{x}}$$

The constraints (4.8), defining $U = \prod_{i=1}^{2} U_i$, and $H_U \in \mathbb{R}^{8\times 4}, b_U \in \mathbb{R}^{8\times 1}$ as the matrices of the H-representation $\{u \in U | H_U u \leq b_U\}$, can be expressed as

$$\begin{cases} \hat{\boldsymbol{u}}(0) - \tilde{\boldsymbol{u}}(0) \in U \\ \hat{\boldsymbol{u}}(1) - \tilde{\boldsymbol{u}}(1) \in U \\ \vdots \\ \hat{\boldsymbol{u}}(N-1) - \tilde{\boldsymbol{u}}(N-1) \in U \end{cases} \longrightarrow \underbrace{\begin{bmatrix} H_U & 0 & \dots & \dots & 0 \\ 0 & H_U & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & H_U \end{bmatrix}}_{\mathcal{H}_U} \hat{\mathbf{u}} \leq \underbrace{\begin{bmatrix} b_U \\ b_U \\ \vdots \\ b_U \end{bmatrix}}_{\mathcal{B}_U} + \mathcal{H}_U \tilde{\mathbf{u}}$$

$$\longrightarrow \mathcal{H}_U \begin{bmatrix} 0 & I \end{bmatrix} \chi \leq \mathcal{B}_U + \mathcal{H}_U \tilde{\mathbf{u}}$$

The constraints (4.9) and (4.11), defining $\hat{H}_X \in \mathbb{R}^{16\times 8}, \hat{b}_X \in \mathbb{R}^{16\times 1}$ as the matrices of the H-representation $\{x \in \hat{\mathbb{X}} | \hat{H}_X x \leq \hat{b}_X\}$, and $\hat{H}_{XF} \in \mathbb{R}^{16\times 8}, \hat{b}_{XF} \in \mathbb{R}^{16\times 1}$ as the matrices of the H-representation $\{x \in \hat{\mathbb{X}}^F | \hat{H}_{XF} x \leq \hat{b}_{XF}\}$, can be expressed as

$$\begin{cases} \hat{\boldsymbol{x}}(0) \in \hat{\mathbb{X}} \\ \hat{\boldsymbol{x}}(1) \in \hat{\mathbb{X}} \\ \vdots \\ \hat{\boldsymbol{x}}(N-1) \in \hat{\mathbb{X}} \\ \hat{\boldsymbol{x}}(N) \in \hat{\mathbb{X}}^F \end{cases} \longrightarrow \underbrace{\begin{bmatrix} \hat{H}_X & 0 & \dots & \dots & 0 \\ 0 & \hat{H}_X & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \hat{H}_X & 0 \\ 0 & \dots & \dots & 0 & \hat{H}_{XF} \end{bmatrix}}_{\mathcal{H}_X} \hat{\mathbf{x}} \leq \underbrace{\begin{bmatrix} \hat{b}_X \\ \hat{b}_X \\ \vdots \\ \hat{b}_X \\ \hat{b}_{XF} \end{bmatrix}}_{\mathcal{B}_X} \longrightarrow \begin{matrix} \mathcal{H}_X \mathcal{A}_\chi \chi \leq \mathcal{B}_X \\ - \mathcal{H}_X \mathcal{A}_{ND} \tilde{\mathbf{x}} \end{matrix}$$

The constraints (4.10), defining $\hat{H}_U \in \mathbb{R}^{8\times 4}, \hat{b}_U \in \mathbb{R}^{8\times 1}$ as the matrices of the H-representation $\{u \in \hat{\mathbb{U}} | \hat{H}_U u \leq \hat{b}_U\}$, can be expressed as

$$\begin{cases} \hat{\boldsymbol{u}}(0) \in \hat{\mathbb{U}} \\ \hat{\boldsymbol{u}}(1) \in \hat{\mathbb{U}} \\ \vdots \\ \hat{\boldsymbol{u}}(N-1) \in \hat{\mathbb{U}} \end{cases} \longrightarrow \underbrace{\begin{bmatrix} \hat{H}_U & 0 & \dots & \dots & 0 \\ 0 & \hat{H}_U & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \ddots & 0 \\ 0 & \dots & \dots & 0 & \hat{H}_U \end{bmatrix}}_{\mathcal{H}_{\hat{U}}} \hat{\mathbf{u}} \leq \underbrace{\begin{bmatrix} \hat{b}_U \\ \hat{b}_U \\ \vdots \\ \hat{b}_U \end{bmatrix}}_{\mathcal{B}_{\hat{U}}} \longrightarrow \mathcal{H}_{\hat{U}} \begin{bmatrix} 0 & I \end{bmatrix} \chi \leq \mathcal{B}_{\hat{U}}$$

5 | Robust distributed predictive control with delay compensation

In Chapter 3 a complete characterization of the experimental setup introduced in Chapter 2 was carried out. We have shown that the laboratory equipment employed in this work introduces non negligible delays. The latter are mainly related to the position data acquisition operation executed by the Decawave real time positioning system.
Moreover, also the disturbance *w(k)* affecting the state of the unicycle linearized model, and the noise *v(k)* affecting the calibrated position measures, produce non negligible effects. They must be considered in the development of the controllers.

The distributed predictive control algorithm, introduced in Chapter 4, has been developed referring to the model in equation (4.1), which does not consider the presence of delays and external disturbances affecting the system. For these reasons, a robust networked version of DPC is developed, capable to deal with the non-idealties affecting the experimental setup, and to provide global stability together with satisfactory performances.

5.1. Compensation of the delay with predictions

As discussed in Section 3.1.1, the Decawave sensors used in this work introduce an average measurement delay of 206.2 ms, with a standard deviation of 41.4 ms. Due to the presence of this delay, we selected the sampling time τ to be 0.2 s, which is equal to the average value of the delay affecting the system.
This choice entails that the measurement of the E-PUCK robot actual position obtained at time instant k, arrives at the controllers with approximately a 1-step delay, at time instant $k+1$.

As DPC is a state feedback control strategy, the computation of the correct control action (4.12) is done under the assumption that the actual state $\boldsymbol{x}^{[i]}(k)$ (3.10), is available at

time instant k.
Recall that

$$\boldsymbol{x}^{[i]}(k) = \begin{bmatrix} \delta x_i(k) & \delta \dot{x}_i(k) & \delta y_i(k) & \delta \dot{y}_i(k) \end{bmatrix}^T = \begin{bmatrix} x_{Pi}(k) & v_{x_{Pi}}(k) & y_{Pi}(k) & v_{y_{Pi}}(k) \end{bmatrix}^T$$

The Decawave sensors provide the measures of the robot position $(\delta x_i(k), \delta y_i(k))$, while the velocities $(\delta \dot{x}_i(k), \delta \dot{y}_i(k))$ are available since they correspond with the inputs to the feedback-linearized robots, see (2.5). The latter are given without delay and without uncertainty, since they are directly computed by the regulators. On the other hand, the positions are received by the controller with a 1-step delay and noise *v(k)*.

This means that the control action $\boldsymbol{u}^{[i]}(k)$, computed by the regulators at the actual time instant k as in equation (4.12), will not be based on the actual position measurement $(\delta x_i(k), \delta y_i(k))$. Instead, the measure of the position taken by the unicycle robot at the past time instant $(\delta x_i(k-1), \delta y_i(k-1))$ will be used for computing a suitable prediction.

Recall the feedback-linearized model of the E-PUCK robot (3.4)

$$\begin{cases} x_{epuck}(k+1) = A_{epuck} x_{epuck}(k) + B_{epuck} u_{epuck}(k) + w(k) \\ x_{epuck}^{measured}(k) = C_{epuck} x_{epuck}(k) + v(k) \end{cases} \tag{5.1}$$

where

$$x_{epuck} = \begin{bmatrix} x_P \\ y_P \end{bmatrix}, u_{epuck} = \begin{bmatrix} v_{x_P} \\ v_{y_P} \end{bmatrix}$$

It is possible to develop the one-step ahead predictor which, given the past state (position measures from decawave sensor) and the past control action, is capable to estimate the current state of the robot, i.e., its actual position.

The one-step ahead predictor related to the nominal model of the unicycle robot (2.5) can be written as

$$x_{epuck}^{predicted}(k) = A_{epuck} x_{epuck}^{measured}(k-1) + B_{epuck} u_{epuck}(k-1)$$

The information used for the computation of the control action at each time step is

$$\bar{\boldsymbol{x}}^{[i]} = \begin{bmatrix} x_{Pi}^{predicted} & v_{x_{Pi}} & y_{Pi}^{predicted} & v_{y_{Pi}} \end{bmatrix}^T \tag{5.2}$$

Note that

$$\begin{aligned}\boldsymbol{x}^{[i]}(k) - \bar{\boldsymbol{x}}^{[i]}(k) =& P\Big(A_{epuck}x_{epuck,i}(k-1) + B_{epuck}u_{epuck,i}(k-1) + w^{[i]}(k-1) \\ & - A_{epuck}x_{epuck,i}^{measured}(k-1) - B_{epuck}u_{epuck,i}(k-1)\Big) \\ =& P\Big(-A_{epuck}v^{[i]}(k-1) + w^{[i]}(k-1)\Big)\end{aligned} \tag{5.3}$$

where

$$P = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

5.2. The model, the predictions and constraint definition

The algorithm developed in Chapter 4 will be now reformulated considering the issues derived from the presence of disturbances and delays.

As shown in Section 5.1, the presence of delays can be compensated thanks to the use of predictions, whose effect is that noisy predictions $\bar{\boldsymbol{x}}^{[i]}(k)$ need to be considered at the place of data $\boldsymbol{x}^{[i]}(k)$.
The algorithm must simply be restructured to cope with uncertainties due to prediction errors and model perturbations.

Under the assumption that the "virtual connections" between subsystems are defined in an ideal way, we rewrite the model equations as follows

$$\boldsymbol{x}^{[i]}(k+1) = A_{ii}\boldsymbol{x}^{[i]}(k) + B_{ii}\boldsymbol{u}^{[i]}(k) + \sum_{j\in\mathcal{N}_i} A_{ij}\boldsymbol{x}^{[j]}(k) + \boldsymbol{w}_P^{[i]}(k) \tag{5.4}$$

In our setup, we derive the perturbation $\boldsymbol{w}_P^{[i]}(k)$ from the perturbation *w(k)* characterizing the E-PUCK motion, i.e.

$$\boldsymbol{w}_P^{[i]}(k) = Pw^{[i]}(k)$$

As done in Chapter 4, we rewrite (5.4) as

$$\begin{aligned}\boldsymbol{x}^{[i]}(k+1) =& A_{ii}\boldsymbol{x}^{[i]}(k) + B_{ii}\boldsymbol{u}^{[i]}(k) + \sum_{j\in\mathcal{N}_i} A_{ij}\tilde{\boldsymbol{x}}^{[j]}(k) + \boldsymbol{w}_P^{[i]}(k) \\ &+ \sum_{j\in\mathcal{N}_i}\big(A_{ij}(\boldsymbol{x}^{[j]}(k) - \tilde{\boldsymbol{x}}^{[j]}(k))\big)\end{aligned} \tag{5.5}$$

As stated in Section 5.1, the available measurement of $\boldsymbol{x}^{[i]}(k)$ is indeed $\bar{\boldsymbol{x}}^{[i]}(k)$, where, according to 5.3

$$e(k) = \boldsymbol{x}^{[i]}(k) - \bar{\boldsymbol{x}}^{[i]}(k) = P\big(w^{[i]}(k-1) - A_{epuck}v^{[i]}(k-1)\big) \tag{5.6}$$

The control law is generated using the available datum $\bar{\boldsymbol{x}}^{[i]}(k)$, i.e.

$$\boldsymbol{u}^{[i]}(k) = \hat{\boldsymbol{u}}^{[i]}(k) + K_i^{aux}\big(\bar{\boldsymbol{x}}^{[i]}(k) - \hat{\boldsymbol{x}}^{[i]}(k)\big) \tag{5.7}$$

The terms $\hat{\boldsymbol{u}}^{[i]}(k)$ and $\hat{\boldsymbol{x}}^{[i]}(k)$ are, as in Chapter 4, the input and the state, respectively, of the nominal model (4.2), whose equation is here reported for clarity.

$$\hat{\boldsymbol{x}}^{[i]}(k+1) = A_{ii}\hat{\boldsymbol{x}}^{[i]}(k) + B_{ii}\hat{\boldsymbol{u}}^{[i]}(k) + \sum_{j\in\mathcal{N}_i} A_{ij}\tilde{\boldsymbol{x}}^{[j]}(k)$$

Using (5.4),(4.2), we can write

$$\begin{aligned}\bar{\boldsymbol{x}}^{[i]}(k+1) - \hat{\boldsymbol{x}}^{[i]}(k+1) &= \bar{\boldsymbol{x}}^{[i]}(k+1) - \boldsymbol{x}^{[i]}(k+1) + \boldsymbol{x}^{[i]}(k+1) - \hat{\boldsymbol{x}}^{[i]}(k+1) \\ &= P\big(A_{epuck}v^{[i]}(k) - w^{[i]}(k)\big) + A_{ii}\big(\boldsymbol{x}^{[i]}(k) - \hat{\boldsymbol{x}}^{[i]}(k)\big) \\ &\quad + B_{ii}K_i^{aux}\big(\bar{\boldsymbol{x}}^{[i]}(k) - \hat{\boldsymbol{x}}^{[i]}(k)\big) + \boldsymbol{w}_P^{[i]}(k) \\ &\quad + \sum_{j\in\mathcal{N}_i} A_{ij}\big(\boldsymbol{x}^{[j]}(k) - \tilde{\boldsymbol{x}}^{[j]}(k)\big) \\ &= \big(A_{ii} + B_{ii}K_i^{aux}\big)\big(\bar{\boldsymbol{x}}^{[i]}(k) - \hat{\boldsymbol{x}}^{[i]}(k)\big) + A_{ii}\big(\boldsymbol{x}^{[i]}(k) - \bar{\boldsymbol{x}}^{[i]}(k)\big) \\ &\quad + \sum_{j\in\mathcal{N}_i} A_{ij}\big(\boldsymbol{x}^{[j]}(k) - \tilde{\boldsymbol{x}}^{[j]}(k)\big) + P\big(A_{epuck}v^{[i]}(k)\big) \\ &= \big(A_{ii} + B_{ii}K_i^{aux}\big)\big(\bar{\boldsymbol{x}}^{[i]}(k) - \hat{\boldsymbol{x}}^{[i]}(k)\big) + \bar{\boldsymbol{w}}^{[i]}(k)\end{aligned} \tag{5.8}$$

where

$$\begin{aligned}\bar{\boldsymbol{w}}^{[i]}(k) =& A_{ii}\big(w^{[i]}(k-1) - A_{epuck}v^{[i]}(k-1)\big) + P\big(A_{epuck}v^{[i]}(k)\big) \\ &+ \sum_{j\in\mathcal{N}_i} A_{ij}\big(\boldsymbol{x}^{[j]}(k) - \tilde{\boldsymbol{x}}^{[j]}(k)\big)\end{aligned}$$

Under the assumption that we can constrain

$$\boldsymbol{x}^{[i]}(k) - \tilde{\boldsymbol{x}}^{[i]}(k) \in \mathcal{E}_i \tag{5.9}$$

$$\boldsymbol{u}^{[i]}(k) - \tilde{\boldsymbol{u}}^{[i]}(k) \in \mathcal{U}_i \tag{5.10}$$

then $\bar{\boldsymbol{w}}^{[i]}(k) \in \bar{\mathbb{W}}_i$, where

$$\bar{\mathbb{W}}_i = A_{ii}\mathbb{W} \oplus A_{ii}(-A_{epuck}\mathbb{V}) \oplus P(A_{epuck}\mathbb{V}) \oplus \bigoplus_{j \in \mathcal{N}_i} A_{ij}\mathcal{E}_j$$

where $\mathbb{W}$ and $\mathbb{V}$ are the sets containing the disturbances $w^{[i]}(k)$ and $v^{[i]}(k)$, selected as hyperrectangles with bounds imposed according to (3.6),(3.3).

