**Alma Mater Studiorum - Università di Bologna**

**Dottorato di Ricerca in**
**Automatica e Ricerca Operativa**

**ING-INF/04 Automatica**

XXI Ciclo

# Coordinated Control of Robotic Swarms in Unknown Environments

Riccardo Falconi

**Il Coordinatore**
Prof. Claudio Melchiorri

**Il Relatore**
Prof. Claudio Melchiorri

A.A. 2005-2008

To all my fellow travelers

There was a great battle of ideas
at the end there were no losers,
no winners
and no ideas.

Stefano Benni - Elianto, 1996

Fratello, non temere, che corro al mio dovere!
Trionfi la giustizia proletaria!

Francesco Guccini - La locomotiva, 1972

# Contents

# Chapter 1

# Introduction and Thesis Outline

This chapter briefly introduces the basic concepts that are behind the idea of using swarm of autonomous robots to accomplish predefined tasks and presents the state-of-the art in this research field. Then, the arguments exposed in this thesis will be briefly presented, along with the outline of this work.

## 1.1 Multiple mobile robot systems

Starting early in 1980's, the attention of researchers was attracted by the idea of creating groups of mobile robots able to collaborate in order to accomplish one or more predefined tasks. The basic principle behind this new approach to the robot coordination was directly inspired by the observation of natural systems. In nature, in fact, it is possible to see a lot of animals that work together for a final common purpose: typical example can found in the sea, on the ground and in the air, and more evolved animals can collaborate to perform more complex *social* behaviors (see Fig. 1.1). Ants and termites can work together to build enormous nests or to transport a pray that is many times heavier than the individuals; birds can fly in compact formations during migrations in order to save energy, and fishes school in groups with thousand individuals to protect from enemies. Predators find it easier to chase down a fish swimming all alone, than trying to cut out a single fish from a huge group. Other animals, like lions, are able to organize to perform more complex tasks like hunting. As example, naturalists observed that in a group of lions some of them have to scare the pray, while some others have to intercept and kill it. Alternatively, they can organize themselves to attack and kill a pray which is too big for a single predator. An example is reported in Fig. 1.1(d), where four lions attaching

(a) Ants collaborate to form a living bridge.

(b) Flock of seagulls at Dunedin Harbor, New Zealand.

(c) Butterfly fishes schooling.

(d) Four lions collaborate during hunting.

Figure 1.1: Examples of social animals

a gnu can be seen. As the animal behavior has evolved in collaborative direction in order to increase the survival probability of the species, researchers start thinking that maybe a multi-robot mobile system, where all the agents collaborate, can increase the chances to accomplish a predefined task. Among all animals, men represent the higher collaborative level, as in the last 10 thousands years was able to drastically modify the environment. Starting from these considerations derived from natural systems, researchers have identified two main collaborative paradigms: the *intentionally cooperative* systems and *collective cooperative* systems. The idea behind the first paradigm is that all the robots in group have knowledge about the presence of other team members, and are able to coordinate each other exploiting global information like the state and the capabilities of teammates. Multi-mobile robot systems based on this approach usually present different kind of vehicles, with different abilities. Examples are presented in [90], [88]. In the first work, the authors demonstrate that an Unmanned Aerial Vehicles (UAV) and ground vehicles can collaborate to perform reconnaissance and surveillance (Fig. 1.2(a)); similar

(a) Three robots cooperating for surveillance. (b) Three robots cooperating in an assembly task.

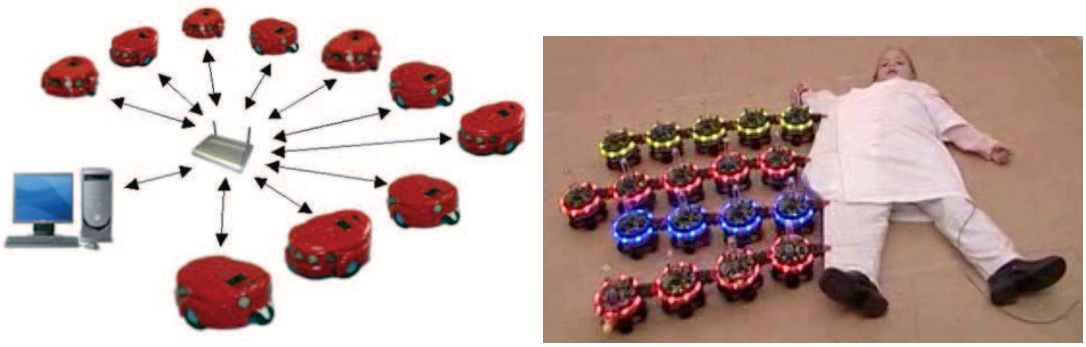
Figure 1.2: Two examples of an heterogenous multirobot system.

results can be found in [54] [1] [84]. In the second paper, authors show the possibility of using three different robots (a mobile manipulator, a crane and a mobile robot equipped with a camera) to perform an assembly task as the three robots have to collaborate to connect a beam at a given location (Fig. 1.2(b)). Other references on the coordination of mobile manipulators can be found in [71] and [73], where the problem of creating a fault tolerant architecture for heterogenous group of mobile robots is addressed.

The key assumption of the second paradigm is that a group of agents is composed of a large number of mobile robots with same characteristics (e.g. same sensors, size and interaction ability) - it is called *homogenous swarm* - where each robot performs its own tasks with minimal information about the state of teammates. In these systems, robots use a local control law in order to create a coherent team behavior, and the situations where such a kind of system can be use have to be described with scalable task functions, in order to add or remove easily agents from the group. Roughly speaking, no team member is important and every one of them can be sacrificed (the swarm is implicitly fault tolerant), while in *intentionally cooperative* systems it is not.

The robustness of multirobot systems is strictly related to the control structure used to organize the agents and to obtain the desired emergent behavior. Four different architectures can be identified in the field of mobile multirobot systems, that are summarized below.

**Centralized:** a global supervisor receives information from team members and the swarm behavior is coordinated from a single control point. This architecture (examples can be found in [66] [50]) has the advantage that as all the information are collected by a single unit, this *node* of the system can be powerful enough to calculate the control law for each robot, considering also the opportunity of having complex tasks. On the other hand, the system is vulnerable, as a fault of the supervisor implies that all the system crashes. These simple considerations were those that led to the development



(a) Centralized architecture of the Decabot Project.

(b) Example of a decentralized architecture

Figure 1.3: Two examples of *centralized* and *decentralized* control of a group of mobile robots

of internet in the late 1960s. In Fig. 1.3(a) an example of centralized architecture is depicted.

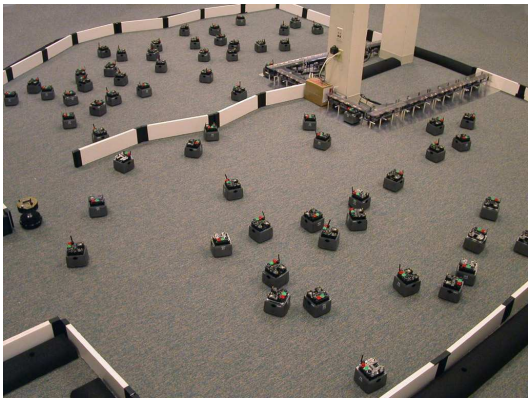
**Decentralized:** it is the most used architecture to control multirobot systems, and can be considered as the opposite of the centralized approach. In a decentralized architecture, each robot acts based only on knowledge of local teammates' state and of environment. This control subsumption is robust to failure but, on the other hand, presents limits to power computation, means that can be non trivial to implement complex tasks in distributed fashion. An example of decentralized architecture is shown in Fig. 1.3(b), where 18 *Swarm-Bots* [38] organize themselves in order to drag a body in an hypothetic disaster scenario.

**Hierarchical:** this architecture, directly inspired by military command protocol, is suitable for some applications. It is based on the idea that some robots can command - as supervisors - a local and relatively small group of team members. Once again, as in the centralized approach, weakness of this architecture can be found in recovering from failures of robots high in the command tree.

**Hybrid:** this approach tries to be a trade off between the centralized and decentralized architecture. In particular, it is base on the idea that one or more high level supervisors allocate tasks and resources, and single low level robots exploit local information to accomplish a predefined task. The hybrid architecture has been used in many applications based on multirobot control [93] [42].

The first experiments on *swarm robotic systems* - the Nerd Herd experiment [60] - deals with a decentralized control architecture, where 20 robots use very little explicit communication rather then *stigmergy* (i.e. communication through environment modification),

(a) Robots involved in the *nerd herd* experiments.

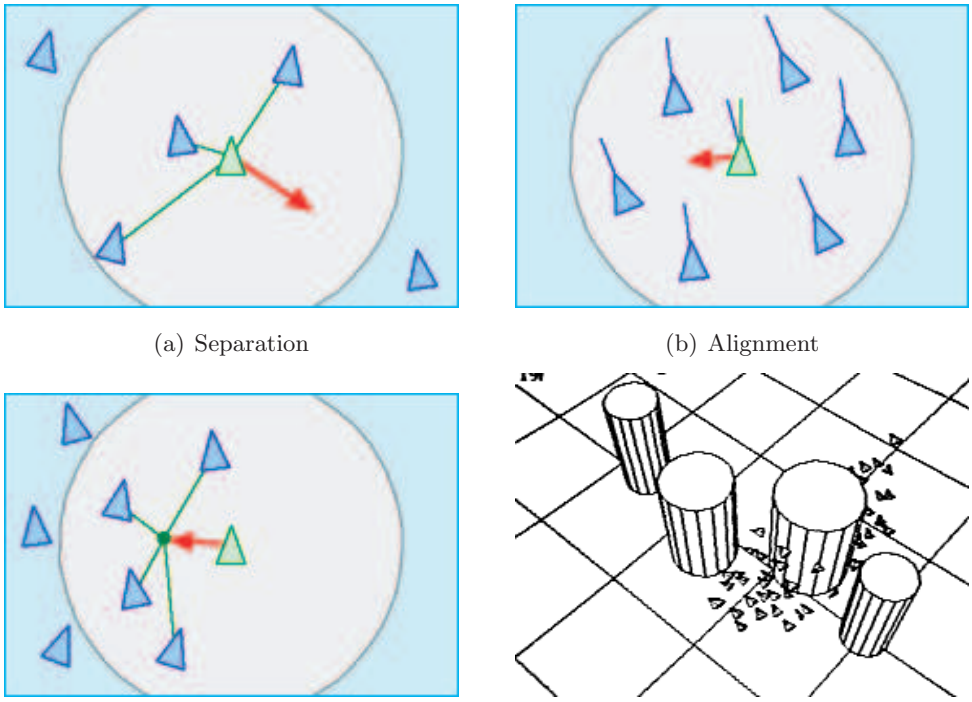


(b) A picture of the Swarmbot experiments

Figure 1.4: Examples of robot swarms

in order to perform foraging, dispersion, surrounding, and herding (see Fig. 1.4(a)). Each single robot is supposed to have very minimal capabilities in terms of sensing and acting on the environment, but they are able to collaborate in order to show complex behaviors. Systems like that, where simple agents can perform complex goals, are called *superadditive*, meaning that the whole can do more than the single. Recent experiments performed with 100 robots (SwarmBots, see Fig. 1.4(b)), has pointed out the possibility of merging different simple tasks in order to achieve complex goals [61].

Since robotics start working on group of mobile robots, studying both homogeneous and heterogeneous groups, their attention focuses on the possibility of obtain a large spectrum of emergent behaviors, such as multiagent manipulation, traffic control and path planning, foraging and coverage, flocking and formation keeping. In particular, the last two has attracted researchers attention for many reasons, among them the fact that they can be used to model the behavior of social insects like ants (*foraging*) or social animals like birds (*flocking*). In foraging domain, where a very large number of mobile robots are involved, objects like pucks are distributed in the environment, and the global goal is to find and collect them; a typical real applications of the foraging problem are demining, search and rescue, toxical waste removal. The main issue in this application is how to coordinate robots in order to explore as fast as possible all the terrain without interfering with each other. Thus, *foraging* is strictly related to the *coverage problem*, whose solution can be applied to real world problems such as mapping, surveillance or industrial surface cleaning. In this cases we can talk of *weakly cooperating systems*, as the solution of the foraging and coverage problem usually requires minimum communication between teammates. Examples of solution for these problems can be found in [20] [94] [83]. Flocking and formation keeping are global goals that can be considered like different realizations of the same basic problem, that is to find a way to coordinate a group of



(a) Separation

(b) Alignment

(c) Cohesion

(d) Boids flocking around obstacles

Figure 1.5: Reynolds' rules for boids flocking.

mobile robots in order to make them move together while preserving group compactness. Eventually, flocking and formation tasks can be involved in a point-to-point group transfer, means the group has to move from a starting zone to a final one. In particular, it is possible to consider the formation keeping task as a subcase of the flocking task: while flocking needs a weak relation between teammates positions, the formation task requires a strong relation in order to achieve and preserve predefined distances between the agents of the group. Typically, flocking is a natural behavior, while formation keeping characterizes human activities.

One of the first solutions to the coordination of mobile agents flocking together was presented by *Reynolds* in [78]. In this work, the author used three basic rules in order to reply the behavior of a group of birds (called *boids*). In particular he defined:

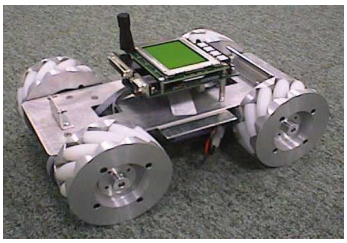
**Separation:** steer to avoid crowding local flockmates (Fig. 1.5(a));

**Alignment:** steer towards the average heading of local flockmates (Fig. 1.5(b));

**Cohesion:** steer to move toward the average position of local flockmates (Fig. 1.5(c)).

The result was a realistic flock-behavior simulation (a snapshot of a simulation can be seen in Fig. 1.5(d)), but, more important, he demonstrates that with a simple set of rules many complex behavioral schemes can be created. A similar approach was later used

(a) A six-legged spider robot.