The RPI sets $\mathbb{Z}_i$ can be defined for the system (5.8) guaranteeing that, if $\bar{\boldsymbol{x}}^{[i]}(k) - \hat{\boldsymbol{x}}^{[i]}(k) \in \mathbb{Z}_i$, then $\bar{\boldsymbol{x}}^{[i]}(k+h) - \hat{\boldsymbol{x}}^{[i]}(k+h) \in \mathbb{Z}_i$, $\forall h = 1, 2, ..., N$.

At this point, to constrain (5.9), we need to enforce, for all:

$$\boldsymbol{x}^{[i]}(k) - \tilde{\boldsymbol{x}}^{[i]}(k) = \boldsymbol{x}^{[i]}(k) - \bar{\boldsymbol{x}}^{[i]}(k) + \bar{\boldsymbol{x}}^{[i]}(k) - \hat{\boldsymbol{x}}^{[i]}(k) + \hat{\boldsymbol{x}}^{[i]}(k) - \tilde{\boldsymbol{x}}^{[i]}(k) \in \mathcal{E}_i$$

Since $\boldsymbol{x}^{[i]}(k) - \bar{\boldsymbol{x}}^{[i]}(k) \in P\big(\mathbb{W} \oplus (-A_{epuck}\mathbb{V})\big)$ and $\bar{\boldsymbol{x}}^{[i]}(k) - \hat{\boldsymbol{x}}^{[i]}(k) \in \mathbb{Z}_i$, (5.9) can be enforced by imposing $\hat{\boldsymbol{x}}^{[i]}(k) - \tilde{\boldsymbol{x}}^{[i]}(k) \in E_i$, where

$$P\big(\mathbb{W} \oplus (-A_{epuck}\mathbb{V})\big) \oplus \mathbb{Z}_i \oplus E_i \subseteq \mathcal{E}_i$$

Also, recall that

$$\begin{aligned}\boldsymbol{u}^{[i]}(k) - \tilde{\boldsymbol{u}}^{[i]}(k) =& \boldsymbol{u}^{[i]}(k) - \hat{\boldsymbol{u}}^{[i]}(k) + \hat{\boldsymbol{u}}^{[i]}(k) - \tilde{\boldsymbol{u}}^{[i]}(k) \\ & K_i^{aux}(\bar{\boldsymbol{x}}^{[i]}(k) - \hat{\boldsymbol{x}}^{[i]}(k)) + \hat{\boldsymbol{u}}^{[i]}(k) - \tilde{\boldsymbol{u}}^{[i]}(k)\end{aligned}$$

Therefore, (5.10) can be imposed by enforcing $\hat{\boldsymbol{u}}^{[i]}(k) - \tilde{\boldsymbol{u}}^{[i]}(k) \in U_i$, provided that $K_i^{aux}\mathbb{Z}_i \oplus U_i \subseteq \mathcal{U}_i$.

Finally, it is worth noting that, to impose $\boldsymbol{x}^{[i]}(k) \in \mathbb{X}_i$ we need to enforce

$$\boldsymbol{x}^{[i]}(k) + \hat{\boldsymbol{x}}^{[i]}(k) - \hat{\boldsymbol{x}}^{[i]}(k) - \bar{\boldsymbol{x}}^{[i]}(k) + \bar{\boldsymbol{x}}^{[i]}(k) \subseteq \hat{\boldsymbol{x}}^{[i]}(k) \oplus \mathbb{Z}_i \oplus P\big(\mathbb{W} \oplus (-A_{epuck}\mathbb{V})\big) \subseteq \mathbb{X}_i$$

Also, to impose $\boldsymbol{u}^{[i]}(k) \in \mathbb{U}_i$ we need to enforce

$$\boldsymbol{u}^{[i]}(k) + \hat{\boldsymbol{u}}^{[i]}(k) - \hat{\boldsymbol{u}}^{[i]}(k) \subseteq \hat{\boldsymbol{u}}^{[i]}(k) \oplus K_i^{aux}\mathbb{Z}_i \subseteq \mathbb{U}_i$$

5.3. The optimization problem

Under the changes defined above, the i-DPC problem defined in Section 4.5 must be reformulated as follows

$$\begin{aligned} &\min_{\hat{\boldsymbol{x}}^{[i]}(k),\hat{\boldsymbol{u}}^{[i]}(k:k+N-1)} V_i^N\big(\hat{\boldsymbol{x}}^{[i]}(k),\hat{\boldsymbol{u}}^{[i]}(k:k+N-1)\big) \\ &V_i^N = \sum_{h=0}^{N-1} l_i\big(\hat{\boldsymbol{x}}^{[i]}(k+h),\hat{\boldsymbol{u}}^{[i]}(k+h)\big) + V_i^F\big(\hat{\boldsymbol{x}}^{[i]}(k+N)\big) \end{aligned} \tag{5.11}$$

subject to, for all $h = 0, ..., N-1$,

$$\begin{aligned} \bar{\boldsymbol{x}}^{[i]}(k) - \hat{\boldsymbol{x}}^{[i]}(k) &\in \mathbb{Z}_i \\ \hat{\boldsymbol{x}}^{[i]}(k+h) - \tilde{\boldsymbol{x}}^{[i]}(k+h) &\in E_i \\ \hat{\boldsymbol{u}}^{[i]}(k+h) - \tilde{\boldsymbol{u}}^{[i]}(k+h) &\in U_i \\ \hat{\boldsymbol{x}}^{[i]}(k+h) \in \hat{\mathbb{X}}_i \subseteq \mathbb{X}_i \ominus \Big(\mathbb{Z}_i \oplus P\big(\mathbb{W} \oplus (-A_{epuck}\mathbb{V})\big)\Big) \\ \hat{\boldsymbol{u}}^{[i]}(k+h) &\in \hat{\mathbb{U}}_i \subseteq \mathbb{U}_i \ominus K_i^{aux}\mathbb{Z}_i \end{aligned}$$

and to the terminal constraint

$$\hat{\boldsymbol{x}}^{[i]}(k+N) \in \hat{\mathbb{X}}_i^F \tag{5.12}$$

6 | Simulations

In this chapter the results of a number of different simulations are reported.
Tests have been performed on the simulated model of the case study discussed in Section 3.3, in order to prove the capability of the algorithms introduced in the previous chapters to deal with communication delays and disturbances.
In the following, a visible evidence of the benefits of the DPC algorithm extended with the modifications discussed in Chapter 5 is presented.

The simulations have been carried out using the MATLAB/SIMULINK environment, extended with the usage of the TrueTime toolbox [3], which is a simulator of real-time control systems. TrueTime facilitates co-simulation of controller task execution in real-time kernels, network transmissions, and continuous plant dynamics. This toolbox makes it possible to obtain model simulations of networked control systems, by dislocating sensors, controllers and actuators, and simulate their connection through a desired network. A long list of different networks can be implemented, like Ethernet, CAN, TDMA, FDMA, Round Robin, Switched Ethernet, FlexRay and PROFINET, together with the wireless ones, like WLAN and ZigBee.
Here, TrueTime is used to simulate the delay affecting the system by realizing a fictitious network which has similar properties with respect to the ones characterizing our experimental setup. This is achieved by properly scheduling the tasks between sensors, controllers and actuators, managing the controller code execution timing.

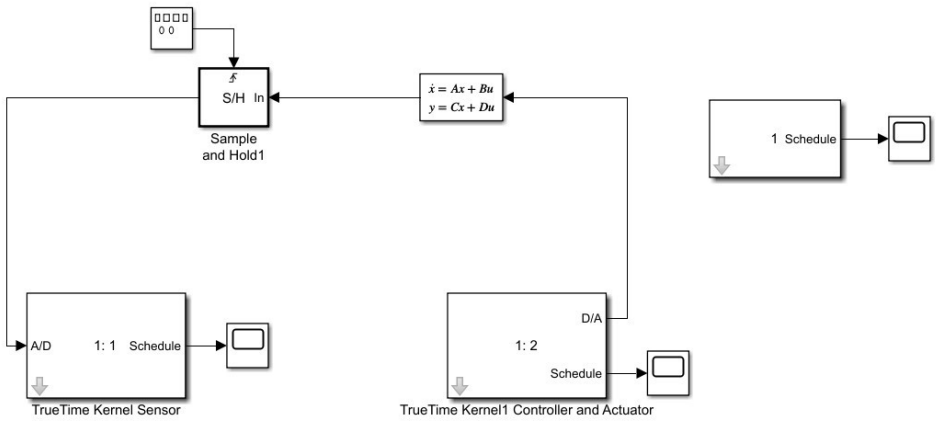


Figure 6.1: TrueTime implementation of the networked control system.

In Chapter 3 we carried out the characterization of the disturbances: we stated that the delays associated to the Bluetooth transmission and actuation of the control action are negligible compared to the ones related to the position data acquisition.
Accordingly, we decided to model the networked control system, with the actuators co-located with the controllers, neglecting the fact that the control actions are not computed directly by the E-PUCK robots. Instead, the sensors have been modeled as standalone blocks, sending the measurements trough a simulated network.
The simulated communication network has been implemented as one relying on the wireless WLAN technology (the most similar to the Bluetooth technology, among the available ones), opportunely tuned to provide the desired delay. The network has been customized to realize a stochastic delay, distributed according to the Gaussian distribution with mean of 206.2 ms, and a standard deviation of 41.4 ms.
The controller and actuator block, where the DPC algorithm has been implemented trough a MATLAB function, is activated by the kernel when a datum is received from the sensor block. In this way the real experimental setup can be simulated and different tests, with and without delays and disturbances, can be performed.

We aim to test the performances of the original DPC algorithm discussed in Chapter 4 and the ones of the robust networked DPC algorithm introduced in Chapter 5, applied to the system discussed in Section 3.3.
We expect that the modified version of the DPC algorithm, in presence of delays and disturbances affecting the system, will provide far better performances compared to the original version. To illustrate that, here are reported a series of tests executed using the original version of the algorithm, showing its limitations when non-idealties coming from the network have a significant role. Then the performances of the novel variant of the

DPC algorithm are reported, for comparison.

In the simulations reported below, it is assumed that the agents are in an initial position far from the set point, i.e., $\boldsymbol{x}_{equilibria} = \begin{bmatrix} 0 & 0 & 0 & 0 & 240 & 0 & -240 & 0 \end{bmatrix}^T$.
Moreover, it is assumed that the initial velocities of the carts among both the directions $(v_{x_{Pi}}(0),\, v_{y_{Pi}}(0))$ are equal to zero.
The controllers must stabilize the positions of the carts in a neighborhood of the equilibrium position, satisfying all the constraints related to speeds and accelerations.

6.1. Nominal system, using the original DPC algorithm

Firstly, we decided to check the performances of the original algorithm applied to the nominal system, assuming that the position data acquisition is ideal, and does not introduce delays and disturbances.

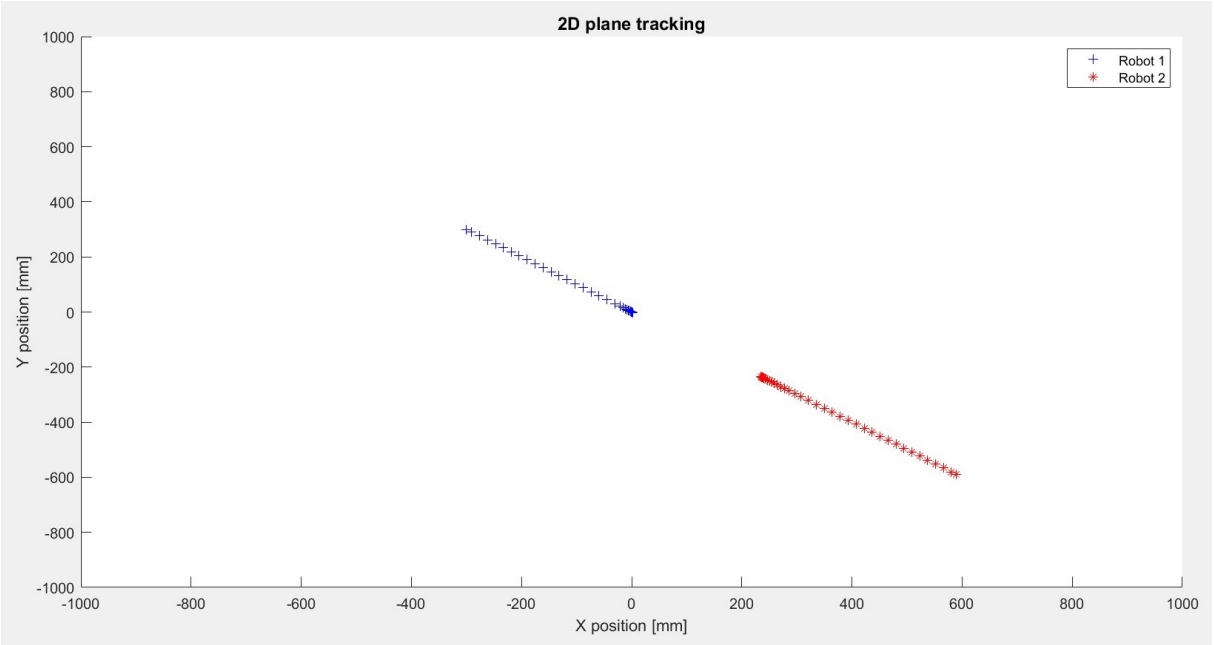


Figure 6.2: 2D tracking.

In Figure 6.2 the stabilization of the carts positions at the equilibria in the 2D plane is reported.

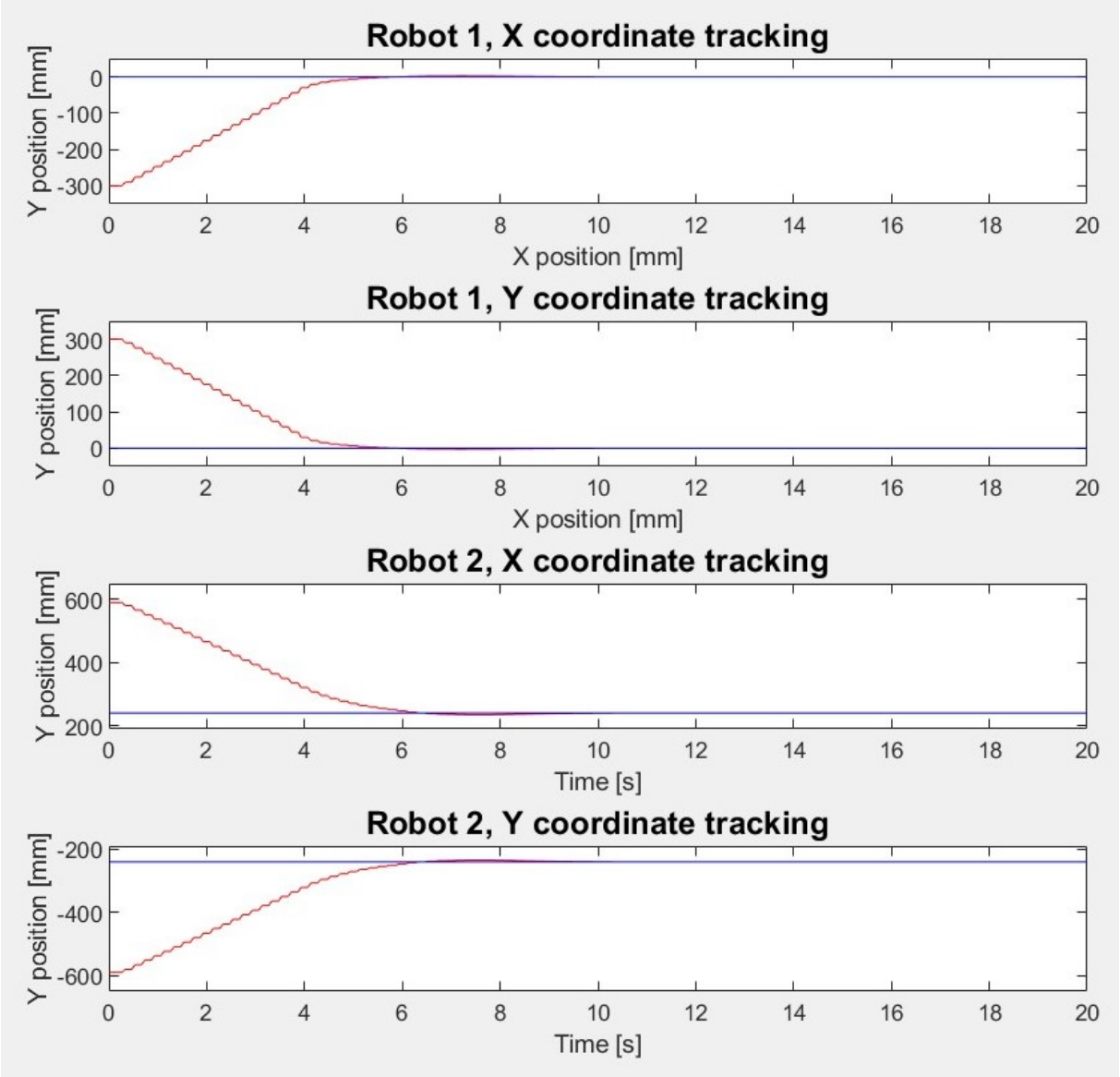


Figure 6.3: X,Y coordinate tracking.

In Figure 6.3 the position tracking along both the X and Y directions are shown.

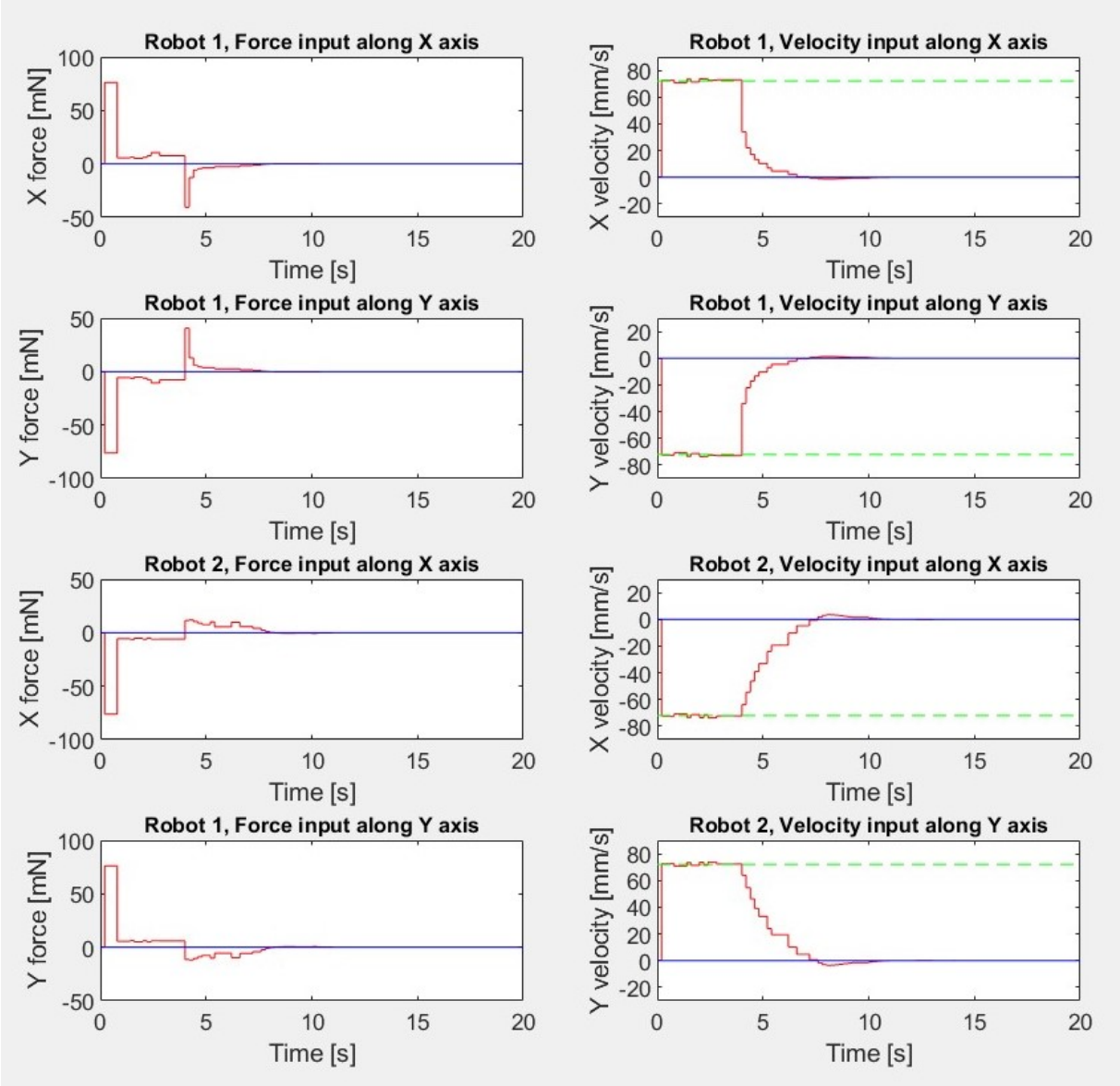


Figure 6.4: Control inputs.

In Figure 6.4 the computed control inputs, in terms of forces and velocities, are reported.

As it can be seen from the results reported in the figures above, it is clear that the original DPC algorithm can correctly stabilize the position of the carts at the equilibria.
This test proves the effectiveness of the DPC algorithm [5] applied to nominal systems, where delays and disturbances are not present. In fact, the regulation is perfect, without the presence of overshoots and oscillations.

6.2. System with delays, using the original DPC algorithm

In this simulation we decided to test the performances of the original DPC algorithm applied to the system considering the presence of the 1-step delay in the acquisition of the position measurement, characterized in Section 3.1.1.
For this test, the existence of the disturbances is not considered, to focus just on the deterioration of the performances associated with the presence of the delays.

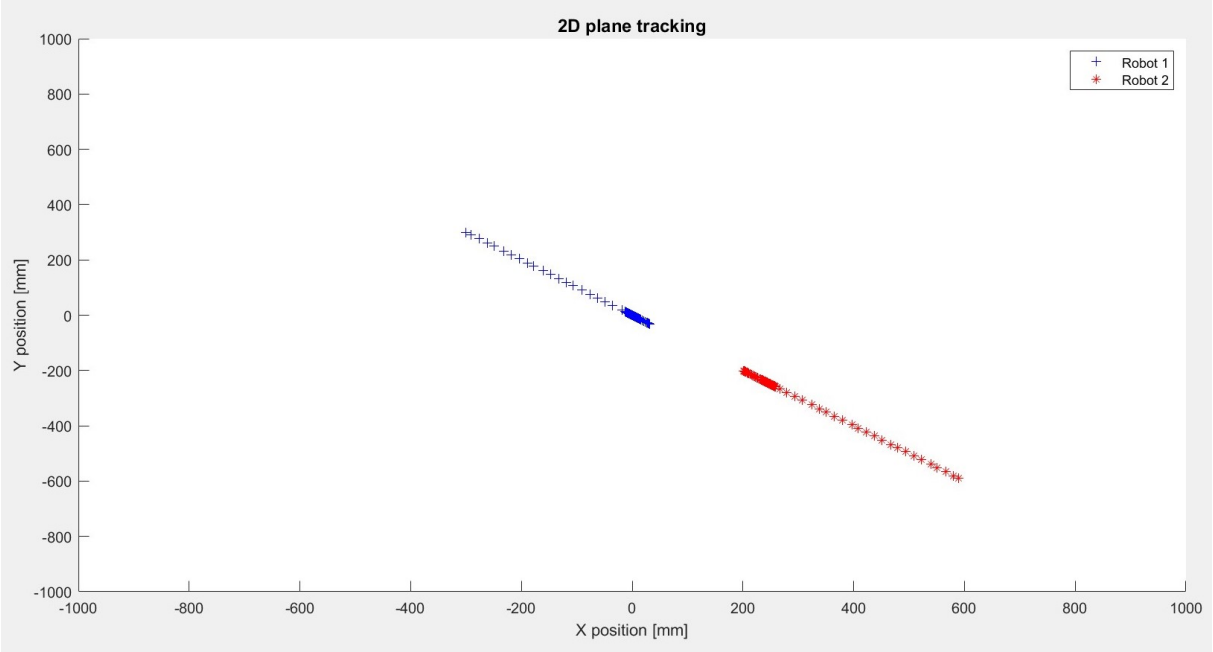


Figure 6.5: 2D tracking.

In Figure 6.5 the stabilization of the carts positions at the equilibria in the 2D plane is reported.

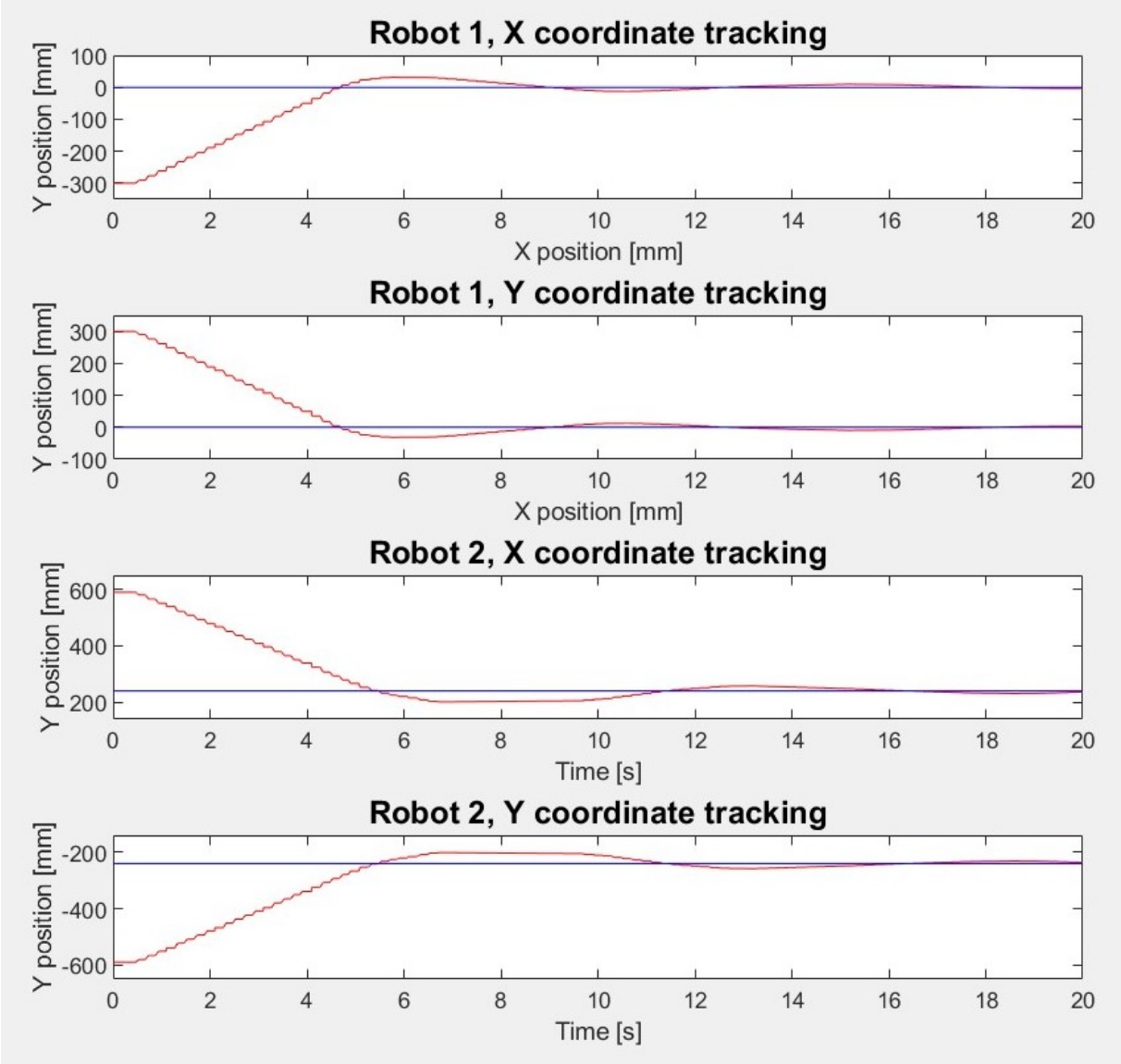


Figure 6.6: X,Y coordinate tracking.