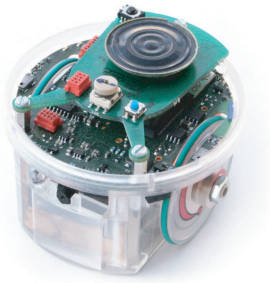


(b) Omnidirectional robot with swedish-wheels

(c) A flying quad-rotor UAV

(d) Couple of Serafina Mark II AUV used in the *Swarmsea* project

(e) Khepera III robot by K-Team.



(f) E-Puck robot by Cyberbotics.

Figure 1.6: Examples of mobile robots used in swarm applications.

in [59] [7] to study emergent behaviors in groups of real robots. The so called behavioral approach was used also in [34] [35] [37] [36], where authors focused their attention on the possibility of using a behavioral approach to decompose a global goal function in simpler tasks (eventually in conflict) with assigned priority-levels.

The most recent architecture pointed out by researchers in order to control an homogeneous group of mobile robot is based on *graph control theory*. This fully decentralized approach, initially used on groups of massless-point agents, is based on concepts borrowed from the graph theory, and it exploits the *Laplacian solution* to the *consensus problem* (or *rendezvous problem*) in order to achieve goals like herding, leader-based optimal control and formation keeping or flocking. Later, similar concepts were applied on non-holonomic vehicles, in particular on differential wheel robots. Examples of these work can be found in [27] [77] [30].

Generally, to achieve a global goal, each robot controller is implemented in two different layers: a high-level layer, devoted to the calculus of the swarm strategy, and a low-level layer, that take into account the kinematic/dynamic robot model and controls the robot's actuators. The idea of separate the *strategy* controller from the *actuators* controller is suitable to work both with heterogeneous and homogenous groups of robots. In case of robot swarms, another approach consists in considering at the same time the global goal

and the kinematic/dynamic constraints of the robot model, and find a control law that globally stabilize the system.

Even if a rich collection of mobile robots can be found (see Fig. 1.6 for some examples), in the field of swarm robotics few of them can be really used due to the complexity of the models. Typically, researchers focused they attention on omnidirectional robots and on differential wheel robots (or tracked robots), both used for outdoor and indoor applications. In Fig. 1.6(b) 1.6(c) an omnidirectional four-wheel robot and an omnidirectional quad-rotor UAV [41] are depicted respectively. They are actually used for indoor activities, but researchers are planning to use them for outdoor surveillance. In Fig. 1.6(d) a couple of Serafina Mark II [76] Autonomous Underwater Vehicles (AUVs) are depicted: by a control point of view, they can be modeled as differential wheel robots except the fact they can change their depth. In Fig. 1.6(e) 1.6(f) an E-puck by Cyberbotics and a Khepera III by K-Team [67] are depicted. These differential wheel robot models are worldwide used for indoor research and educational activities.

In our research, we decide to focus our attention on the implementation of decentralized algorithms able to reply some basic simple animals behaviors, such as hunting or formation keeping. In particular, we concentrate on the idea of having robots (or, more generally, mobile agents) dealing with constraints that limit their ability to understand the environment, like the problem of moving in unknown environments or the use of local sensing data instead of using global data.

## 1.2 Outline

This thesis, that gathers the work carried out by the author in the last three years of research, is composed by two main topics corresponding to two different tested algorithms. Both of them deals with the problem of driving a swarm of robots through an unknown environment while achieving one or more predefined tasks. The first algorithm, based on *null space behavioral* concepts, was studied and developed at LAR (Automation and Robotic Lab) of the University of Bologna and tested by means of simulations. The second algorithm, instead, is based on concepts borrowed from the *graph theory* and was developed and tested also on real Khepera III robots at DISAL (Distributed Intelligent Systems and Algorithms Laboratory) at École Polytechnique Fédérale de Lausanne, Switzerland.

In Chapter 2 our attention is focused on the development of a decentralized algorithm to coordinate a group of autonomous mobile agents moving through an unknown environment while performing more tasks defined with different priorities. The main idea is to decompose a global *swarm task* in different simple subtasks in order to flock the swarm toward a target with known position and surround it with regular polygon formation (i.e. minimize the target escape possibilities). After a brief introduction to the main paradigms

used in literature to merge different tasks, our algorithm - based on the so called *Null space behavioral approach* (NSB) is presented. In particular, we focus on the possibility of having a decentralized version of the classical NSB that exploits only local data. Moreover, we have introduced extra degrees of freedom to ensure the swarm avoid unknown obstacles exploiting data collected by swarm individuals. To conclude, a new strategy to avoid local minima is investigated, and numerical simulation are used to validate our approach.

In Chapter 3 concepts and basic notions from *graph theory* are resumed, in order to provide the reader the basic knowledge needed to face the so called *consensus problem* and to apply it on groups of autonomous mobile agents. In particular, a classical well-known solution of the *rendezvous problem* for holonomic agents is introduced, exploiting the idea of using the so called *Laplacian approach* to drive all the states of a system to a predefined final value. After that, the solution of the consensus problem is applied to solve some standard problems in graph control theory, such as formation keeping, connectedness preserving. Moreover, under the assumption that some agents are controllable from an exogenous system, optimal control problems and herding problems are faced. All these classical problems are illustrated by means of simulation examples.

In Chapter 4, the concepts of graph theory are applied on systems composed by differential-wheeled robots. In particular, the chapter is divided in two main parts: in the first one, the consensus problem is solved, i.e. a control law is found to drive all the robots to a final common point; in the second part, using a modified version of that control law, it is shown that the swarm is able to reach a predefined configuration and to maintain it while moving in the environment. As an example application of the defined algorithm, a swarm of robots is controlled to surround a fixed target, engaging in a typical hunting task. As for the previous chapters, here all the results are supported by simulation examples implemented both in Matlab/Simulink and Webots. The chapter is concluded with comments about the implementation of the algorithm on real Khepera III robots with range and bearing on board platform, an hardware board that allows robot to find teammates' position without an external detection system like Global Positioning System (GPS) or fixed cameras. The experiments were performed at DISAL lab at EPFL.

Chapter **??** collects final comments about the obtained results and the possible guidelines for future work.

Even if it is not strictly correlated to the research guideline of this thesis, in Appendix **??** a brief description of RobotiCad is provided. RobotiCad is a Matlab/Simulink toolbox developed by the author in the last three years in order to simulate open chain manipulators in pseudo industrial environments. This toolbox is a 55-thousands-lines software composed by a collection of many Matlab functions and a Simulink block library. It was originally developed to provide a useful tool to the students of the course on "Industrial Robotics" at the Faculty of Engineering of the University of Bologna, so that they

can easily understand basic concepts behind robotic theory. After a brief description of the toolbox, this chapter deals with two case studies: the first one is a typical industrial problem that was solved by students during the academic year 2007/2008; the second one summarizes the results obtained in [10], where the toolbox was integrated with genetic algorithms in order to design the *minimum* kinematic configuration parameters of a medical robots to heal patients affected by Benign Paroxymal Positional Vertigo and variants (collectively called vestibular lithiasis), a common disorder caused by a malfunction of the inner ear.

# Chapter 2

# Decentralized Control for Robot Teams in Unknown Environments

In this chapter a decentralized control algorithm for swarm behavior and obstacle avoidance in unknown environments is presented. The proposed technique is based on the Null Space Based behavioral approach to merge different predefined tasks with assigned priority levels. In particular, the algorithm focuses on the possibility of sharing sensors data in order to avoid obstacles that can be detected in the environment and on the implementation of strategies to avoid that the group (or a single robot) gets stuck in a local minima. The effectiveness of our approach is investigated by means of numerical simulations with different environments.

One of most interesting problem in mobile robotics is to control one or more autonomous agents in order to achieve different tasks at the same time. This problem has been solved essentially in two different ways: the first one concerns the use of a cost function where all the tasks are mixed with different weights; the second one is based on the possibility of creating a hierarchy among the different tasks, so that the lower tasks are satisfied only when the higher tasks became accomplished. A typical approach to this problem was pointed out firstly in [28], [13], [16], were the authors used the kinematic redundancy of the system to merge more tasks. This approach, called Null-Space Based Behavioral approach, is used in this chapter in a decentralized way in order to drive a platoon of agents from a starting random configuration to a final regular one, in particular driving the swarm through an unknown environment toward a target area. The key point of this work is the decentralization of the algorithm and the implementation of an obstacle avoidance function that allows the group to escape from local minima while preserving

group properties. In this chapter, after a brief introduction on the behavioral control paradigms, the robot model used in our algorithm is introduced and the decentralized control structure used in this work is explained. To conclude, in Section 2.4 simulation results are collected in order to demonstrate the efficiency of the algorithm.

## 2.1 Background on behavioral control

Generally, the behavior of a robot is expressed through the definition of a cost function and its value is the index of task achievement. The basic example is the point-to-point motion of a vehicle: the cost function is usually a function of the distance from the final position (*target*), and the control can be either a constant value (motors are always on with constant velocity till the final point is reached or, in presence of disturbance, the error is small enough) or a varying value depending on the distance or, more generally, on the environment. In fact, if we suppose that a forbidden area is between the robot and the final position, the function that drives the robot toward the end point must be modified to avoid that area. Problems arise when the driving function tries at the same time to move the robot toward the target and away from the obstacle. In this case, we deal with *conflicting tasks*. This problem has been solved in literature with different approaches that have evolved depending on the on-board computational power of robots and on the different ideas behind the way to solve conflicts. The most common approaches to solve problem are summarized below.

### 2.1.1 Competitive behaviors coordination

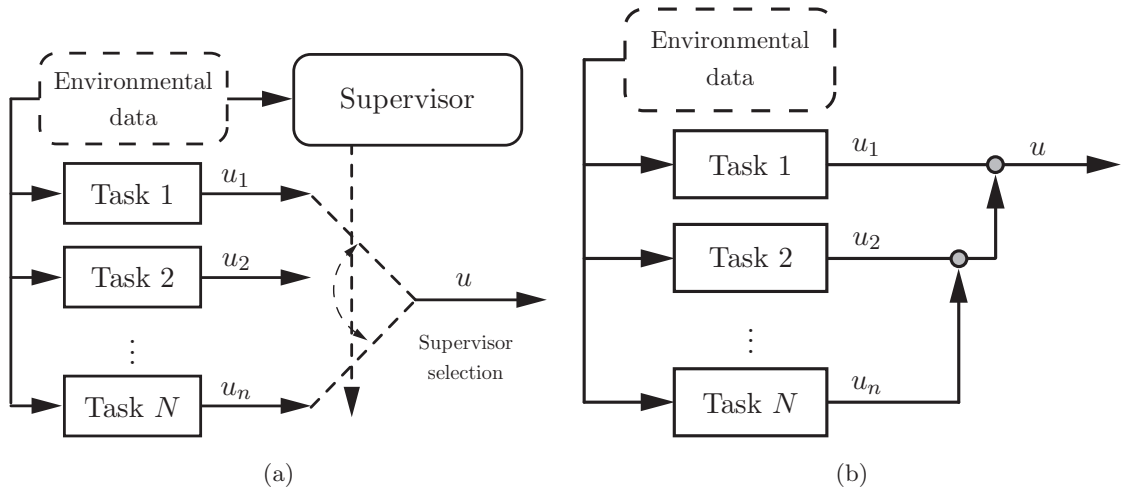



Figure 2.1: Different schemes of the competitive behavior coordination paradigm.

This approach, presented in [75], can be implemented in various ways, but the general

concept is that even if many tasks are defined at the same time, only one of them produces the signal command used to drive the robot. Fundamentally, there are two paradigms to implement this control logic:

- Each task is an asynchronous function that uses the environmental data to calculate the command signal needed by the robot to accomplish it. In this case, a supervisor that analyze the same data is needed in order to switch between the different tasks. Since the supervisor can switch task in each moment, all the tasks have to be computed in parallel, and therefore high computational power is required for the system (see Fig. 2.1(a)).

- Tasks are organized in different layers, each one with an assigned priority level (see Fig. 2.1(b)), and the output they calculate is always deleted by higher priority tasks. This means that, as long as the primary task is not satisfied, the lower priority tasks can be deactivated and they are not considered in the command signal. Typically, the obstacle avoidance is defined as the higher priority task. The computational power needed for this scheme is lower than in the previous approach.

Of course, the functions devoted to the calculus of each task output can be implemented as preferred, and the most used is the virtual potential concept [14] [46] [85] [52] [47].

### 2.1.2 Cooperative behaviors coordination

Another paradigm used to solve conflicting tasks is the *cooperative coordination*. Generally, in this paradigm any task is completely deleted by others, i.e. all the tasks are merged together in order to reach a final control signal that is a trade-off between all the desired results. A typical scheme for cooperative control is the so called *motor schema* [8] depicted in Fig. 2.2, that is a cooperative scheme directly inspired from biological considerations. As can be seen it is requested that a supervisor is present in the control architecture in order to multiply each contributes for a weight defined considering environmental information. All the weights can be instantaneously changed by the supervisor, depending on the importance that each task has in that moment. The motor schema has an output given by the linear combination of the weighted contribution provided by each task, i.e.

$$u = \sum_{i=1}^{N} \alpha_i \cdot u_i$$

Since all the tasks must produce an output, it is clear that the computational power requested for this control scheme is higher than in the *layered approach*. If the weights produced by the supervisor are $\alpha_i \in \{0, 1\}$, the motor schema can be seen as the supervised competitive control of Fig. 2.1(a).

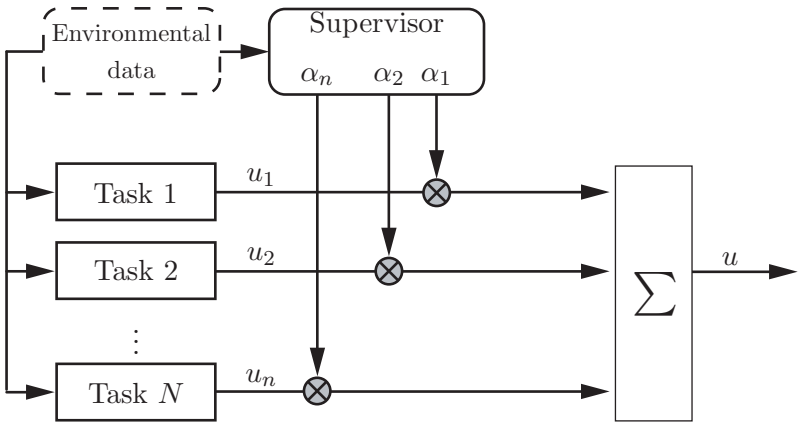



Figure 2.2: Scheme of the cooperative behavior coordination paradigm.

### 2.1.3 Null-space behavior coordination

As discussed, the possibility of accomplishing different tasks at the same time is of interest in robotic missions. A way to solve this problem is to use a task-priority inverse kinematic function [68] that, in case of conflicting tasks defined with an associated priority, allows to avoid singularity configurations. In [37,36] this approach, originally developed for ground-fixed robotic manipulators, has been applied to a platoon of mobile robots. The basic concepts that will be used later in this chapter are summarized below.

Let $\bar{x} = f(q)$ be the task variable to be controlled, e.g. mean value or variance or regular polygon configuration. By deriving, it results:

$$\dot{\bar{x}} = \frac{\partial f(q)}{\partial t} = \frac{\partial f(q)}{\partial q}\frac{\partial q}{\partial t} = J(q)\dot{q}, \tag{2.1}$$

where $q$ is the state-space vector of the system and $J(q) \in \mathbf{R}^{m\times n}$ is the Jacobian of the system. As we consider a 2-dimensional instance of our algorithm, we have $m = 2$. If $\dot{\bar{x}}$ is known for a predefined task, i.e. $\dot{\bar{x}} = \dot{\bar{x}}_{task}$, the value of $\dot{q}_{task}$ can be computed by inverting Eq. (2.1):

$$\dot{q}_{task} = J^{\dagger}(q_{task})\dot{\bar{x}}_{task} \tag{2.2}$$

where $J^{\dagger}(q)$ is the (pseudo-)inverse of the Jacobian matrix. If we consider groups of robots with at least two agents, it results $m < 2n$. This means that the Jacobian matrices of the task have always a low-rectangular shape and the inversion problem admits infinite solutions (see [87] for a tutorial). The relationship expressed in Eq 2.2 has been widely studied in robotics. A typical solution is given by:

$$J^{\dagger} = J^{T}\left(JJ^{T}\right)^{-1}$$

that pursues minimum-norm velocities, corresponding to minimum square solution. With

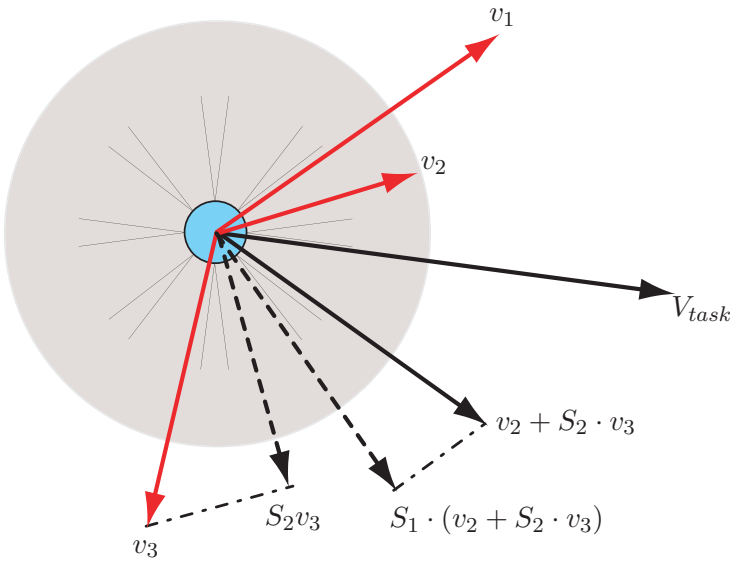



Figure 2.3: Multiple tasks merged together, where $S_h = \left(I - J_h^{\dagger} J_h\right)$.

this solution, it follows:

$$\dot{q}_{task} = J^{\dagger}(q_{task})\dot{\tilde{x}}_{task} = J^T \left(JJ^T\right)^{-1} \dot{\tilde{x}}_{task} \tag{2.3}$$

The *null* space of the Jacobian matrix is spanned by $\left(I - J^{\dagger}J\right)$. This technique is useful because, by projecting the lower priority task onto the null space of the higher one, it is possible to employ those extra degrees of freedom to merge together more tasks while preserving different priorities. As an example, let's suppose to have three tasks $(v_1, J_1)$ $(v_2, J_2)$ and $(v_3, J_3)$, where $\dot{\tilde{x}}_i = v_i$, and the lower index corresponds to the higher priority.

The final control action for the desired merged tasks is given by:

$$v_{task} = v_1 + \left(I - J_1^{\dagger} J_1\right) \left(v_2 + \left(I - J_2^{\dagger} J_2\right) \cdot v_3\right) \tag{2.4}$$

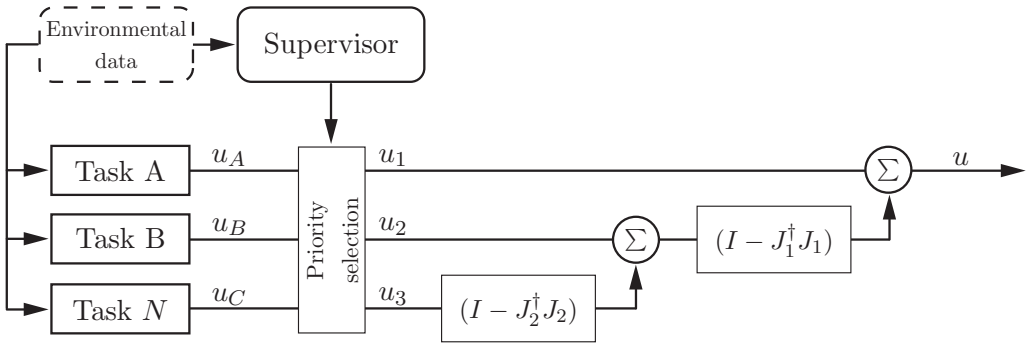



Figure 2.4: Scheme of the null-space behavioral coordination paradigm.

With reference to Fig. 2.3, it is easy to see that, considering for simplicity only two tasks $v_1$ and $v_2$, all the components of the second task that conflict with the first one are filtered out and do not affect the final result. A sketch of the behavioral control is showed in Fig. 2.4. Now, the motion controller needs the position trajectory besides the velocity trajectory. In a continuous-time system, this can be obtained by time integration but, since all the robots are in the end controlled using discrete time systems, this will introduce a numerical drift in the integration process. To avoid this problem, the position of the agents can be reconstructed by using the so called *closed loop inverse kinematic (CLIK)*, that is:

$$\dot{q}_{task} = J^{\dagger}(q_{task})(\dot{\bar{x}}_{task} + \Omega\tilde{x}_{task})$$

where $\Omega$ is a suitable diagonal positive-definite constant matrix and $\tilde{x}_{task} = \bar{x}_{task} - \bar{x}$ is the task error.

## 2.2 Problem Statement and Robot Model

Starting from the concepts introduced in Sec 2.1, our goal was to create a high-level decentralized controller able to manage different priority tasks in order to move a group of robots in an unknown environment from initially random positions to a final target.

To define the problem, let's suppose to have a group of autonomous agents moving in an unknown environment that have to flocks in formation toward a target placed in a known position. Let $G$ be a platoon of $n$ robots, and $A$ a *subgroup* of $G$, i.e. every group of robots whose cardinality is less or equal to $n$:

$$A \text{ is a subgroup} \quad \Leftrightarrow \quad \|A\| \leq n$$

Let $\Phi$ be the subgroup with zero elements. The position of the $i$-th robot on the plane is denoted by the vector $q_i = [x_i,\ y_i]^T$, $q_i \in \mathbf{R}^2$, and the vector describing the position of all robots in a subgroup $A$ is given by:

$$q_A = \left[q_1^T, q_2^T, \ldots, q_{\|A\|}^T\right]^T, \qquad q_A \in \mathbf{R}^{2\|A\|}$$

We consider robots bearing only proximity sensors, limited communication range, limited target-visibility range and able to detect their position. Moreover, we suppose that each robot is able to exchange information with its neighbors, but no data broadcasting is introduced. A schematics of one of the robots is shown in Fig. 2.5, where:

- $R^i_{COMM}$ is the radius of the communication area: the $i$-th robot can communicate with the $j$-th robot only if their communication areas intersect. Namely,

$$C^i = \{ \bigcup_{R_j \in G} :| \ q_i - q_j \ | \leq R^i_{COMM}\},$$

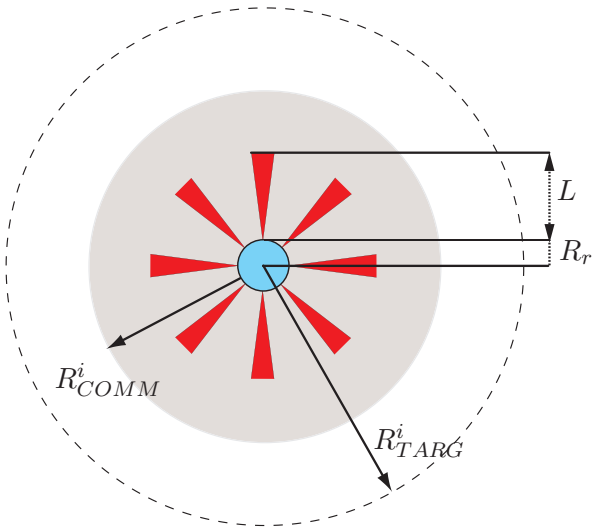



Figure 2.5: Description of the robot used in our algorithm.

is the subgroup of robots that can communicate with the $i$-th robot. We consider $R_i \in C^i$;

- $R^i_{TARG}$ is the radius of the target visibility area: a target can be acquired by the $i$-th robot only if the robot-target distance is lower than $R^i_{TARG}$.

$$T = \{ \bigcup_{R_j \in G} :| \, q_j - q_{target} \, | \leq R^j_{TARG} \}$$

is the subgroup that collects the robots able to see the target;

- $R_R > 0$ is the robot's body radius.
- $L$ is the range of the proximity sensors on board. As shown in Fig. 2.5, they are modeled as cones (triangle in $2D$) to take into account measurement errors in case of irregular obstacles.

During the implementation of our algorithm, we have supposed that the communication radius $R^i_{COMM}$ is always relatively short. This limitation was introduced to imitate natural constraints recognized in massive swarms of animals like fishes or birds, where each individual can share information only with local teammates.

As an example, in Fig. 2.6 five robots ($R_1 \ldots R_5$) are shown in a particular configuration where the following subgroups can be recognized:

- $C^1 = \{5\}$; $C^2 = C^4 = \{3\}$; $C^3 = \{2,\ 4\}$; $C^5 = \{1\}$;
- $T = \{2,\ 3,\ 4\}$;

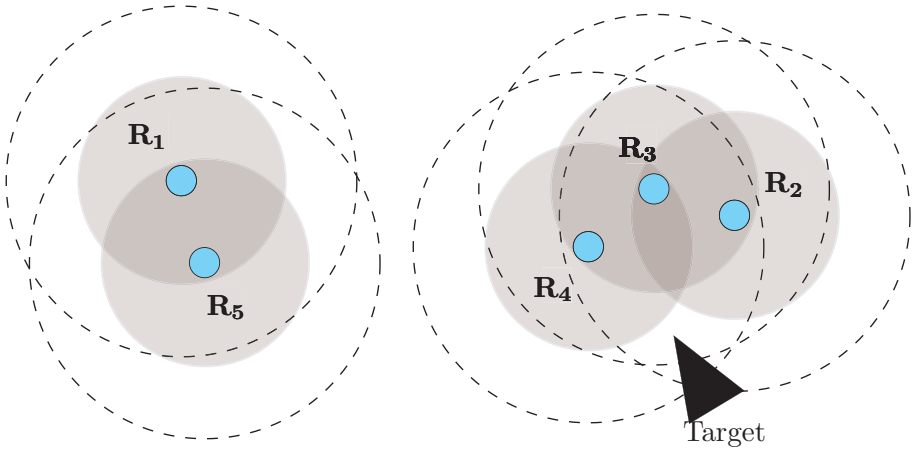



Figure 2.6: Communication topology and data sharing

where the gray areas are the communication areas and the dotted circles represent the target-visibility areas.

As we are interested in defining a general algorithm and not a specific one for a given robot model, we assume that the vehicles are omnidirectional robots. A low-level control can be introduced to apply this general algorithm to different robot models, tanking into account a different sensor configuration or other constraints on the ability of each robot of sensing the environment. Moreover, this algorithm can be used to control heterogeneous group of robots. In our case, it is:

$$\begin{aligned} \dot{q}_i(t) &= \begin{bmatrix} \dot{x}_i(t) \\ \dot{y}_i(t) \end{bmatrix} = \begin{bmatrix} \dot{u}_{i,x}(t) \\ \dot{u}_{i,y}(t) \end{bmatrix} = u_i(t) \\ q_i(0) &= q_i^0 \end{aligned} \tag{2.5}$$

$t \geq 0$, $\forall R_i \in G$, where $u_i(t)$ is the control input of the $i$-th robot and $q_i(0)$ its initial position.