In Figure 6.6 the position tracking along both the X and Y directions are shown.

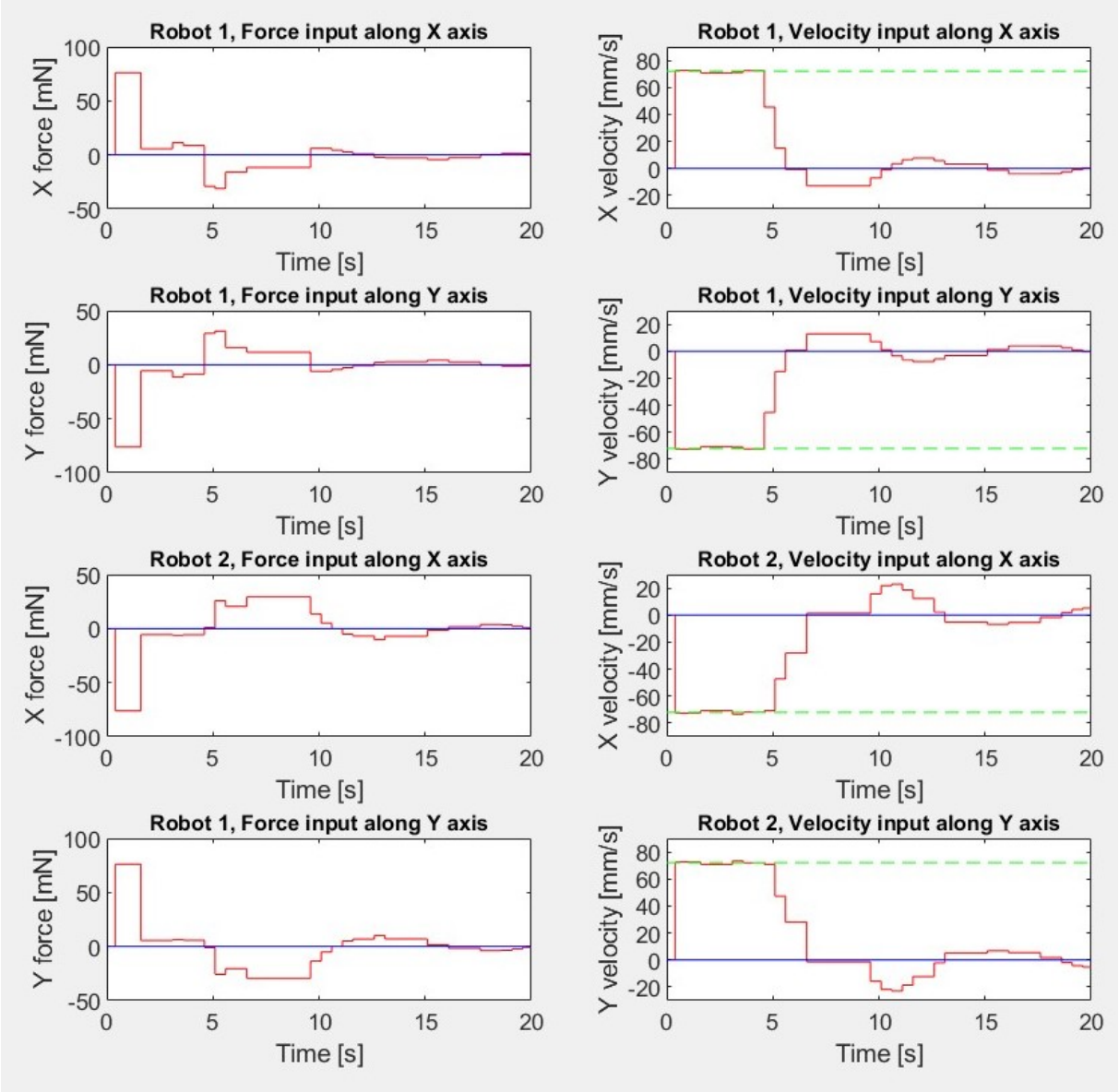


Figure 6.7: Control inputs.

In Figure 6.7 the computed control inputs, in terms of forces and velocities, are illustrated.

The results reported in the figures above show that the original DPC algorithm can stabilize the position of the carts around the equilibrium position.
However, if compared to the previous test in Section 6.1, there are significant oscillations in the tracking of the positions for both carts. Moreover, the steady state condition (null velocities) is reached with a longer settling time compared to the previous test (stated that the initial perturbation is equal among the simulation).
The results are justified by the fact that the original DPC algorithm was not developed to deal with systems affected by delays. In fact, their presence deteriorate the tracking

performances which, depending on the desired requirements, may not be satisfactory. This enforce the necessity to extend the algorithm with the modification discussed in Chapter 5.

6.3. System with delays and disturbances $w(k)$ and $v(k)$, using the original DPC algorithm

In this simulation we decided to test the performances of the original DPC algorithm applied to the system considering the presence of the noise $w(k)$ affecting the E-PUCK robots state, and the noise $v(k)$ affecting the position measurement, together with the presence of the 1-step delay introduced by the Decawave sensors.
The disturbances are realized according to the characterization accomplished in Chapter 3.

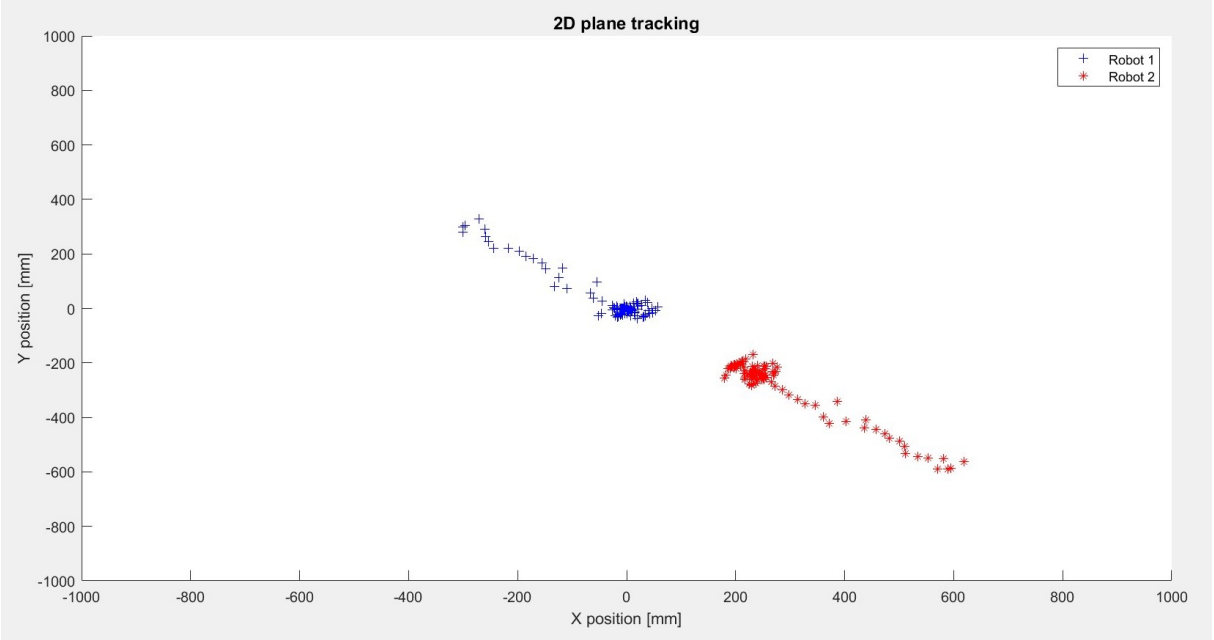


Figure 6.8: 2D tracking.

In Figure 6.8 the stabilization of the carts positions at the equilibria in the 2D plane is reported.

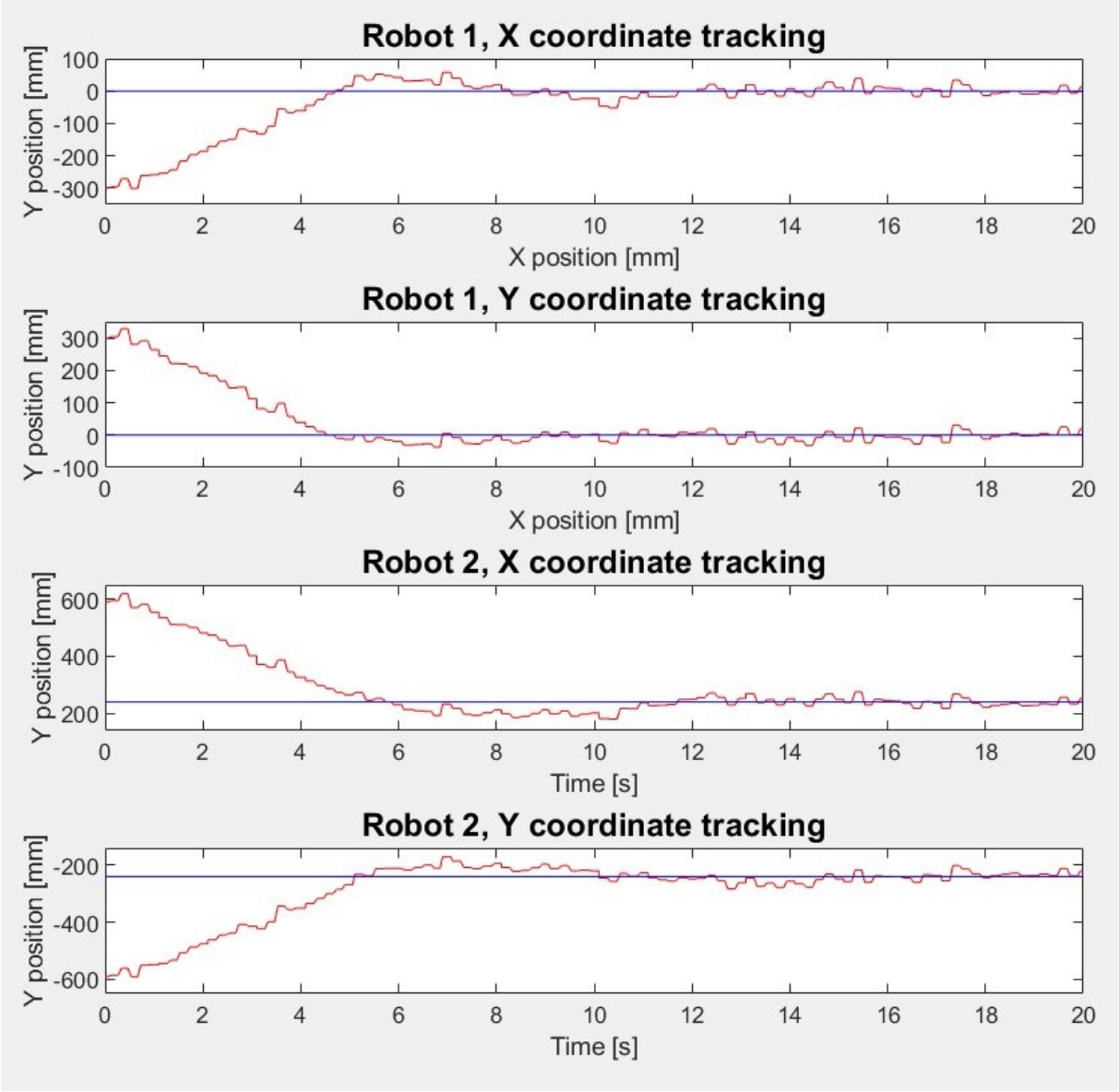


Figure 6.9: X,Y coordinate tracking.

In Figure 6.9 the position tracking along both the X and Y directions are shown.