## 2.3 Control Structure

The overall structure of the controller is reported in Fig. 2.7. Since, from Eq. (2.5), the control action affects directly the velocity, we define the controller output for the $i$-th robot $R_i$ as the contribution of two different velocity terms

$$u_i(t) = V_{obst,i}(t) + V_{task,i}(t) \tag{2.6}$$

where $V_{obst,i}(t)$ and $V_{task,i}(t)$ are devoted to avoid obstacles that can be present in the unknown environment and to accomplish the predefined mission, respectively. Both these terms are thoroughly explained below.

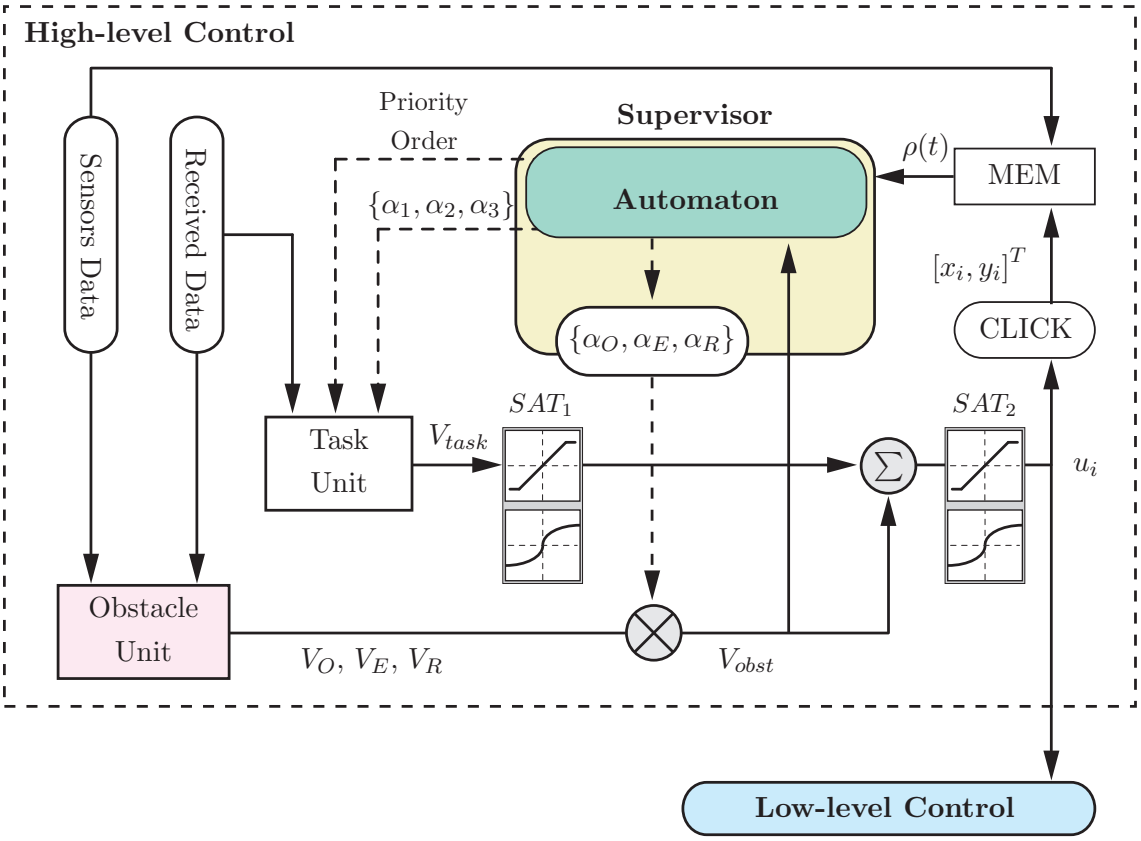



Figure 2.7: Control architecture

### 2.3.1 Task unit

The block named "Task Unit" is devoted to calculate the value of the control that the agent should perform in order to achieve the predefined tasks. In particular, we choose to decompose the global task "follow and surround a target" in three different subtasks, that are, for the $i$-th robot:

**Definition 2.1** (Tasks definition)**.**

1. *move in such a way as to contribute moving the center of the group $C^i$ toward the target position;*
2. *keep a predefined distance from the target position;*
3. *maintain the same distance with respect to the closest neighbor (if $|C^i| \geq 2$).*

*where the given order corresponds to the task priority levels.*

As our aim is to coordinate the swarm in an unknown environment, the best way to merge these three tasks is to use the Null Space behavioral coordination control paradigm,

explained in Sec 2.1.3, because allows to avoid conflicting situations among the tasks to be performed. The contribution for obstacle avoidance in Eq 2.6 has been decoupled from the task contribution because, as robots do not known in advance the position and the shape of the obstacles, any predefined task can be computed in advance. This feature introduces a difference with respect to the work presented in [34] [35] [36] [37]. With respect to Eq 2.7 we have introduced a modification that, as will be explained later, is very important for our purposes. In particular, Eq 2.7 is changed as follows:

$$v_{task} = \bar{v}_1 + \left(I - J_1^{\dagger} J_1\right)\left(\bar{v}_2 + \left(I - J_2^{\dagger} J_2\right) \cdot \bar{v}_3\right) \tag{2.7}$$

where the generic term $\bar{v}_i = \alpha_i \cdot v_i$, means that the supervisor has to monitor both the possibility of exchange the priority order and the weight to assign to each task. In simple words, we can see our approach as a mixture where the null-space paradigm is helped by basic concepts borrowed from the cooperative paradigm. The tasks defined in Def 2.1 can be formalized as follow.

**Task$_1$: Mean value** function (MVf).

The mean value and the corresponding Jacobian matrix are defined as follows:

$$q_{mean,C^i} = \frac{1}{\|C^i\|} \sum_{j \in C^i} q_j = \begin{bmatrix} x_{mean,C^i} \\ y_{mean,C^i} \end{bmatrix}, \tag{2.8}$$

$$J_{mean,C^i} = \frac{1}{\|C^i\|} \begin{bmatrix} \cdots & 1 & 0 & \cdots \\ & 0 & 1 & \end{bmatrix} \tag{2.9}$$

This function moves the mean value of a subgroup $C^i$ towards the target position. The reference value is

$$\bar{x}_{ref,MVf} = \bar{x}_{target} = \begin{bmatrix} x_{target} \\ y_{target} \end{bmatrix}. \tag{2.10}$$

**Task$_2$: Nearest-Neighbors equidistant** function (NNf)

This function moves the $i$-th robot of a subgroup $C^i$ in order to keep the same distance between $(R_i, R_j)$ and $(R_i, R_k)$ i.e. $\mid q_i - q_j \mid - \mid q_i - q_k \mid = 0$. The nearest neighbors w.r.t. $R_i$ are chosen run time by $R_i$ and can change. The NNf and the corresponding Jacobian matrix are defined as follows:

$$q_{NNf,C^i} = \mid q_i - q_j \mid - \mid q_i - q_h \mid, \tag{2.11}$$

$$J_{NNf,C^i} = \left[ 0, \quad \cdots \quad 0, \quad \left(\frac{x_i - x_j}{\Delta_{ij}} - \frac{x_i - x_h}{\Delta_{ik}}\right), \left(\frac{y_i - y_j}{\Delta_{ij}} - \frac{y_i - y_h}{\Delta_{ik}}\right), \quad 0, \quad \cdots \quad 0 \right], \tag{2.12}$$

where $\Delta_{i,j} = \mid q_i - q_j \mid$ and $\Delta_{i,k} = \mid q_i - q_k \mid$. The reference value is $\sigma_{ref,NNf} = 0$.

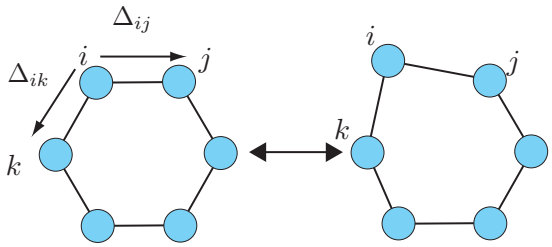



Figure 2.8: Example of configurations with Nearest-Neighbors function

**Task$_3$: On-a-Circle** function (OCf)

$$q_{OCf,C^i} = \mid q_i - q_{mean,C^i} \mid = R_f, \tag{2.13}$$

$$J_{OCf,C^i} = \left[ \begin{array}{ccccccc} 0, & \cdots & 0, & \dfrac{x_i - x_{mean,C^i}}{\mid q_i - q_{mean,C^i} \mid}, & \dfrac{y_i - y_{mean,C^i}}{\mid q_i - q_{mean,C^i} \mid}, & 0, & \cdots \quad 0 \end{array} \right] \tag{2.14}$$

This function keeps $R_i$ on a circle centered in the mean value of $C^i$ and with fixed radius $R_f$ ($\bar{x}_{OCf} = R_f$).

As can be seen from Fig. 2.8 and Fig. 2.9, these two functions used alone cannot force the robots of a subgroup in a regular formation. Merged together, NNf and OCf functions create a complex behavior that drives all the robots of the same communication subgroup to a regular polygon configuration.

The mixture of cooperative and Null Space Based control can be easily justified by the fact that each $\alpha_i$ introduces an extra degree of freedom in the system, allowing the group to change its behavior in more complex ways. In particular, the $\alpha_i$ terms can be used to reduce or increase the module of each behavior function that acts on the group. As an example, let's consider four robots that are flocking in square formation and that have to cross a gate in order to reach a target. As we have decoupled the obstacle avoidance unit from the task unit, the robots depicted in Fig. 2.10(a) should not break the formation although the task priority order is changed. In fact, OCf task and NNf task, combined, force the robots in a regular configuration regardless the their priorities. Otherwise, if we suppose to set $\alpha_3 = 0$ and suppose that $\alpha_2$ is continuous decremented

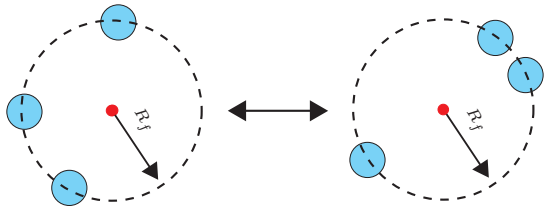
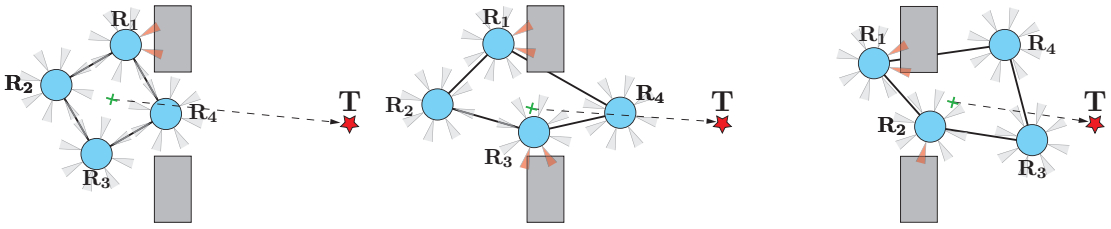



Figure 2.9: Example of configurations with On-a-Circle function

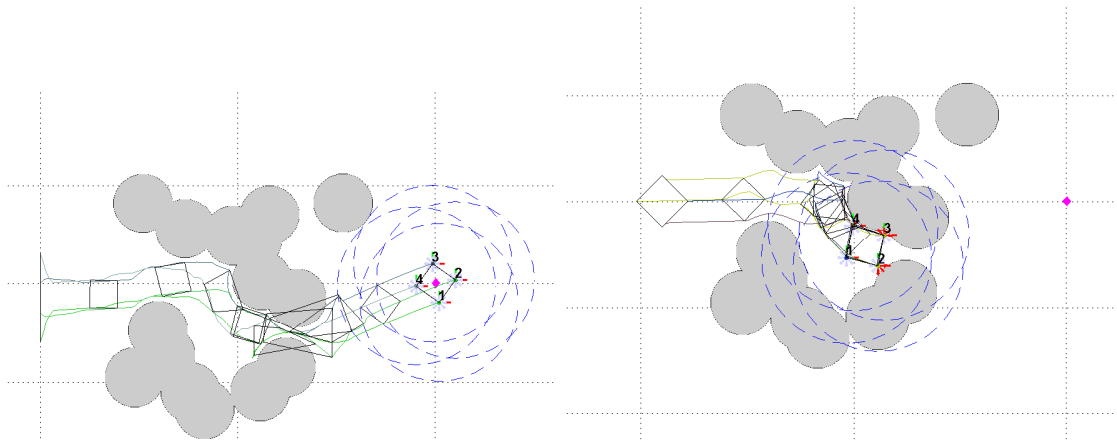



(a) (b) (c)

Figure 2.10: Critical situation in gate-crossing task.

while at least one obstacle is sensed by the group, the configuration can be stretched and the formation can cross the gate (Fig. 2.10(b)-2.10(c)). When any obstacle is sensed, all the weights are incremented and the group can achieve again the square formation. As a simulation example, consider Fig. 2.11 where the same set up is used to simulate two different behaviors of a four-robot-swarm trying to reach a target in presence of obstacles. In particular, in Fig. 2.11(a) the OCf contribution is deleted and the contribution of NNf is reduced by setting $\alpha_{OCf} = 0$,$\alpha_{NNf} = 0.1$ respectively when the robots meet obstacles; in Fig. 2.11(b) all the contribution are at the maximum value, thus robots are not able to break the formation and cross the gate between the obstacles.

The overall scheme of the *Task unit* block is depicted in Fig. 2.12, where for simplicity only two generic behavior functions $(v_1, J_1)$ and $(v_2, J_2)$ are used. Note that, by the definitions of the three tasks, each robot computes its $V_{task}$ exploiting only local data, means that the controller is fully decentralized. So far we have considered all the agents as



(a) (b)

Figure 2.11: Simulation of a group of four robot moving in unknown environment toward a target using varying (Fig. 2.11(a)) and fixed (Fig. 2.11(b)) $\alpha$ weights.

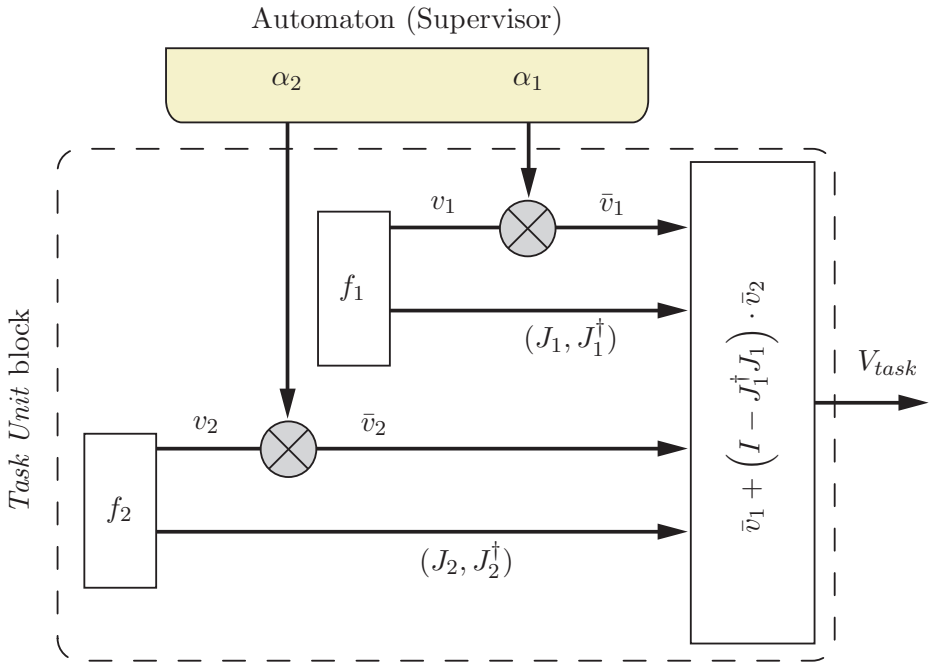
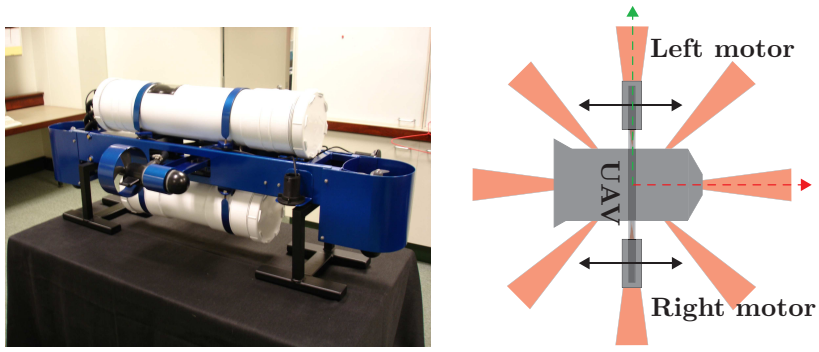



Figure 2.12: Scheme of the *Task unit* block.

fully actuated agents able to move on the plane in both $x$ and $y$ directions instantaneously. An application of this algorithm was presented in [31], where it was supposed to apply it to an underwater vehicle modeled as a differential robot. An example of this kind of vehicle is the Mako Autonomous Underwater Vehicle [18] depicted in Fig. 2.13. Considering the single robot kinematic equations for $x$, $y$ holds:

$$\begin{cases} \dot{x}_i(t) &= A(t)_i \cos\phi_i(t) \\ \dot{y}_i(t) &= A(t)_i \sin\phi_i(t) \end{cases}$$




(a) (b)

Figure 2.13: The Mako Autonomous Underwater Vehicle 2.13(a) modeled as a differential mobile robot 2.13(b)

and the robot is controlled by:

$$\begin{cases} A_i(t) &= \sqrt{u_{i,x}^2(t) + u_{i,y}^2(t)} \\ \phi_i(t) &= \text{atan2}(u_{i,y}, u_{i,x}) \end{cases}$$

### 2.3.2 Obstacle avoidance unit

The *Obstacle Avoidance* unit in Fig. 2.7 is devoted to the calculus of the term $V_{obst,i}(t)$ in Eq 2.6. As we have supposed that robots can communicate data to their neighbors and each robot knows the position of the other members of its own communication group, it is possible to exploit these information in order to improve the classical obstacle avoidance algorithms. Before going on with the definition of the obstacle avoidance term, for exposure clarity we define the following potential function:

$$f(X_0, X) = \frac{1}{\frac{1}{\tanh(X_0)} - 1} \cdot \left( \frac{1}{\tanh(X)} - 1 \right) \tag{2.15}$$

whose profile is depicted in Fig. 2.14. It is important to notice that the function $f(X_0, X)$ is equal to 1 in $X = X_0$ and, for $X < X_0$ it tends to 0, for $X > X_0$ it goes to infinity. Although the choice of this function as a potential function for obstacle avoidance may appear an odd choice, in the following it will be clearly explained how it is useful, especially if this obstacle avoidance algorithm is applied to heterogenous group of robots. In particular, the control signal $V_{obst,i}(t)$ for obstacle avoidance is expressed as:

$$V_{obst,i}(t) = \alpha_{O,i} V_{O,i}(t) + \alpha_{E,i} V_{E,i}(t) + \alpha_{R,i} V_{R,i}(t) \tag{2.16}$$

where:

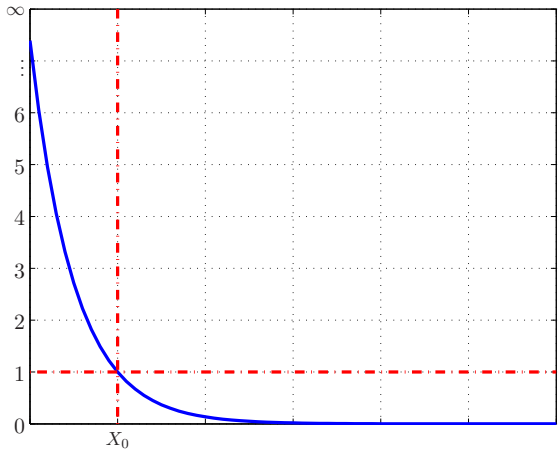



Figure 2.14: Potential function in Eq 2.15.

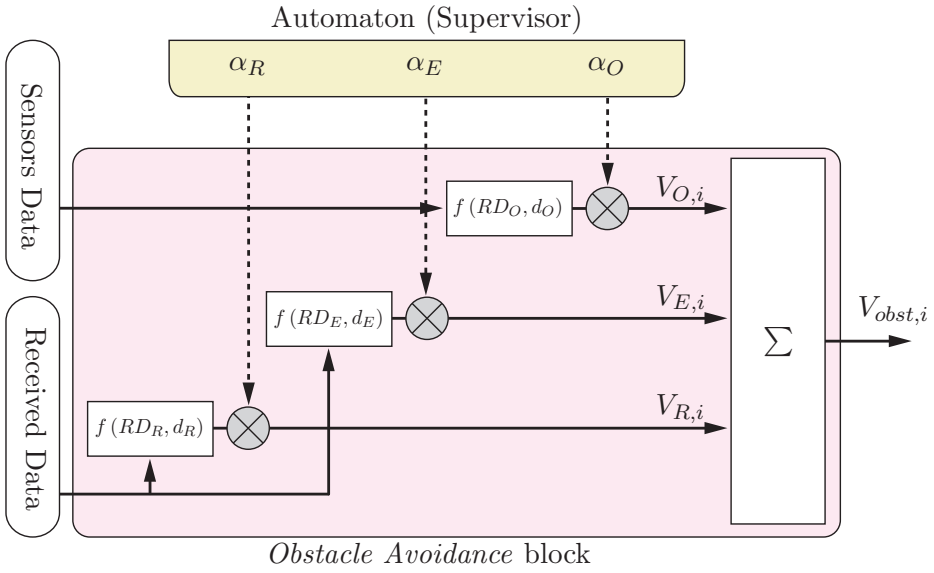



Figure 2.15: Scheme of the *Obstacle Avoidance* block.

- $(\alpha_{O,i} V_{O,i}(t))$ represents the contribution given by the obstacles that are directly detected from the $i$-th robot $R_i$.
- $(\alpha_{R,i} V_{R,i}(t))$ represents the contribution calculated considering the obstacles detected from the other robots of the group, thus the obstacle position can be reconstructed by the robot $R_i$.
- $(\alpha_{E,i} V_{E,i}(t))$ represents the contribution derived from the known position of the other robots of the group $C^i$.

The scalars $\alpha_{O,i}$,$\alpha_{E,i}$, $\alpha_{R,i}$ may be changed in real-time by the controller supervisor in order to define witch kind of obstacles is more important. For example, in some applications it may be more important that the robots do not collide each other rather than avoid environmental obstacles. As $V_{obst,i}(t)$ is calculated with the linear combination of three different contributions, the *obstacle unit* block follows the cooperative paradigm (see Fig. 2.15).

The definition of the three terms in $V_{obst,i}(t)$ is based on the potential defined in Eq 2.15 exploiting different *safety values* $X_0$. In particular, the first component of $V_{obst,i}$ is calculated by considering the sensors measurement: for each sensor that relieves an obstacle at a distance $d_O < L$, the robot calculates a potential field that points to the opposite direction and whose intensity is defined by using Eq. (2.15) as:

$$V_{O,i} = f\,(RD_O, d_O) \tag{2.17}$$

where $RD_O$ is a safe distance (or rescue distance), set in advance (e.g. $RD_O$ is the minimum allowed distance between a robot and an obstacle). We set $RD_O = 1.1 \cdot R_R$.

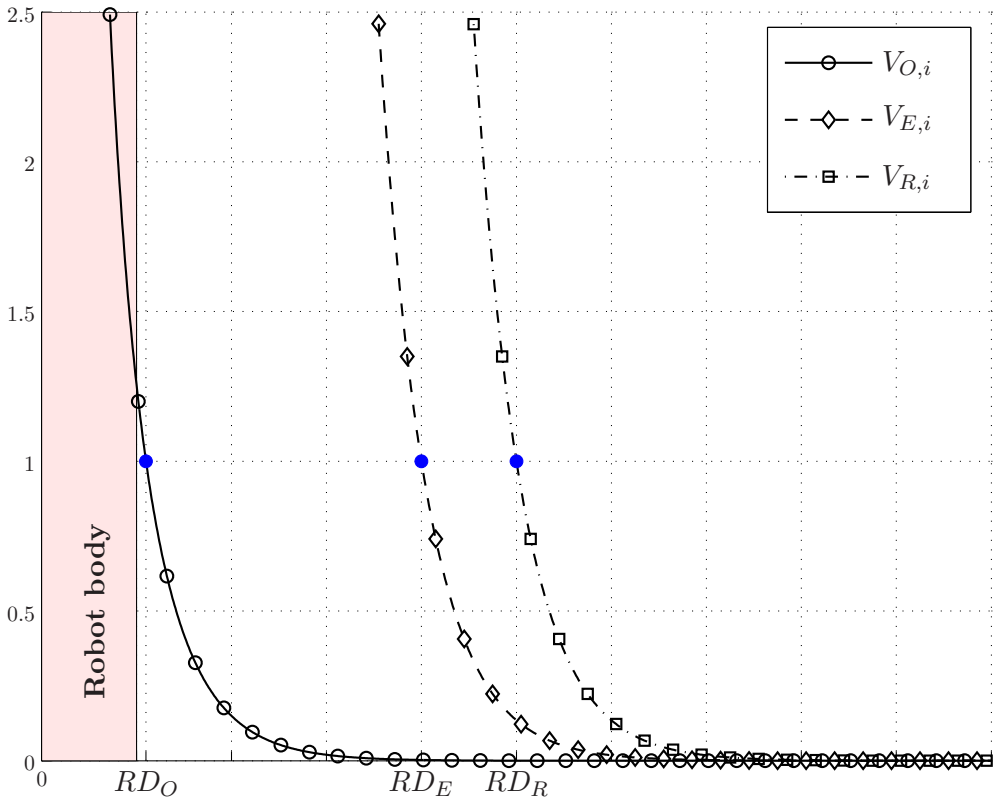



Figure 2.16: Artificial Potential Field for obstacle avoidance.

Note that:

$$\begin{cases} d_O = RD_O \;\Leftrightarrow\; V_{O,i} = 1 \\ d_O > RD_O \;\Leftrightarrow\; V_{O,i} < 1 \end{cases}$$

The second term of the obstacle avoidance function takes into account the positions of the obstacles detected by the other robots in the same communication subgroup. Considering $R_i$ and $C^i$, the position $P_O = [x_O,\ y_O]^T$ of an obstacle detected by $R_i$ is broadcasted to all communicating robots. If $R_j \in C^i$, it calculates an Artificial Potential Field (APF) to take into account the obstacles detected by the others. The expression of the APF used for other obstacles is obtained as in Eq. (2.17):

$$V_{E,i} = f\left(RD_E, d_E\right), \tag{2.18}$$

where $d_E$ is the Euclidean distance between $R_i$ and the other obstacles detected by $R_j$ and the safe distance is $RD_E = 1.1 \cdot R_R$.