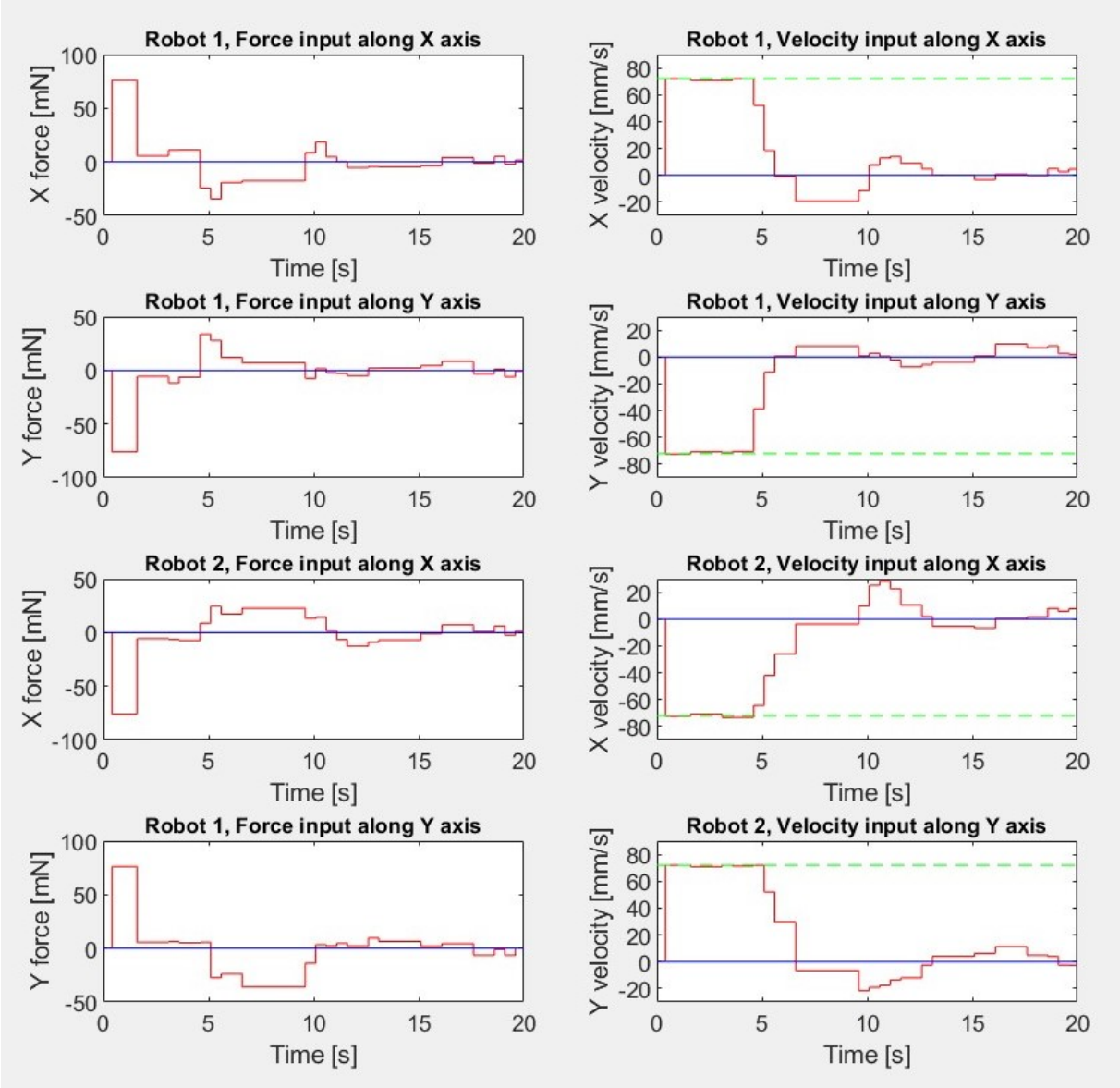


Figure 6.10: Control inputs.

In Figure 6.10 the computed control inputs, in terms of forces and velocities, are illustrated.

The results reported in the figures above show that the original DPC algorithm can stabilize the position of the carts at the equilibria, despite the presence of the disturbances and the delays. However, similarly to the test reported in Section 6.2, the performances deteriorate. In fact, the presence of overshoots and oscillations in the tracking of the reference positions results in a longer settling time.

In order to provide results closer to the ones obtained from test 6.1 also in presence of non-idealties, we need to use the robust networked DPC scheme discussed in Chapter 5.

6.4. System with delays and disturbances $w(k)$ and $v(k)$, using the robust networked DPC algorithm

In this simulation we tested the performances of the modified DPC algorithm presented in Chapter 5 applied to the system introduced in Section 3.3, considering the presence of the disturbance $w(k)$ affecting the E-PUCK robots state, the noise $v(k)$ affecting the position measurement, and the 1-step delay introduced by the Decawave sensors.
This simulation is performed in the same conditions as the one reported in Section 6.3. In this way, it is possible to make a comparison between the performances provided by the two DPC algorithms.

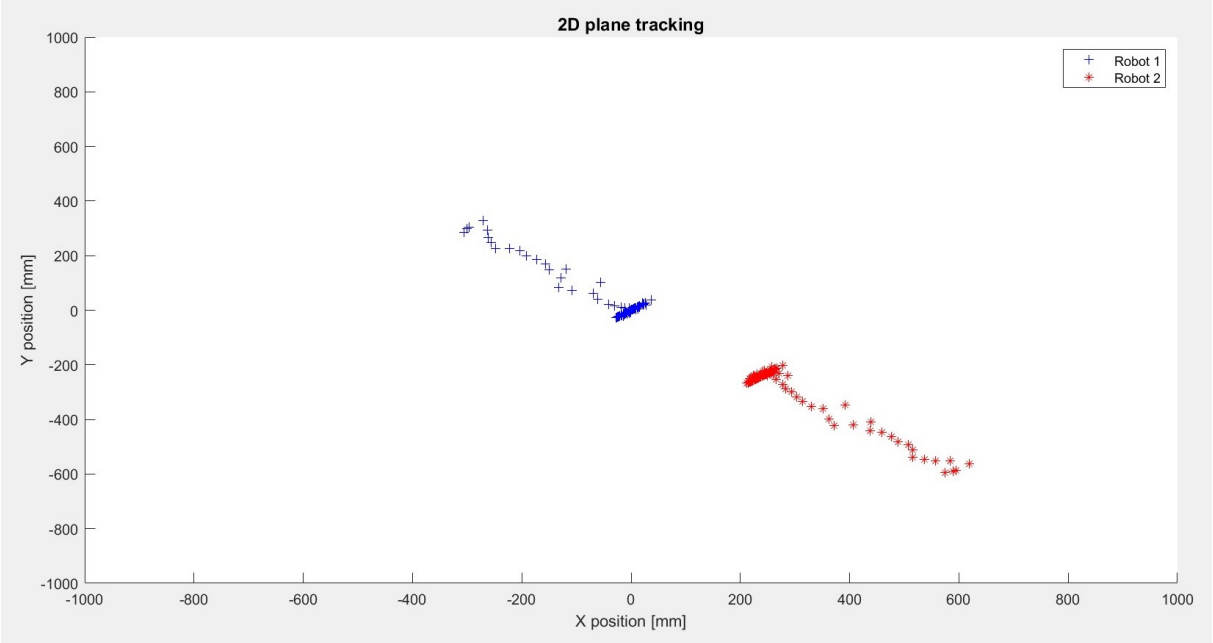


Figure 6.11: 2D tracking.

In Figure 6.11 the stabilization of the carts positions at the equilibria in the 2D plane is reported.

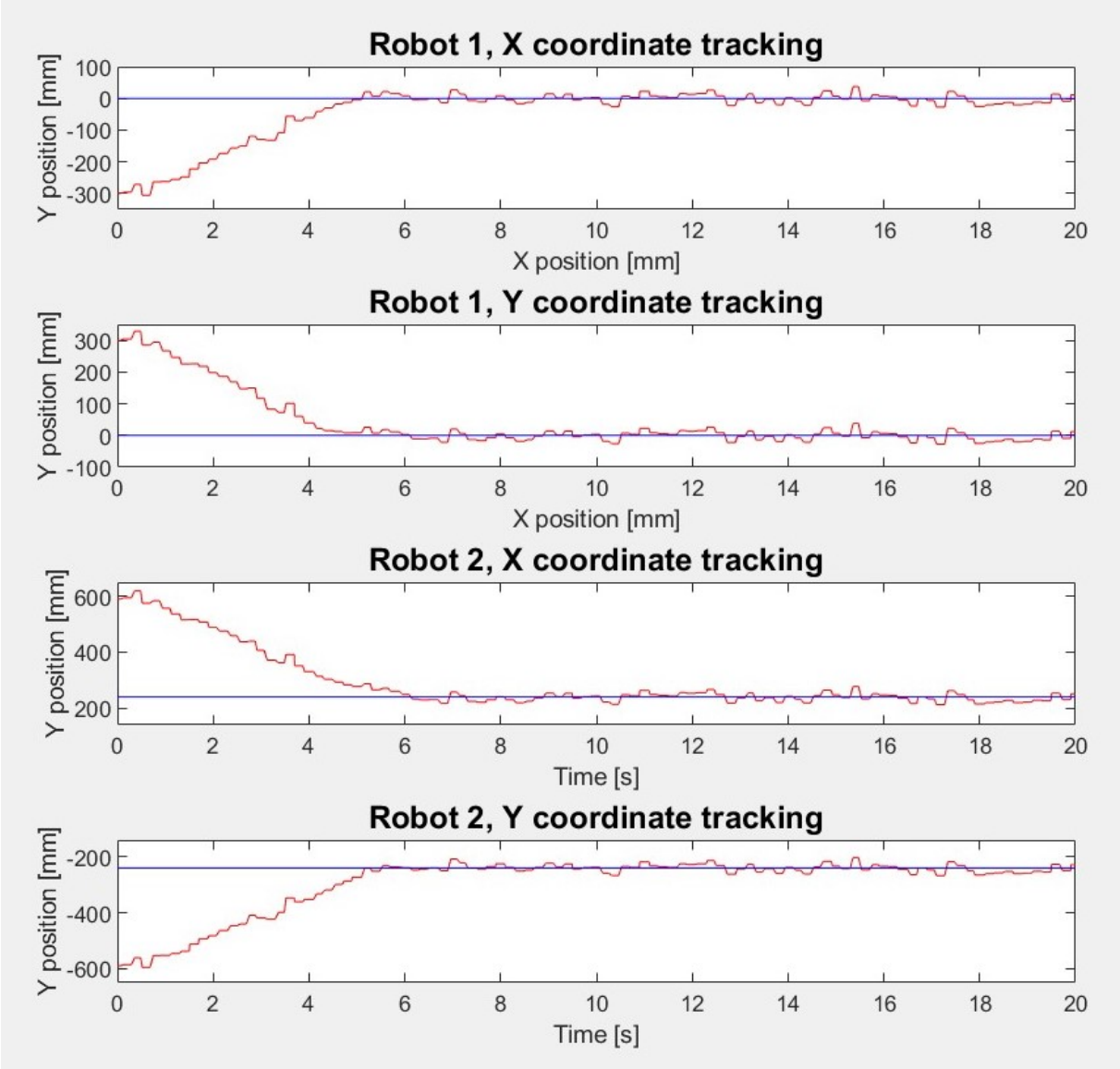


Figure 6.12: X,Y coordinate tracking.

In Figure 6.12 the position tracking along both the X and Y directions are shown.

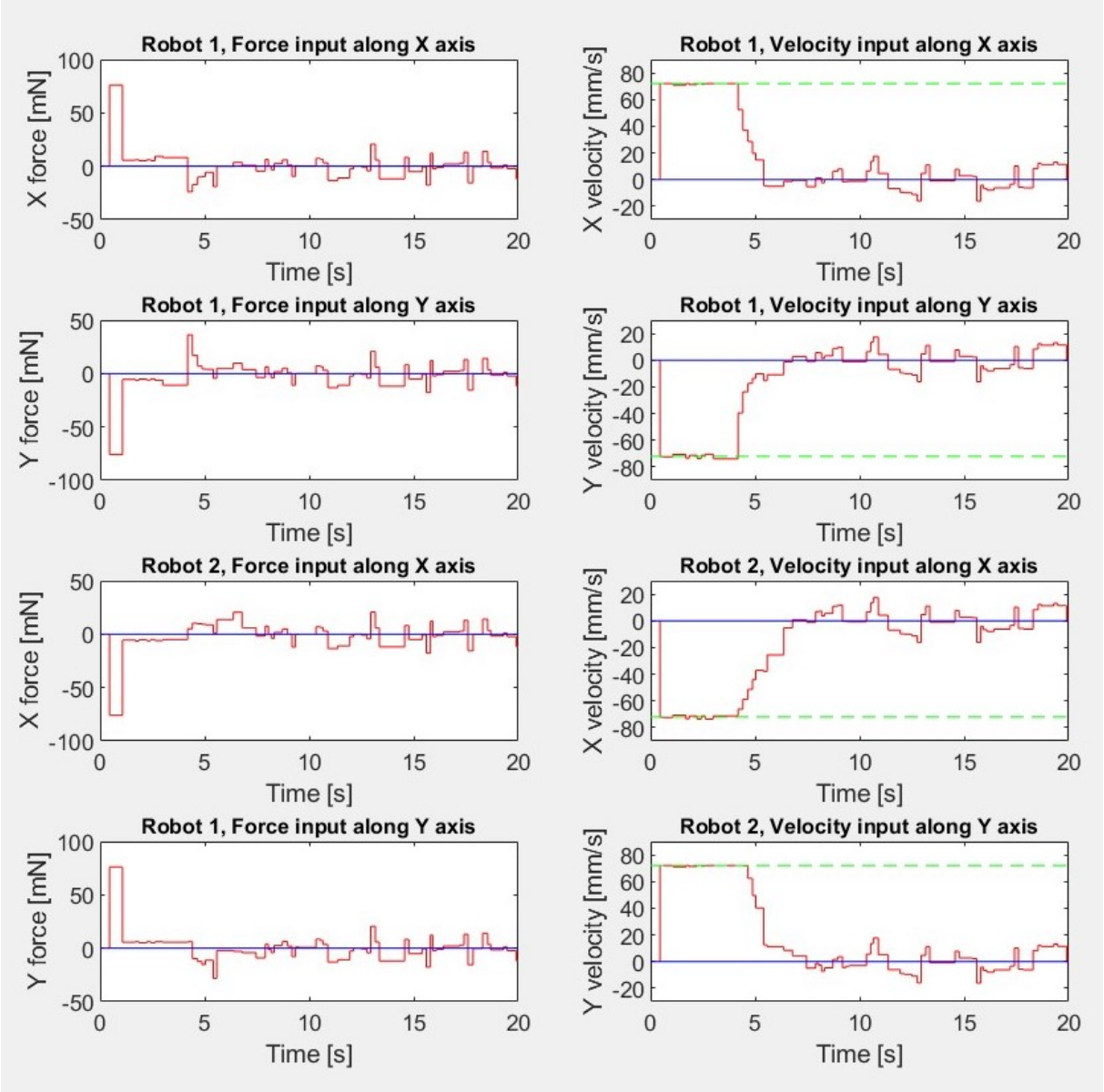


Figure 6.13: Control inputs.

In Figure 6.13 the computed control inputs, in terms of forces and velocities, are reported.

The results of this simulation show that the novel version of the DPC algorithm is capable to stabilize the E-PUCK robots position in a neighborhood of the equilibria.
Comparing the results in Figure 6.12 with the ones reported in Figure 6.9, it is clearly visible that overshoots and oscillations introduced by the usage of the original strategy to control the system replica, vanish with the adoption of the novel, robust networked version of the DPC algorithm.
The performances obtained in the simulation here reported are similar to the one shown

in test 6.1, where the algorithm [5] is applied to the nominal system.