The last contribution to the obstacle avoidance component of the control law is given by:

$$V_{R,i} = f(RD_R, d_{R,j}), \tag{2.19}$$

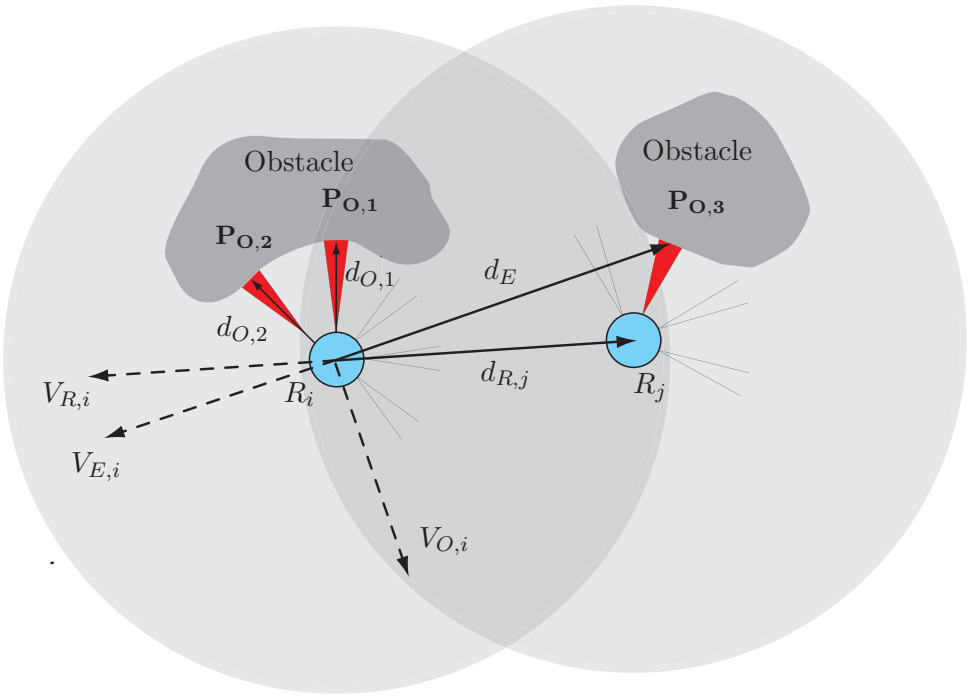



Figure 2.17: Artificial Potential Fields for obstacle avoidance.

where $d_{R,j}$ is the Euclidean distance between $R_i$ and $R_j$, and the safe distance is $RD_R = 2{\cdot}R_R{+}L$. This value ensures that any robot cannot detect others of the same subgroup with its sensors. This means that, in absence of faults, when a robot detects an obstacle, this cannot be another robot and must be an object present in the environment. These three contributions are depicted in Fig. 2.16. In Fig. 2.17 an example with 2 communicating robots is shown: $P_{O,1}$ and $P_{O,2}$ are the points of the obstacle detected by the robot $R_i$, while $P_{O,3}$ is the point of the obstacle detected by $R_j$.

Let's consider now the overall control structure depicted in Fig. 2.7. The blocks *Task Unit* and *Obstacle Unit*, as explained in previous sections, generate the control action to accomplish one or more predefined tasks and to avoid obstacles respectively. The typical problem that arises when dealing with obstacles avoidance in unknown environments is the possibility of getting stuck in a local minima, i.e. a robot or the entire group is not able to avoid obstacles and gets stuck in a zone of the map other than the final one. To recognize these dangerous situations we have introduced the *MEM* block: during the execution of the algorithm, the $i$-th robot memorizes the last $m$ positions (if an obstacle is detected) and, if the mean value of the memorized data is lower than the robot radius $R_R$, it detects the local minima and send the information to the supervisor.

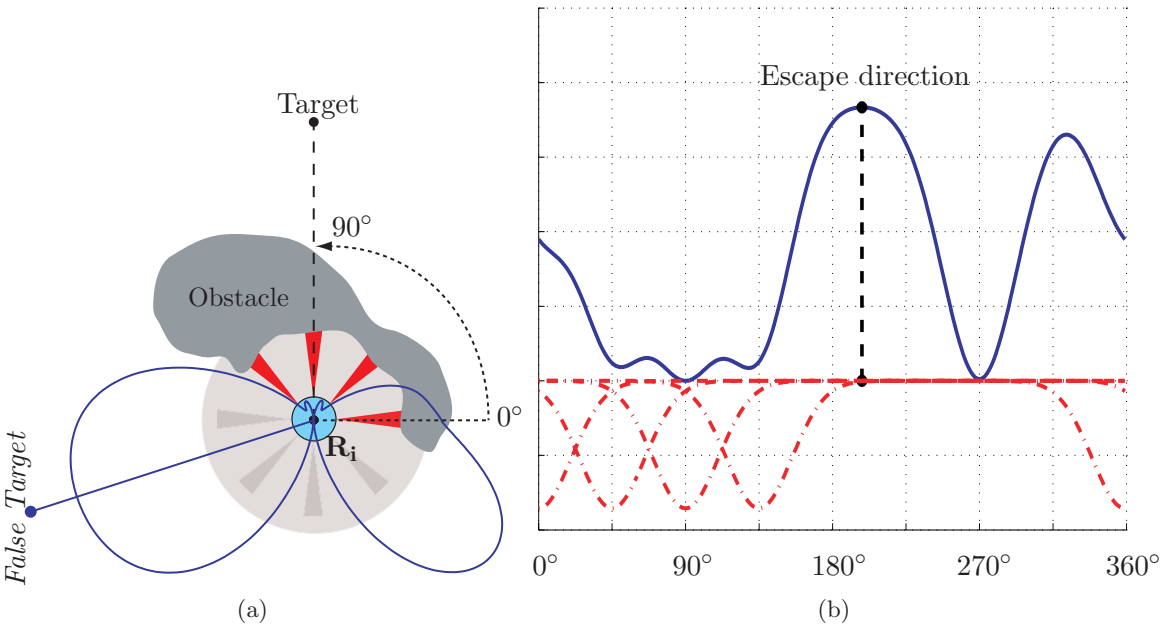



Figure 2.18: Local minima escape example: example (a) and corresponding probability profile (b, solid line) and single Gaussian curves (b, dash-dotted lines).

The test performed by the *MEM* block is:

$$\begin{cases} \left| \frac{1}{m} \sum_{h=1}^{m} q_{i,h} - q_i(t) \right| \leq R_R \rightarrow \rho(t) = 1 \quad \text{(local minima)} \\ \left| \frac{1}{m} \sum_{h=1}^{m} q_{i,h} - q_i(t) \right| > R_R \rightarrow \rho(t) = 0 \quad \text{(moving)} \end{cases} \tag{2.20}$$

where $\rho(t)$ is a flag transmitted to the supervisor so it can modify the $\alpha$-tasks multipliers. The number of samples $m$ memorized by the *MEM* block can be tuned: in fact, for a low number of samples, the robot can detect a local minima also if it is not stuck but is moving slowly; on the other hand, a large number of samples can prevent the detection of a local minima in a short time.

Once a local minima is detected, the robot has to calculate a smart strategy to avoid it. Our local-minima-escape strategy is based on the idea of exploiting a virtual target (or *false target*) that is temporary assumed as the new point that the robot has to reach to accomplish its primary mission.

The new false target is defined by choosing a point at the same distance of the real target. The angle to identify the escape direction is defined considering the sensors data: as can be seen in Fig. 2.18, a probabilistic curve is designed by the entrapped robot as a sum of negative Gaussian curves centered on the direction where sensors have detected

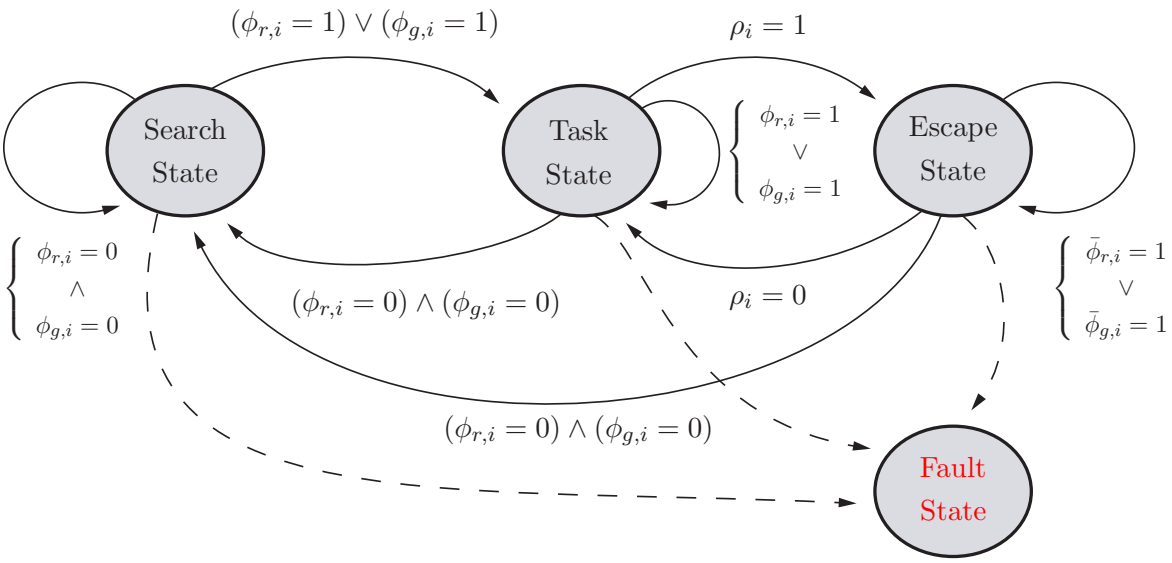



Figure 2.19: Automaton scheme

a critical obstacle. Moreover, to avoid deadlock situations where the robot moves in the opposite direction w.r.t. to real target, that direction is strongly penalized.

The last element to be analyzed in the high-level controller depicted in Fig. 2.7 is the *Automaton* block, that can be considered the core of the supervisor. In fact, using the information received from the *MEM* block and from the *Obstacle Unit* block, it is devoted to change the weights introduced in Sec 2.3.1-2.3.2. Its behavior is summarized in Fig. 2.19 where, depending on the current state, the output information-set is changed run-time.

For clearance of representation, we introduce the following flag functions with reference to the $i$-th robot and its communication subgroup $C^i$:

$$\phi_{r,i} = \begin{cases} 1 & \text{if } \mathrm{R_i} \text{ acquires target} \\ 0 & \text{otherwise} \end{cases} \qquad \bar{\phi}_{r,i} = \begin{cases} 1 & \text{if } \rho_\mathrm{i} = 1 \ \wedge \ \mathrm{R_i} \text{ define } \textit{false target} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_{g,i} = \begin{cases} 1 & \text{if } \exists\ \mathrm{R_h} \in \mathrm{C^i} : \phi_{\mathrm{r,h}} = 1 \\ 0 & \text{otherwise} \end{cases} \qquad \bar{\phi}_{g,i} = \begin{cases} 1 & \text{if } \exists\ \mathrm{R_h} \in \mathrm{C^i} : \rho_\mathrm{h} \wedge \bar{\phi}_{\mathrm{r,h}} = 1 \\ 0 & \text{otherwise} \end{cases}$$

In order to create more realistic simulations, a *fault state* has been introduced to take into account the possibility of faults. In particular, we suppose that this state is activated in case of motor faults, communication faults and sensor faults.

The description of the states of the automaton is reported:

**Search State:** Any robot in $C^i$ has acquired a target. Each robot is moving randomly trying to be as far as possible from the others.

**Task State:** The target has been acquired by at least one robot in $C^i$. Using shared data, every robot in $C^i$ knows the target position. Using *swarm behavior functions* and

avoiding obstacles each robot tries to match its subgroup mean value with target's position (see Subsec 2.3.1).

**Escape State:** A robot $R_h \in C^i$ is in a local minimum. To escape, it tries to reach a *false target*.

**Fault State:** The current robot is in fault condition.

To conclude the analysis of the high level control scheme depicted in Fig. 2.7, note that two blocks named respectively $SAT_1$ and $SAT_2$ has been introduced after the calculus of $V_{task}$ and after the calculus of $u_i = V_{task} + V_{obst}$. The meaning of these two blocks is easy to explain: the block $SAT_1$ has been introduce in order to limit the norm of the maximum control action that can be generated by the *Task Block*. This justifies the odd choice made for the potential field used for obstacle avoidance: since the norm of the control action can not exceed the value of 1, the obstacle avoidance will always create an avoidance control action that can prevent the robot going too close to the obstacles. The block $SAT_2$ has been introduce to normalize the control output so that it can be applied to different kind of actuators. Note that to get a control action as smooth as possible, the saturation can be replaced by the function $-1 \leq \tanh(x) \leq 1$.

## 2.4 Simulation Results

The control algorithm presented in this chapter has been validated by means of numerical simulations using Matlab. The environment created is a 4 x 4 m. arena, where many round obstacles with radius of 30 cm. are randomly placed. As we are interested in testing our algorithm in configurations with many local minima, the obstacles are randomly merged together in order to create more complex environments. The model of the robots used in our simulations has the following characteristics:

- Robot radius $R_R = 1$.
- Eight proximity sensors equally spaced around the robot and with a maximum range $L = 3$.
- Communication range $R^i_{COMM} = 40$ (represented as a dotted circle).
- Target visibility range $R^i_{TARG} = \infty$.

The multipliers used by the high level control are summarized in the following table:

| Automaton State | $\alpha_1$ | $\alpha_2$ | $\alpha_3$ |
|---|---|---|---|
| Search | 1 | 0 | 0 |
| Task | 1 | 1 | 1 |
| Escape | 1 | 0.1 | 0 |
| Fault | 0 | 0 | 0 |

The data are collected over 100 simulation runs, with fixed target at $[100, 0]^T$. In Fig. 2.20 the evolution of the mean value and the standard deviation of the distance of a single robot from the fixed target is depicted. In the arena 34 obstacles are present and the starting point is randomly chosen a zone defined by in $x \in [-170 \ldots -100]$ cm. and $y \in [-170 \ldots 170]$ cm.

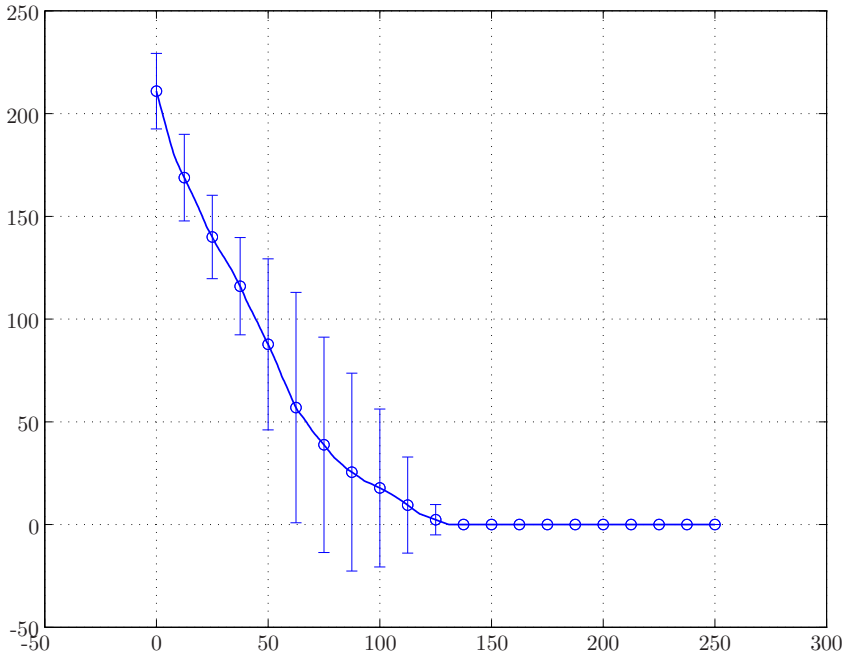
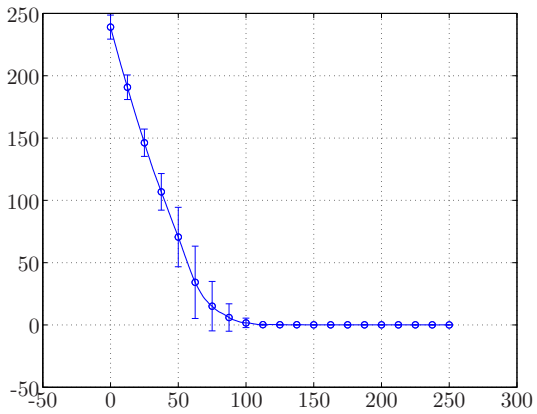
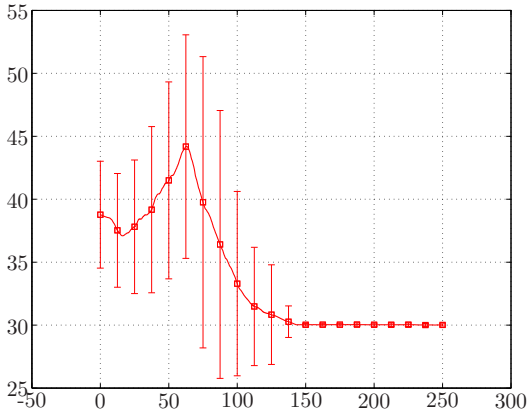



Figure 2.20: Evolution of the mean value and the standard deviation of the distance of a single robot from the fixed target.

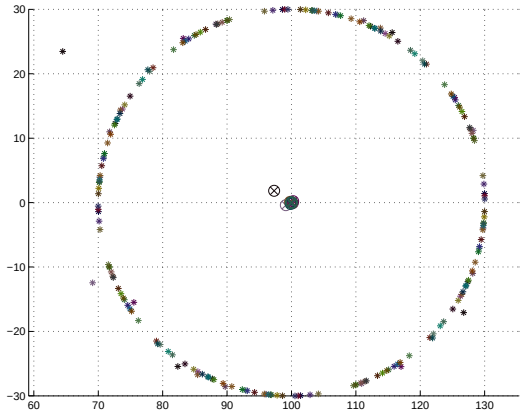
Considering four robots moving in the same environment, the evolution of the radius of the formation and the distance from the target of the formation mean value in case of fixed are depicted in Fig. 2.21(a) and in Fig. 2.21(b) respectively. The starting positions are randomly chosen in a zone located in $x \in [-170 \ldots -100]$ cm. and $y \in [-170 \ldots -70]$ cm. In Fig. 2.21(c) the final robots positions ($*$) and the final centroid positions ($\otimes$) are shown; in Fig. 2.21(d) a statistical graph summarizes the percentage of tests collected w.r.t. the final distance to the target.



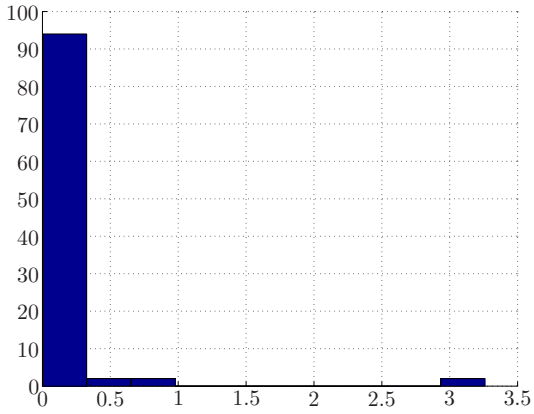

(a) Evolution of the distance between group centroid and static target.




(b) Evolution of the group radius.




(c) Final robots ($*$) and centroid ($\otimes$) positions.




(d) percentage of results collected w.r.t. the final distance between the centroid and the target.

Figure 2.21: Statistical data about four-agent fixed-target simulations.

The simulations with moving target are performed with the same starting conditions but on the same environment, that is depicted in Fig. 2.22, where a simulation is reported: note that in absence of obstacles, the mean value of the swarm (dash-dotted line) converges to the target's trajectory (dashed line). The target path is generated using spline passing through four fixed points and do not take care about obstacles. Statistical data are collected over 100 simulations and reported in Fig. 2.23

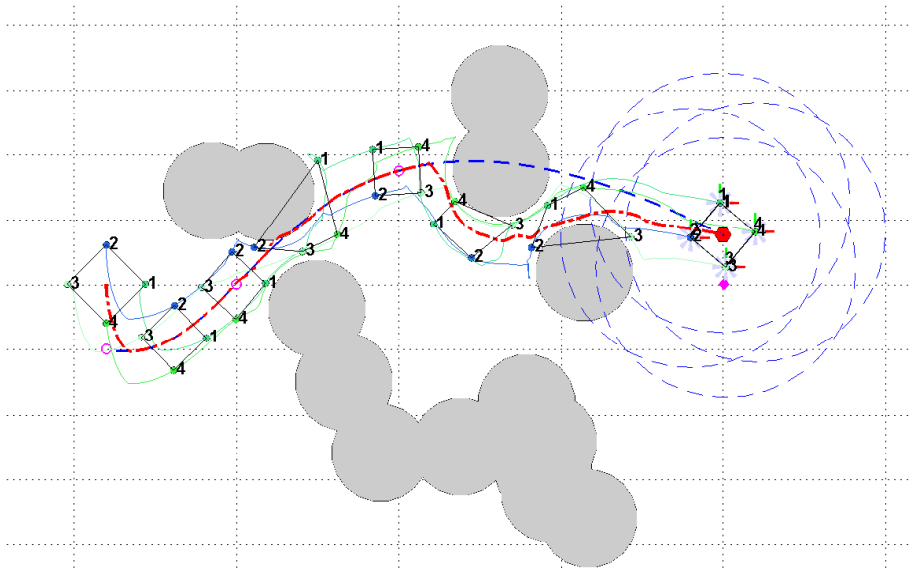
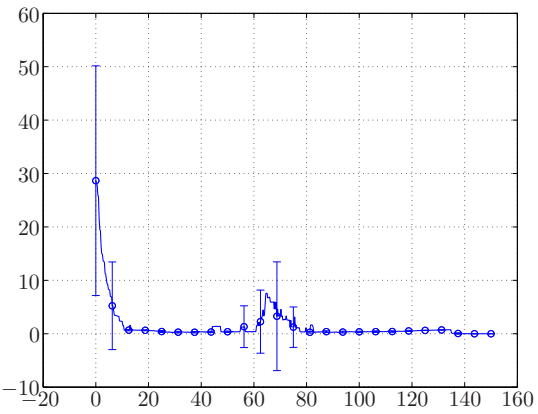
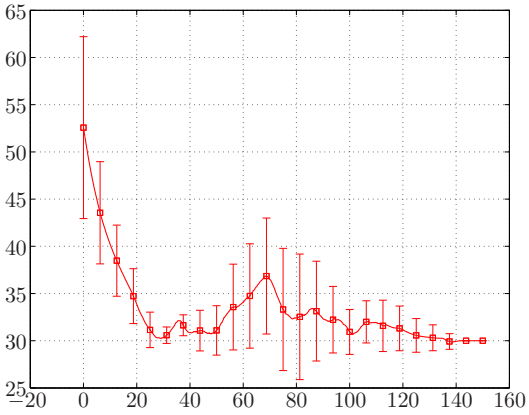


Figure 2.22: Simulation field with random obstacles and moving target.




(a) Evolution of the distance between group centroid and moving target.




(b) Evolution of the group radius in case of moving target.

Figure 2.23: Statistical data about four-agent fixed-target simulations.

# Chapter 3

# Graph Theory

In this chapter, basic notions on graph theory are provided, and many famous theorems are reported. To clarify the concepts exposed below, many examples are provided by the author. Moreover, the *consensus problem* is explained and some applications of the classical solution are shown. These basic concepts will be used in Chapter ?? and applied to a system of real robots.

## 3.1 Introduction on Graph Theory

### 3.1.1 Basic notions about graph theory

In mathematics and computer science, graph theory is the study of graphs, that are mathematical structures used to model relations between objects of a certain collection. From a historical point of view, the graph theory was initially used to solve optimization problems: a famous example is the *Seven Bridges of Konigsberg* problem. In a paper published by Leonard Euler in 1736, the author discuss about the possibility of crossing all the seven bridges of Konigsberg only one time. In Fig. 3.1 the problem is reported in its original version.

In more recent years, graph theory has been studied as a new way to solve many different problems, such as traffic routing problems [72], payload transport, task assignment, air traffic control and many other applications, included robotics. Before dealing with some standard application of these concepts, some basic definitions and notions are introduced [21].

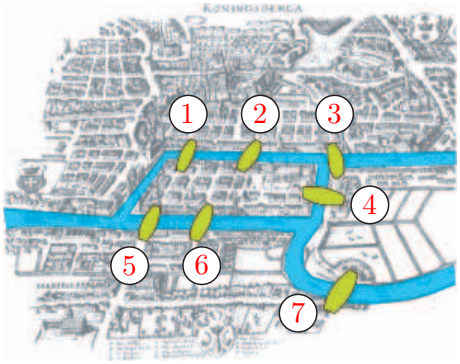



Figure 3.1: Konigsberg bridges

**Definition 3.1** (Undirected graph)**.** *An interconnected undirected graph $G$ with $N$ elements can be defined by a couple of sets $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the* vertex set *(or* node set*) and $\mathcal{E}$ is the* edge set *and represents the connections between couples of nodes:*

- $V = \{n_i, i = 1 \dots N\}$,
- $E = \{(n_i, n_j) \subseteq V \times V \mid n_i \neq n_j\}$,

*As we do not consider any direction map over $E$, the elements of this set are unordered pairwise of nodes.*

Let's note that $E \subseteq V \times V$, i.e. many connections couldn't exist.

**Definition 3.2** (Subgraph)**.** *Given a graph $G = (V, E)$, a subgraph is defined as $\bar{G} = \{\bar{V}, \bar{E}\}$, where $\bar{V} \subseteq V$ and $\bar{E} \subseteq E$*

**Definition 3.3** (Path over G)**.** *A* path *$P(n_0, n_L)$ over a given graph $G$ connecting two nodes $n_0$, $n_L$ is defined as a finite sequence of nodes $n_i$ such that $n_{k-1} \neq n_k$, $k = 1 \dots L$*

**Definition 3.4** (Connected graph)**.** *A graph is said to be connected if a path $P(n_i, n_j)$ exists for each couple of nodes $(n_i, n_j)$, $n_i \neq n_j$.*

**Definition 3.5** (Neighbors set (or friend set))**.** *Given a connected graph $G = (V, E)$ and a node $n_i \in G$, the neighbors set $N(n_i)$ is defined as $N(n_i) = \{n_j \in V : (n_i, n_j) \in E\}$*

**Definition 3.6** (Degree of a node)**.** *Given a connected graph $G = (V, E)$ and a node $n_i \in V$, the degree of the node $n_i$ is given by the number of its neighbors, i.e.* $\deg(n_i) = N(n_i)$.