The usage of the predictor described in Chapter 5 allows to mitigate for the effects of the delay related to the position data acquisition, guaranteeing better performances.
In fact, the elimination of overshoots and oscillations provides results that are satisfactory for most of the applications. Moreover, it results in a shorter settling time.

6.5. Robust networked DPC algorithm applied to path tracking

A final simulation aims to show the capability of robust networked DPC to make the carts follow a certain path, stabilizing the position of the agents in several different configurations which are far from the equilibria. In fact, by slightly modifying the algorithm, it is possible to have the carts follow a desired trajectory.

It is sufficient to modify the stage and final costs $l_i(\hat{\boldsymbol{x}}^{[i]}, \hat{\boldsymbol{u}}^{[i]})$ and $V_i^F(\hat{\boldsymbol{x}}^{[i]})$ of the cost function introduced in (5.11).
They must be redefined considering the new goal position of the i-th subsystem, defined as $\boldsymbol{x}_{goal}^{[i]}$.

$$V_i^F(\hat{\boldsymbol{x}}^{[i]}) = \left\| \hat{\boldsymbol{x}}^{[i]} - \boldsymbol{x}_{goal}^{[i]} \right\|_{P_i^0}^2$$
$$l_i(\hat{\boldsymbol{x}}^{[i]}, \hat{\boldsymbol{u}}^{[i]}) = \left\| \hat{\boldsymbol{x}}^{[i]} - \boldsymbol{x}_{goal}^{[i]} \right\|_{Q_i^0}^2 + \left\| \hat{\boldsymbol{u}}^{[i]} \right\|_{R_i^0}^2$$

Moreover, the constraint related to the final position (5.12) must be redefined, since the final state $\hat{\boldsymbol{x}}^{[i]}(k+N) - \boldsymbol{x}_{goal}^{[i]}$ must lie inside the terminal set $\hat{\mathbb{X}}_i^F$.

$$\hat{\boldsymbol{x}}^{[i]}(k+N) - \boldsymbol{x}_{goal}^{[i]} \in \hat{\mathbb{X}}_i^F$$

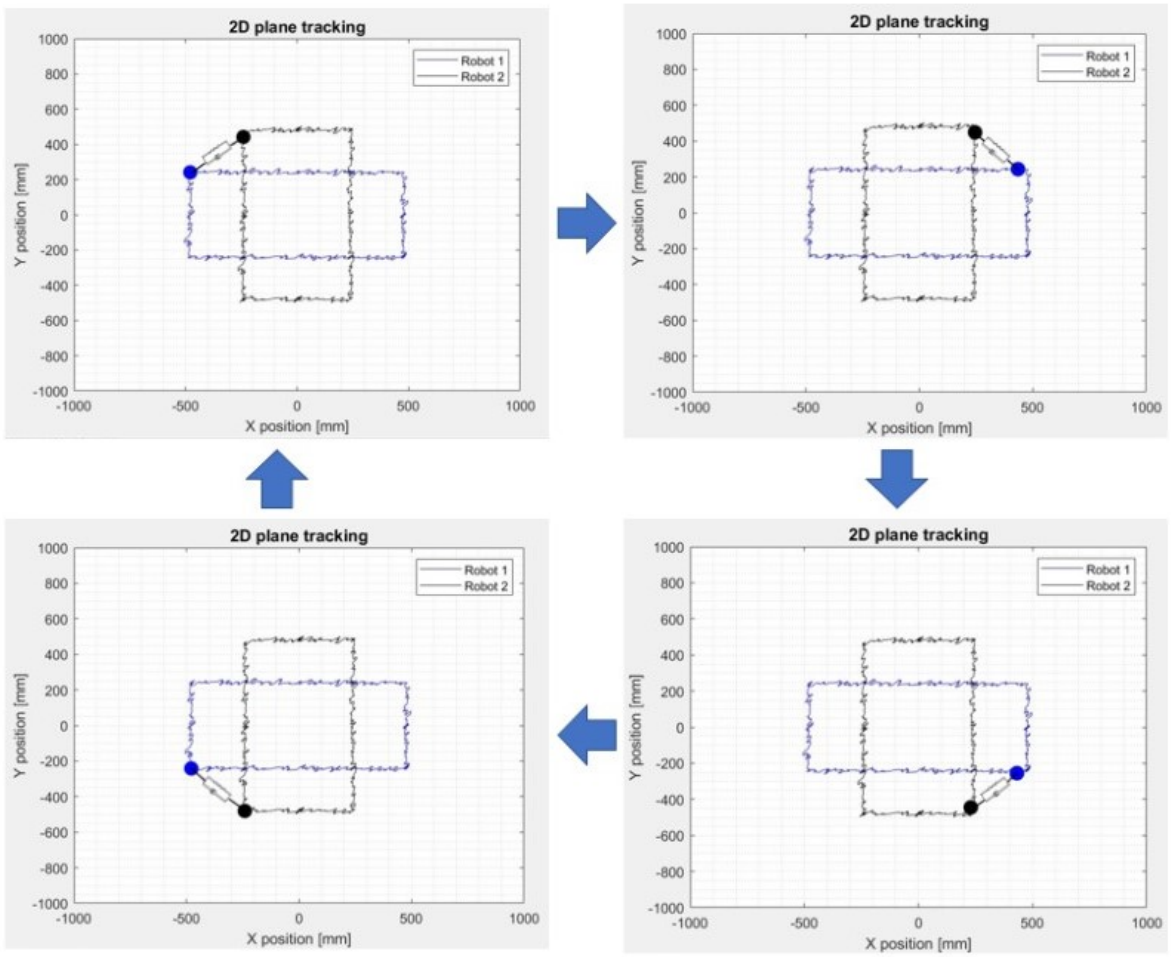


Figure 6.14: Path tracking.

The results reported in Figure 6.14 show that the robust networked DPC algorithm is capable to stabilize the positions of the carts in configurations which are different from the equilibria.

7 | Experimental results

In this chapter we report the experimental results validating robust networked DPC applied to our experimental setup.
Indeed, tests have been carried out on the real experimental setup described in Chapter 2, where the position measures are provided by the Decawave sensors, and the carts are represented by the E-PUCK robots virtually connected through a spring and a damper.

The experimental results here reported aim to show the ability of the modified version of the algorithm discussed in Chapter 5 to regulate the positions of the E-PUCK robots to the equilibria, regardless of the presence of delays and disturbances introduced by the usage of a networked control system architecture.

During the execution of the tests on the experimental setup, the limitation of the devised configuration emerged.
In fact, as discussed in Section 2.2, the arena has been placed directly on the ground, since its dimensions do not allow to locate it on a desk. This solution is really practical and allow for portability of the setup, but necessitates to have a flat floor, without irregularities. Otherwise, the tiny wheels of the robot can start sliding, and the robot can stop its motion for few instants.
This problem has been encountered many times in the realization of the tests on the experimental setup, because, as can be seen in Figure 2.1, the floor of the laboratory is not completely flat.

During the characterization of the noise affecting the E-PUCK robot state (Section 3.2), this phenomena does not appear to be relevant. In fact, during the realization of linear segments, it never happens that the E-PUCK robot completely stops. For this reason it was assumed that the laboratory floor surface was suited for our purposes.
Unfortunately, in the experimental validation phase, we noticed that when the robots begin rotating to reach the goal position, the wheels often start sliding when they run into some irregularity of the floor.
This issue can be seen as an additional disturbance affecting the system, which contributes compromising the execution of the tests when the original algorithm was used. In fact,

the RPI sets computed for the control strategy [5], developed for nominal models not considering the presence of disturbances, are smaller if compared to the ones of the novel version of the algorithm (as discussed in Section 5.2).
Therefore, the presence of the delays and the disturbances, together with the other non-idealities affecting the experimental setup (mainly the sliding of the wheels phenomena previously described), result in the fact that the controllers couldn't provide an online solution without violating the constraints.
Consequently, the optimization problem, solved using the *quadprog* MATLAB function, lost its feasibility, and the controllers failed their task to conduct the robots to the equilibrium position.
For this reason, we cannot report successful tests where the controllers developed using the original version of the algorithm discussed in Chapter 4 correctly regulate the position of the E-PUCK robots in a neighborhood of the equilibria.

The robust networked DPC algorithm, instead, proved its effectiveness in these conditions, when delays and disturbances have a significant relevance on the system.
Several tests have been performed, assuming that an initial perturbation has been applied to the system, where the robots were placed in a position which was different from the equilibrium one. Moreover, similar to the simulations reported in the previous chapter, it was assumed that the initial velocities of the robots $v_{x_{Pi}}(0)$ and $v_{y_{Pi}}(0)$ were equal to zero.

Here we report two tests, showing the ability of the modified DPC algorithm to regulate the positions of the robots at the equilibria, starting from different positions and robot orientations, located in the reachable area inside the arena.

7.1. Experimental test with robots starting from opposite positions

Firstly, we report the test where the carts have been placed at the limits of the arena, oriented facing each other ($\vartheta_1 = \pi/2$ and $\vartheta_2 = 0$).

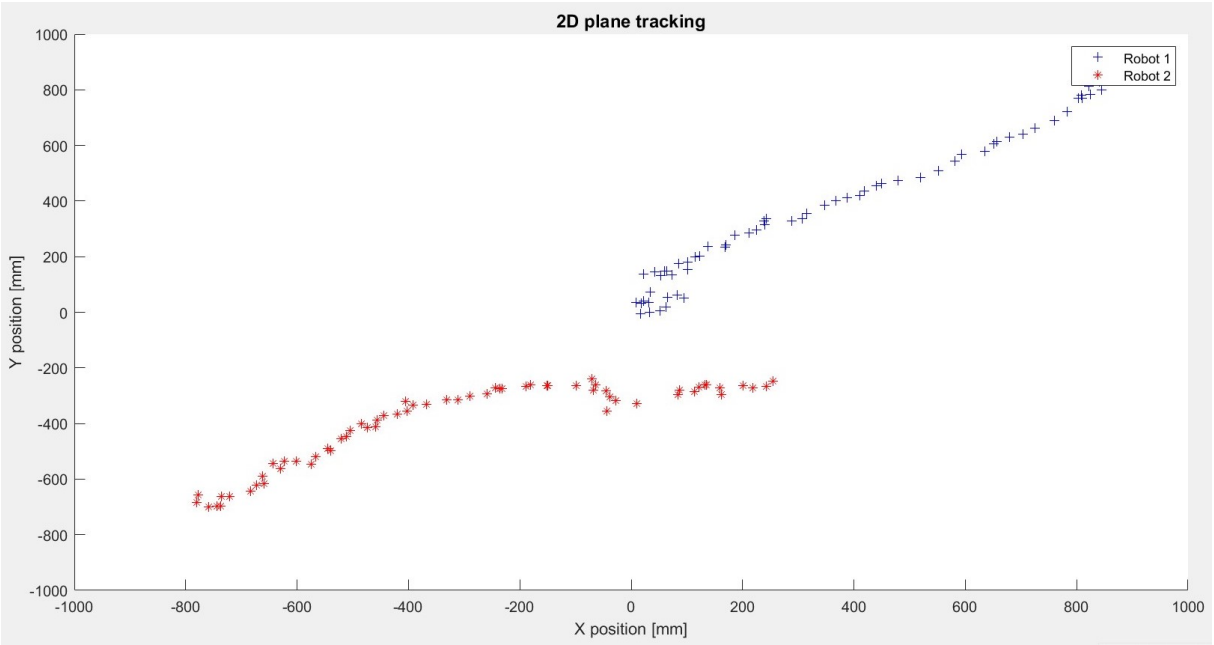


Figure 7.1: 2D tracking.

In Figures 7.1 the stabilization of the carts positions at the equilibria in the 2D plane is reported.

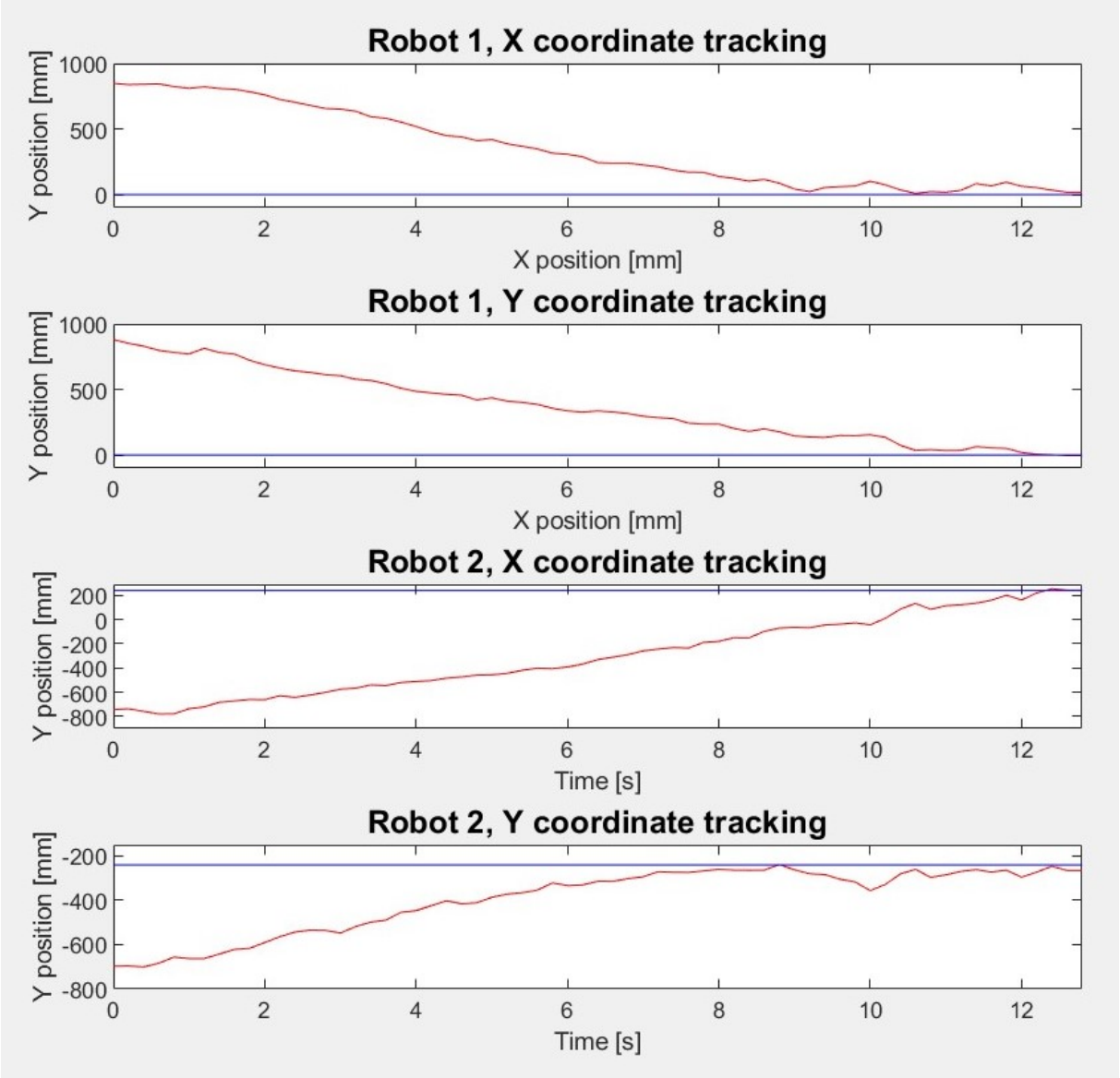


Figure 7.2: X,Y coordinate tracking.

In Figure 7.2 the position tracking along both the X and Y directions are shown.

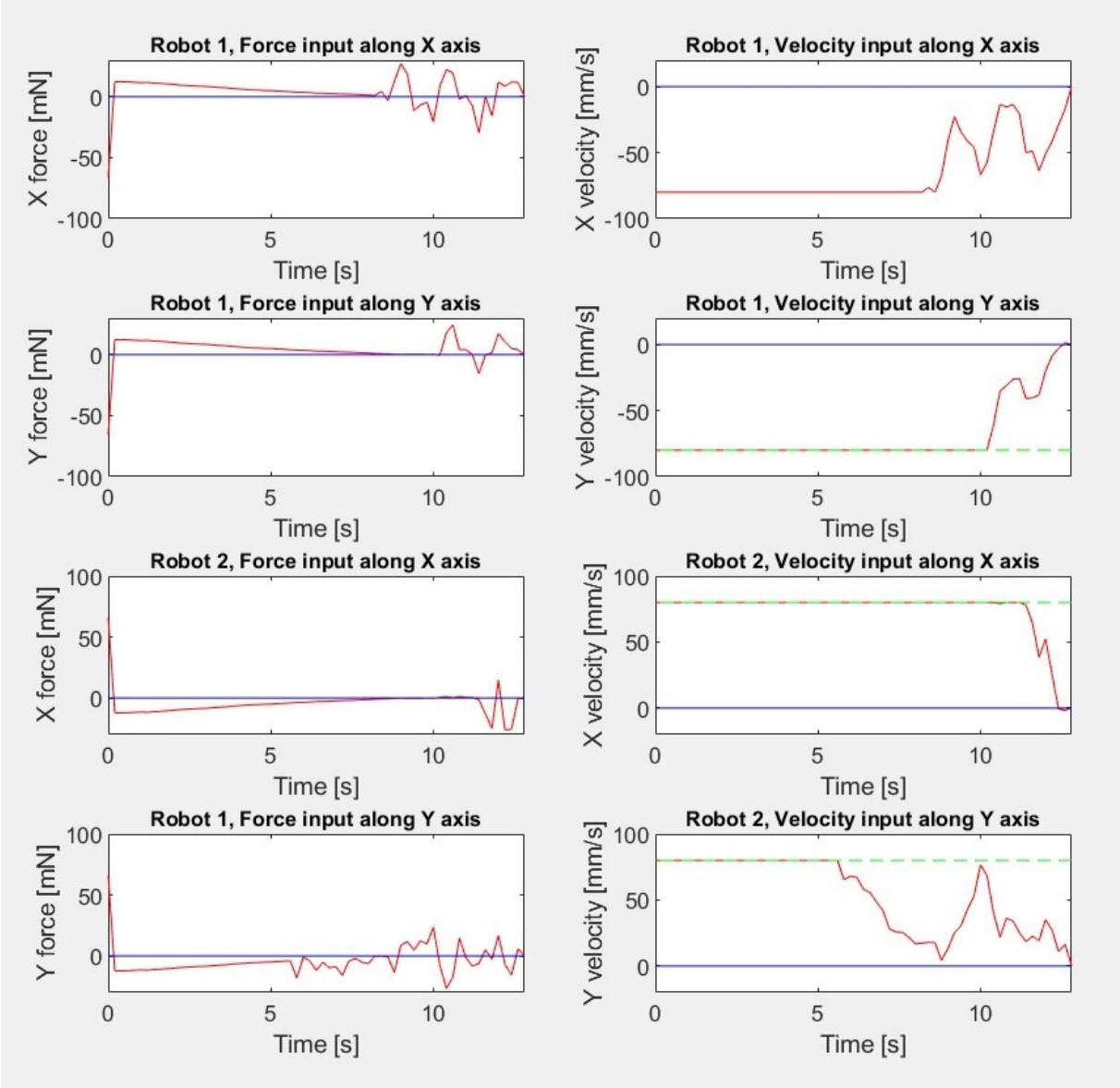


Figure 7.3: Control inputs.

In Figure 7.3 the computed control inputs, in terms of forces and velocities, are reported. Some comments are provided in Section 7.3.

7.2. Experimental test with robots starting from adjacent positions

In this test we report the regulation of the robots positions in a neighborhood of the equilibria, starting from the limits of the arena, with the robots oriented facing the same direction ($\vartheta_1 = 0$ and $\vartheta_2 = 0$).

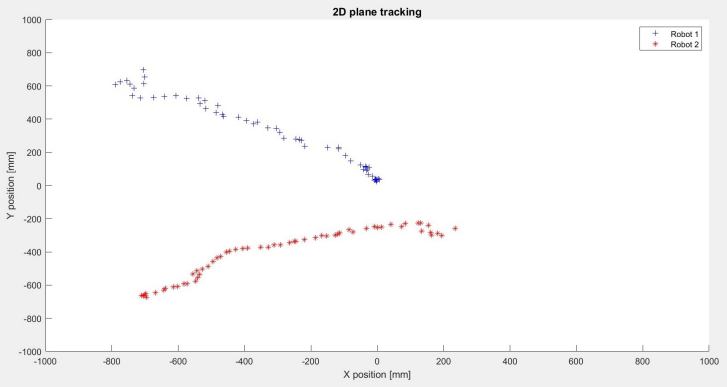


Figure 7.4: 2D tracking.

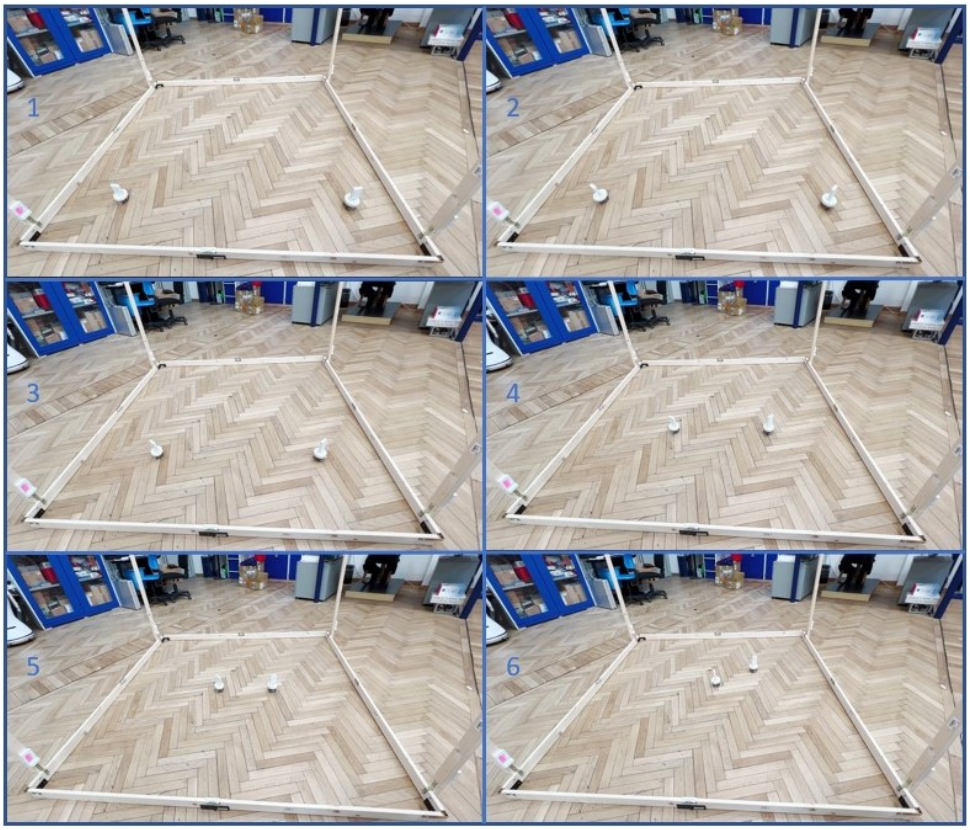


Figure 7.5: Experimental test.

In Figure 7.4 and 7.5 the stabilization of the carts positions at the equilibria in the 2D plane is reported.

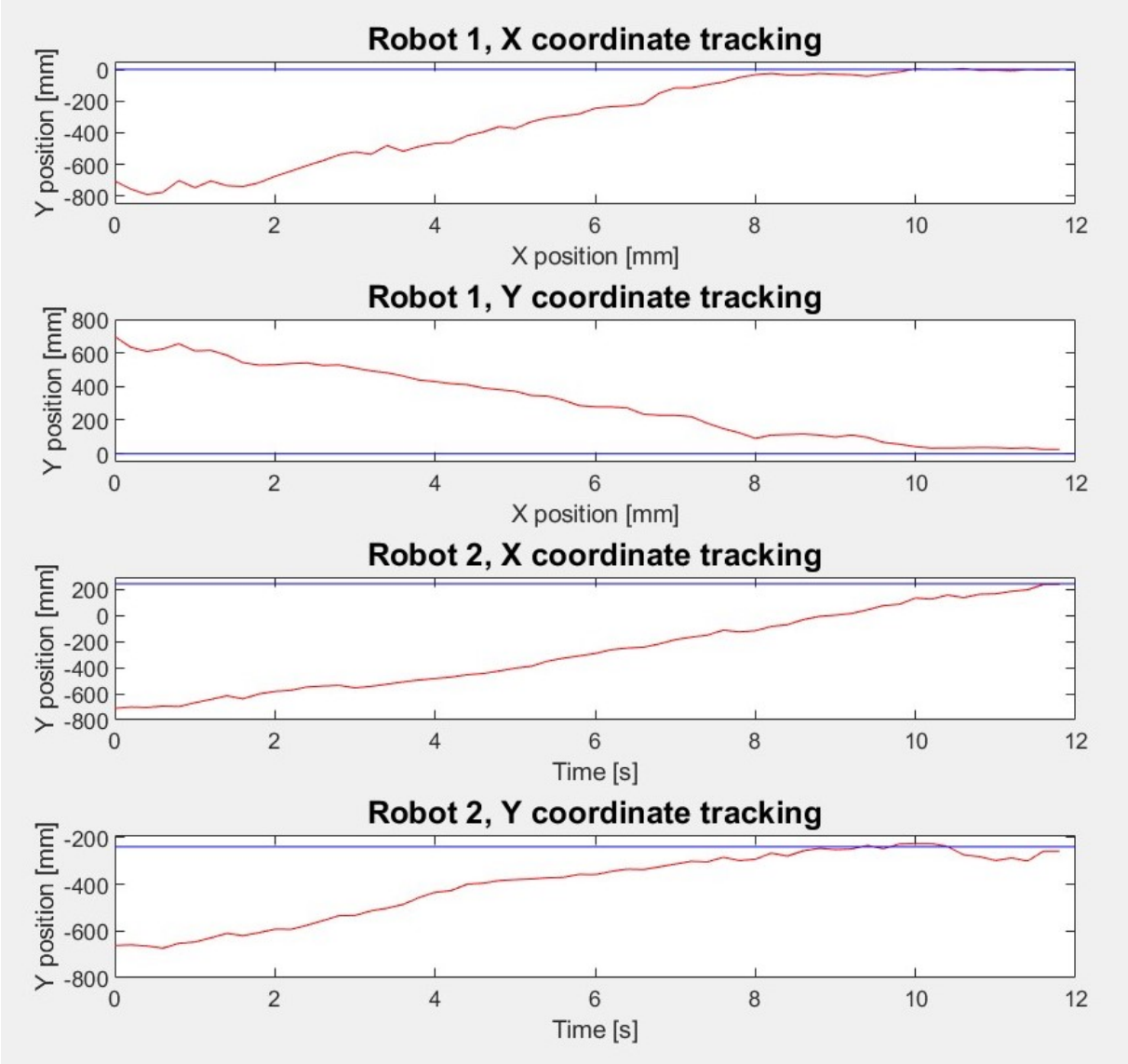


Figure 7.6: X,Y coordinate tracking.

In Figure 7.6 the position tracking along both the X and Y directions are shown.

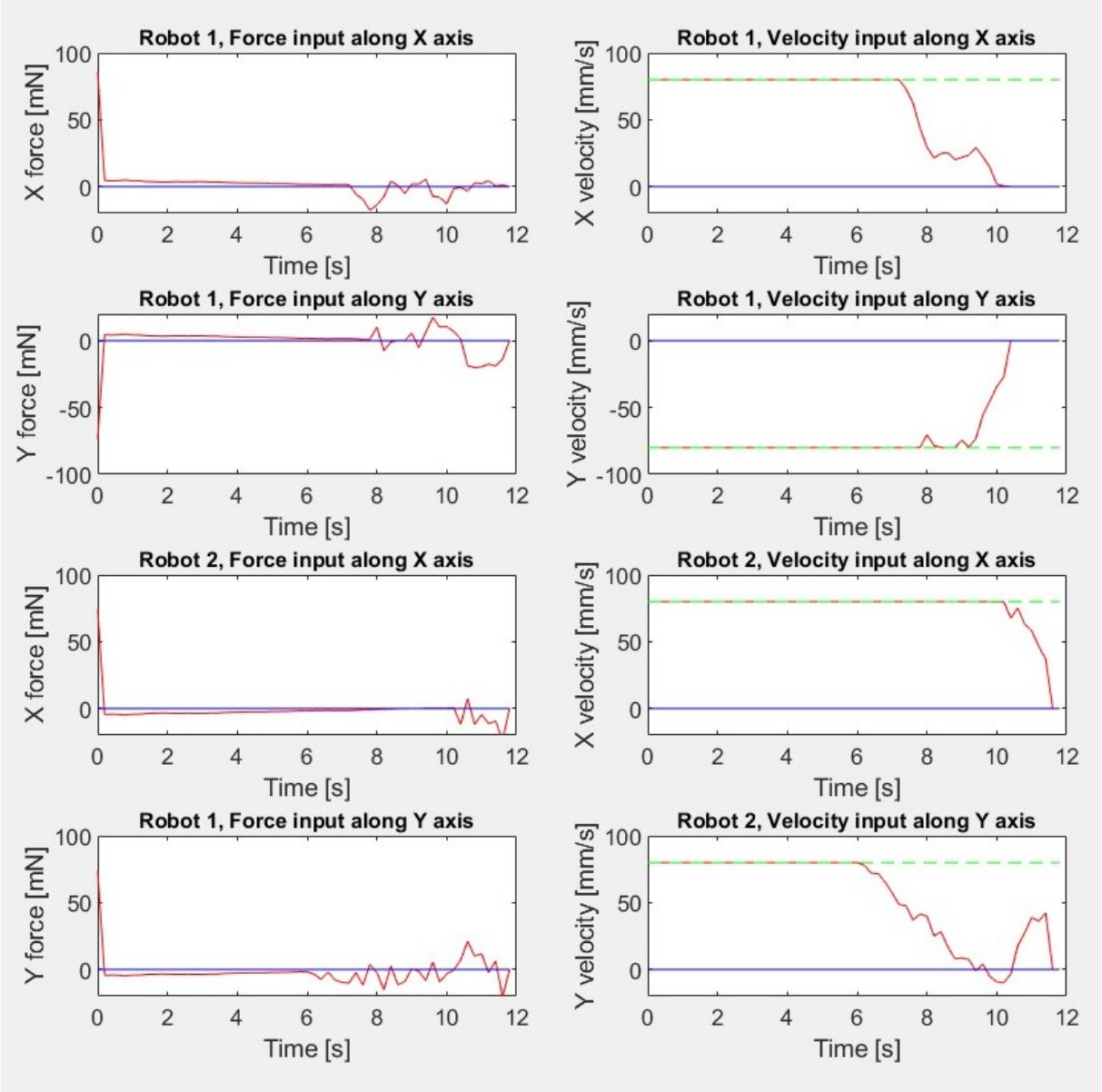


Figure 7.7: Control inputs.

In Figure 7.7 the computed control inputs, in terms of forces and velocities, are reported. Some comments are provided in Section 7.3.

7.3. Considerations on the experimental results

The results of the tests reported in sections 7.1 and 7.2 show performances which are similar to the expected ones obtained in the simulation described in section 6.4.
In fact, the robust networked DPC algorithm is capable to correctly regulate the position of the robots at the equilibria, respecting the constraints on the velocities and the accelerations, despite the presence of delays and disturbances.

As expected, there were no oscillations and overshoots in the tracking of the X,Y coordinates as it can be seen in Figures 7.2 and 7.6.
However, from Figures 7.3 and 7.7, the effects of the sliding of the wheels phenomena, previously discussed, can be noticed. In fact, the spikes in the control inputs (forces and velocities) were related to the time instants when the robot got stuck for few moments in some irregularity of the floor. Nonetheless, this was not a problem and the robust controllers were always capable to find a feasible solution, thanks to the large RPI sets defined in Section 5.2.

These results validate the expected performances obtained in simulation, and show the suitability of the modification made to the DPC algorithm discussed in Chapter 5.
In particular, these tests witness the fact that the original algorithm, developed for nominal models, was not capable to control the real system in presence of non-idealties typical of networked control system. On the other hand, the variant presented in this thesis has proven to be perfectly capable to deal with delays and disturbances, providing satisfactory performances.

8 | Conclusions and future developments

In this thesis we developed a lightweight experimental setup subject to non-idealties similar to the ones introduced by the adoption of networked control architectures. This setup can be used as benchmark for testing hierarchical and distributed control strategies in multi-agent networked applications.
The used laboratory equipment has been analyzed in details and a complete characterization of the noises and delays introduced by each device has been completed. Moreover, a calibration strategy improving the measurements provided by the real time positioning system set up by Decawave UWB sensors is presented.

A state-of-the-art distributed model predictive control strategy, named DPC [5], has been implemented and tested on the multi-agent case study, composed by carts moving on a bi-dimensional plane, virtually connected by springs and dampers.
Several simulations have been carried out, considering the presence of noises and delays compatible with the ones characterizing the real experimental setup. The outcomes of these tests show the deterioration of the regulation performances when network non-idealities are present. In particular, we noticed that undesired phenomena like overshoots and oscillations arise.

To overcome such limitation, we developed a new, robust, and networked version of the DPC algorithm, capable to deal with delays and disturbances. In particular, we equipped the controller with a predictor, to compensate for the effects of the delays. Moreover, the robust positive invariant sets constraining the optimization variables have been re-designed to consider the presence of the noise $w(k)$ affecting the system state, the noise $v(k)$ affecting the measures, and the error due to the predictions computed to compensate for the delay.
The effectiveness of the novel algorithm has been first tested in simulation. The results show that overshoots and oscillations in the regulation of the carts positions in a neighborhood of the equilibria, vanish with the adoption of the robust and networked version

of the DPC algorithm.

Finally, the results obtained in simulation were validated in tests on the experimental setup consisting of carts virtually connected by springs and dampers. The carts are represented by E-PUCK unicycle robots and their position is obtained by UWB positioning sensors. Connections between robots have been virtually imposed by software, enforcing couplings between agents.
Experimental tests confirmed the results obtained in simulation, witnessing that the original algorithm was not suitable to deal with network-induced non-idealities. At the same time, they show that the robust networked DPC algorithm here presented was capable to provide good performances also in presence of delays and disturbances.

Possible future developments of this work can focus on improving the experimental setup, e.g., by using a larger number of agents.

In this work we mainly focused on the delays related to the position measurements acquisition, since they were much larger than the ones related to the Bluetooth transmission between robots. However, it is well known that in real networked applications, delays in the communication between subsystems can become significant, and can alter the performances of the regulators.
For this reason, future work can consider to account also for these non-idealities by evaluating the effects that they introduce.

Moreover, in this work we developed a variant of the DPC algorithm that perfectly fits the developed experimental setup, considering the presence of the 1-step delay in data acquisition and the necessity to introduce a 1-step ahead predictor. A potential future work can aspire to devise a more general version of the proposed algorithm, which is possibly capable to handle the presence of multi-step delays, possibly acting also on the communication channels between agents.

Bibliography

[1] C. Bisdikian. An overview of the bluetooth wireless technology. *IEEE Communications Magazine*, 39(12):86–94, 2001.

[2] R. F. Brena, J. P. García-Vázquez, C. E. Galván-Tejada, D. Muñoz-Rodriguez, C. Vargas-Rosales, and J. Fangmeyer. Evolution of indoor positioning technologies: A survey. *Journal of Sensors*, 2017.

[3] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén. How does control timing affect performance? *Control Systems Magazine*, 23(3):16–30, 2003.

[4] Decawave. Api guide. URL `https://www.decawave.com/product/dwm1001-development-board/`.

[5] M. Farina and R. Scattolini. Distributed non-cooperative mpc with neighbor-to-neighbor communication. *IFAC Proceedings Volumes*, 44(1):404–409, 2011. 18th IFAC World Congress.

[6] M. Farina, P. Colaneri, and R. Scattolini. Block-wise discretization accounting for structural constraints. *Automatica*, 49(11):3411–3417, 2013.

[7] Y. Gao. Multi-robot cooperative control with uwb localization technology. *Master thesis*, Politecnico di Milano, academic year 2020-2021.

[8] M. Herceg, M. Kvasnica, C. Jones, and M. Morari. Multi-Parametric Toolbox 3.0. In *Proc. of the European Control Conference*, pages 502–510, Zürich, Switzerland, July 17–19 2013.

[9] A. Jiménez and F. Seco. Comparing ubisense, bespoon, and decawave uwb location systems: Indoor performance analysis. *IEEE Transactions on Instrumentation and Measurement*, PP:1–12, 04 2017.

[10] J. Löfberg. Yalmip : A toolbox for modeling and optimization in matlab. In *In Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

[11] P. Marshall. A comprehensive guide to industrial networks. URL https://www.perrymarshall.com/articles/industrial/part-1/.

[12] D. Mayne, M. Seron, and S. Raković. Robust model predictive control of constrained linear systems with bounded disturbances. *Automatica*, 41(2):219–224, 2005.

[13] L. Molu. Optimal control: Model predictive controllers. URL https://scriptedonachip.com/Optimal-Controllers-MPC.

[14] S. Rakovic, E. Kerrigan, K. Kouramas, and D. Mayne. Invariant approximations of the minimal robust positively invariant set. *IEEE Transactions on Automatic Control*, 50(3):406–410, 2005.

[15] R. Scattolini. Architectures for distributed and hierarchical model predictive control – a review. *Journal of Process Control*, 19(5):723–731, 2009.

[16] N. Semiconductors. Mma7260qt datasheet. URL https://www.nxp.com/docs/en/data-sheet/MMA7260QT.pdf.

A | Appendix A

C code running on the E-PUCK robots, implementing the feedback linearization.

```
#include "xc.h"
#include "e_epuck_ports.h"
#include "e_init_port.h"
#include "e_led.h"
#include "e_motors.h"
#include "e_uart_char.h"
#include "matlab.h"
#include "utility.h"
#include "e_acc.h"
#include "e_ad_conv.h"
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

//define constants with global visibility
#define PI 3.14159265358979f
#define STEPST02PI 1300
#define T_SPEED 300

//define the timing to be in seconds
#ifdef CLOCKS_PER_SEC
#undef CLOCKS_PER_SEC
#endif
#define CLOCKS_PER_SEC FCY

int main(void)
{
// variables definition
int selector; //selector position
time_t timer1; //timing variable
char car; //variable to get new data from MATLAB
```

```
35 int dato[4];
36 int speedl; //speed of the left wheel in step/s
37 int speedr; //speed of the right wheel in step/s
38 long i;
39 float ecall = 1.0f; //distance epsilon of point P for the feedback linearization
40 float E = 5.2f;
41 float R = 2.05f;
42 float kk = 500.0f/PI; // step/rad
43 float vR;
44 float vL;
45 float wR;
46 float wL;
47 float w;
48 float vxp=0.0f, vyp=0.0f, v = 0.0f, theta = 0.0f;
49 int x, y, z;
50 TypeAccRaw ACC; // accelerations
51 float tau = 0.01f; //internal sampling time
52
53 //initialization of the input/output ports, the UART module, the motors, and accelerometer
54 e_init_port();
55 e_init_uart1();
56 e_init_motors();
57 e_set_speed_left(0);
58 e_set_speed_right(0);
59 e_init_ad_scan(ALL_ADC);
60 e_acc_calibr();
61
62 selector=getselector(); //selector position reading
63 //selector in position zero allows for programming (waiting loop)
64 if(selector==0){
65 while(1);
66 }
67
68 //selector in position different from zero will start the code execution,
69 //the UART is cleaned and LED7 confirm the correctness of the set-up
70 LED7 = 1;
71 int flag = 0;
72
73 //starting of the infinite loop cycle. The micro-controller wait that a command is
74 //received from the serial port and if it is equal to "v", it will interpret the
75 //following four data as: linear velocities along the two axis (vxp,vyp), linear speed (v)
76 //and angle (theta) of the robot
77 while(1){
78 time(&timer1);
79 e_getchar_uart1(&car);
```

```
80
81  //we must send from MATLAB the char 'v' followed by four integers
82  if (car=='v'){
83  while(e_receive_int_from_matlab(dato,4)==0);
84  LED4=0;
85  //data must be divided by 1000, to be consistent with the MATLAB script
86  vxp = dato[0]/1000.0f;
87  vyp = dato[1]/1000.0f;
88  //v = dato[2]/1000.0f;//not necessary for the adopted feedback linearization
89  theta = dato[3]/1000.0f;
90  car = '\0';
91  flag = 1;
92  }//Closing of the if related to command 'v'
93
94  //normal loop computing the feedback linearization, actuating the motors and
95  //sending the accelerations
96  while(!e_ischar_uart1()) {
97  timer1=time(NULL);
98
99  //if the linear speed is close to zero, set it to 0.001
100 if (v >= 0.0f && v < 0.001f)
101 v = 0.001f;
102 if (v <= 0.0f && v > -0.001f)
103 v = -0.001f;
104
105 //implementation of the non-singular feedback linearization
106 v = vxp*cos(theta)+vyp*sin(theta);
107 w = (1/ecall)*(vyp*cos(theta)-vxp*sin(theta));
108 theta = theta + tau*w; //update of the angle theta
109
110 //speed of the wheels
111 vR = v + E * w / 2.0f;
112 vL = v - E * w / 2.0f;
113 //angular speeds
114 wL = vL / R;
115 wR = vR / R;
116 //velocities in step/s
117 speedl = floor(kk * wL);
118 speedr = floor(kk * wR);
119 //saturation of the velocities
120 if (speedl>1000) speedl = 1000;
121 if (speedl<-1000) speedl =-1000;
122 if (speedr>1000) speedr = 1000;
123 if (speedr<-1000) speedr =-1000;
124
125 //acceleration reading
```

```
126 ACC = e_read_acc_xyz();
127 x = ACC.acc_x;
128 y = ACC.acc_y;
129 z = ACC.acc_z;
130 //accelerations data are sent through the serial port
131 char vectorACC[20]={0};
132 sprintf(vectorACC,"a:%d:%d:%d:\n",x,y,z);
133 e_send_uart1_char(vectorACC,strlen(vectorACC));
134 while(e_uart1_sending()) ;
135
136 //actuation of the motors
137 e_set_speed_left(speedl);
138 e_set_speed_right(speedr);
139
140 for(i=0;i<30000;i++){
141 asm("nop");
142 }
143 //management of the timing in seconds
144 while(((difftime (time(NULL),timer1))/(float)CLOCKS_PER_SEC) < tau) {
145 if (flag == 1) {
146 LED4=1;
147 flag=0;
148 }}}
149 }//end of the infinite loop
150 }//end of the main
```

List of Figures

List of Tables