From the definition of *degree* of a node follows:

**Definition 3.7** (k-regular graph)**.** *Given a graph $G = (V, E)$ with $N$ nodes, the graph is said to be $k$-regular if all the nodes have the same number of neighbors, i.e.*

$$G \text{ is } k - \text{regular} \Leftrightarrow N(n_i) = N(n_j), \forall n_i, n_j \in V$$

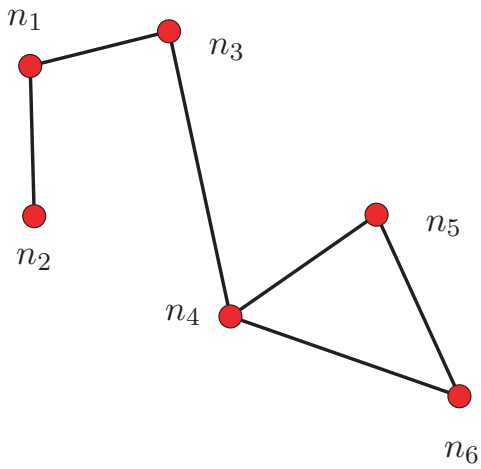



Figure 3.2: Example of undirected graph $G$

A complete graph can be defined also considering the degree of the nodes: indeed, a graph with $N$ nodes is complete if it is $k$-regular and if $k = N - 1$ (no self loops are considered)

**Example 3.1.** *As an example, let's consider the graph reported in Fig. 3.2. In this six-nodes graph, the two sets that defines $G$ are:*

- $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$,
- $E = \{\{n_1, n_2\}, \{n_1, n_3\}, \{n_3, n_4\}, \{n_4, n_5\}, \{n_4, n_6\}\}$,
- $N(n_4) = \{n_3, n_5, n_7\}$
- $\text{Path}(n_2, n_3) = \{n_2, n_1, n_3\}$

Once an orientation map is defined on a graph $G$, it is called *oriented graph*. In particular, an oriented graph is defined as:

**Definition 3.8** (Oriented graph)**.** *Given a graph $G$ and an orientation map $M$ over it, the resulting directed graph can be defined as $G = \{V, E_M\}$, where*

- $E_M = \{(n_i, n_j), (n_j, n_i) \subseteq V \times V \mid n_i \neq n_j, (n_i, n_j) = -(n_j, n_i)\}$

To conclude, a weight map can be defined on elements of both vertex set and edge set. In these cases, weighted graphs are defined.

**Definition 3.9** (Weighted graph)**.** *Given a graph $G$, a weight map over the vertex set $\mathcal{W}_V$ and a weight map over the edge set $\mathcal{W}_E$, three new graphs can be defined:*

1. *Edge-weighted graph $G_{\mathcal{W}_E} = (V, E, \mathcal{W}_E)$*

2. *vertex-weighted graph* $G_{\mathcal{W}_V} = (V, E, \mathcal{W}_V)$

3. *edge-vertex weighted graph* $G_{\mathcal{W}_E,\mathcal{W}_V} = (V, E, \mathcal{W}_E, \mathcal{W}_V)$

Note that the two maps $\mathcal{W}_V$ and $\mathcal{W}_E$ can be defined dynamically as long as also the edge set $E$ can be time-variable. Generally, a weight defined over an edge of a graph can be interpreted, for example, as a cost, as a distance or a capacity and, in general, a cost function can be used to define a weight map.

As will see in Section 3.1.2, the orientation map, the edge-weight map and the node-weight map can be encoded in an appropriate matrices.

### 3.1.2 Algebra in graph theory

An interesting way to study the properties of a graph is to find a method to write its structure in a mathematical mode. To study the system represented by a graph, in this paragraph the most important matrices that define a graph and their spectral properties are exposed.

**Definition 3.10** (Adjacency matrix)**.** *Given a graph* $G = (V, E)$ *with* $N$ *nodes, the* Adjacency matrix $\mathcal{A} \in \mathbb{R}^{N\times N}$ *is a binary matrix that is defined as:*

$$\mathcal{A}(i,j) = \begin{cases} 1, & \text{if } (n_i, n_j) \in E \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

**Example 3.2.** *With reference to Fig. 3.2, the adjacency matrix is:*

$$\mathcal{A} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{3.2}$$

In case $G$ is a complete graph, the adjacency matrix is a unit matrix except for zeros on the main diagonal. The definition of the adjacency matrix is the same also for undirected graphs, and some properties of $\mathcal{A}$ can be extrapolated:

- $\mathcal{A}$ is symmetric;
- $\mathcal{A}$ has a complete set of *real* eigenvalues;
- $\mathcal{A}$ has an orthogonal eigenvector basis.

**Definition 3.11** (Degree matrix)**.** *Given a graph $G = (V, E)$ with $N$ nodes, the* degree matrix *of $G$ is a square matrix $\mathcal{D} \in \mathbb{R}^{N \times N}$ defined as:*

$$\mathcal{D}_{i,j} = \begin{cases} \deg(n_i) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

*where $\deg(n_i) = |N(n_i)|$ is the* degree *of the node $n_i$.*

Once the adjacency matrix is defined, the *degree matrix* can be constructed starting from a zeros matrix and setting the values on the diagonal as the sum of the elements of the corresponding row of the adjacency matrix. This means that:

$$\mathcal{D}_{i,j} = \begin{cases} \sum_{i=1}^{N} a(i,j) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

**Example 3.3.** *The degree matrix for the graph depicted in Fig. 3.2 is:*

$$\mathcal{D} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix} \tag{3.5}$$

A new matrix, that collect all the data stored in the adjacency matrix and in the degree matrix, can be defined. It is called *Laplacian* matrix and is usually defined as follows:

**Definition 3.12** (Laplacian matrix)**.** *Given a graph $G = (V, E)$ with $N$ nodes, the* Laplacian matrix *$\mathcal{L} \in \mathbb{R}^{N \times N}$ of $G$ (called also* admittance matrix *or* Kirchhoff matrix*) is defined as $\mathcal{L} = \mathcal{D} - \mathcal{A}$ or, more explicitly, as:*

$$\mathcal{L}(i,j) = \begin{cases} \deg(n_i) & \text{if } i = j \\ -1 & \text{if } i \neq j,\ (n_i, n_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

**Example 3.4.** *Considering once again the graph in Fig. 3.2, the corresponding Laplacian matrix is:*

$$\mathcal{L} = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 \end{bmatrix} \tag{3.7}$$

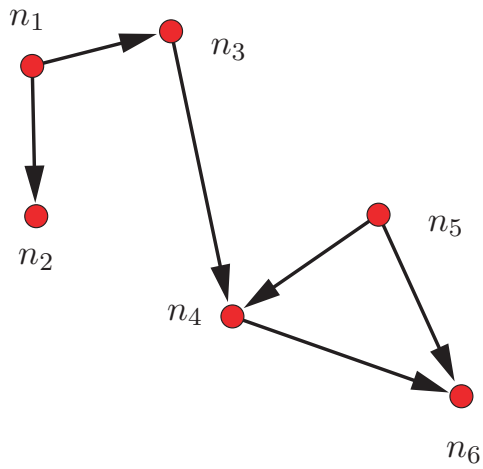



Figure 3.3: Example of oriented graph: on orientation map is applied to the graph depicted in Fig. 3.2

The last matrix that is useful to define over a given graph $G = (V, E)$ is the *incidence matrix*.

**Definition 3.13** (Incidence matrix)**.** *Given a connected graph $G = (V, E)$ and an orientation map $M$ over $G$, the* incidence matrix $\mathcal{I} \in \mathbb{R}^{N \times |E|}$ *is defined as:*

$$i_{i,k} = \begin{cases} -1 & \text{if } e_k = (n_i, n_j) \\ 1 & \text{if } e_k = (n_j, n_i) \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

*where $|E|$ is the cardinality of the edge set and $e_k$ is the $k$-th edge of $G$.*

Roughly speaking, the incidence matrix is a low rectangular matrix (square matrix only if $|E| = N$, i.e. $G$ is a single-path graph) where $k$-th column contains 1 in the $i$-th row if the $n_i$ node is the *tail* node (or starting node) for the edge $e_k$, contains -1 in the $i$-th row if the $n_i$ node is the *head* node (or ending node) for the edge $e_k$, and 0 otherwise. It is important to notice that over an undirected graph $G$ can be defined many different orientation maps: in fact, all the incidence matrices defined over an undirected graph $G$ differ only by inverting the direction of some edges. The most important properties of the incidence matrix is that it allows to define the Laplacian matrix of a graph as $\mathcal{L} = \mathcal{I} \cdot \mathcal{I}^T = \mathcal{D} - \mathcal{A}$.

**Example 3.5.** *An example of incidence matrix over the graph reported in Fig. 3.2 is:*

$$\mathcal{I} = \begin{bmatrix} -1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{3.9}$$

*and the corresponding oriented graph is depicted in Fig. 3.3.*

As the Laplacian matrix collect all the information that can be used to represent a graph, it is generally considered the most important matrix in graph theory. For this reason its spectral properties have been widely studied. Some of the properties that will be used in this work are summarized below.

**Theorem 3.1.** *Given a connected graph $G = (V, E)$, the corresponding Laplacian matrix is always positive-semidefinite: $\forall i,\ \lambda_i \geq 0$.*

This means that if the spectrum of $\mathcal{L}$ is defined as $\Lambda(\mathcal{L}) = \{\lambda_0 \dots \lambda_n\}$, all the elements of $\Lambda(\mathcal{L})$ can be rearranged in a non-decreasing order:

$$\begin{cases} \lambda_0 \leq \lambda_1 \leq \lambda_2 \dots \leq \lambda_n \\ \lambda_0 = 0 \end{cases} \tag{3.10}$$

The fact that $\lambda_0 = 0$ follows directly from the definition of $\mathcal{L}$: in fact, as all the row-sums are equal to zero, means that all the columns of $\mathcal{L}$ are linear-independent except one. Therefore, there must be a null eigenvalue. Consequently, this means that the *null space* of $\mathcal{L}$ is always a non-empty subspace with at least one eigenvector corresponding to the null eigenvalue. In particular, the following theorem holds:

**Theorem 3.2.** *Given a graph $G = (V, E)$ and the corresponding Laplacian matrix $\mathcal{L}$, the number of non-connected components (or disjointed subgraph) in $G$ is equal to the multiplicity of the $\lambda_0$ eigenvalue of $\mathcal{L}$. More generally, given a graph $G = (V, E)$ with $N$ nodes and $C$ connected components, then $rank(\mathcal{L}) = N - C$.*

**Example 3.6.** *For the graph represented in Fig. 3.2 (witch Laplacian matrix $\mathcal{L}$ is reported in Example 3.4), the spectrum of $\mathcal{L}$ is:*

$$\Lambda(\mathcal{L}) = \{0.0,\ 0.3249,\ 1.4608,\ 3.0,\ 3.0,\ 4.2143\}$$

*If we cancel the edge connecting the third node to the fourth one, we get a new graph composed of two separated subsets. Analyzing the spectrum of this new graph $\tilde{G} = (V, \tilde{E})$, where $\tilde{E} = \{E - \{(n_3, n_4)\}\}$, it is:*

$$\Lambda(\tilde{\mathcal{L}}) = \{0.0,\ 0.0,\ 1.0,\ 3.0,\ 3.0,\ 3.0\}$$

*where the Laplacian matrix $\tilde{\mathcal{L}}$ differs from $\mathcal{L}$. Let's note that the multiplicity of the null eigenvalue is two.*

In case we deal with connected graphs, it is proved that

$$0 < \lambda_1 \leq \lambda_2 \ldots \leq \lambda_n \tag{3.11}$$

Starting from these considerations, it follows: $eigv(\mathcal{L}) = \{\nu_1 \ldots \nu_N\}$, with $null(\mathcal{L}) = span(\nu_1) = span(\mathbf{1})$, where $\mathbf{1}$ is a $n$-dimensional vector of 1.

**Definition 3.14** (Algebraic connectivity)**.** *Given a connected graph $G = (V, E)$, the smallest non-zero eigenvalue is called* algebraic connectivity *of $G$*

The algebraic connectivity of a graph is upper bounded by a value that depends on the graph configuration:

**Theorem 3.3.** *Given a connected graph $G = (V, E)$ with Laplacian matrix $\mathcal{L}$, for the eigenvalues of $\mathcal{L}$ the following propositions hold:*

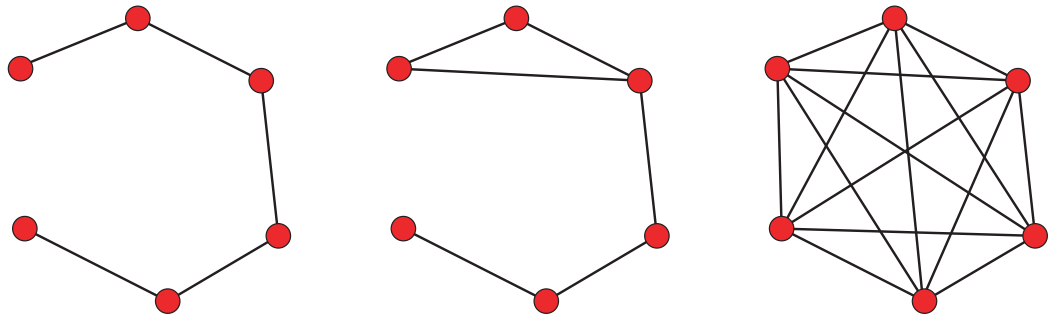
- $\lambda_1 \leq \dfrac{N}{N-1} min\{\deg(n_i), \forall v_i \in V\}$*, i.e. the algebraic connectivity is upper bounded;*
- $\sum_{i=1}^{N} \lambda_i = 2|E| = \sum_{i=1}^{N} \deg(n_i)$

In graph theory, the *algebraic connectivity* is usually called *Fiedler value* (from the name of Miroslav Fiedler, the father of the theory of algebraic connectivity [24], [26]). Empirically, another widely used interpretation of the algebraic connectivity of a graph can be found in literature: the value of $\lambda_1$ increase as long as increase the number of edges in a graph. Therefore one can says that the value of the algebraic connectivity is a good index to understand how strong the graph is connected: if the value of $\lambda_1$ is low the graph can be divided cutting few edges, viceversa if $\lambda_1$ is high many edges must be cut to split the starting graph.

In Fig. 3.4 three different edge sets are used on the same vertex set. As can be seen, as long as the the number of edges increases, the value of $\lambda_1$ tends to its maximum value. Note that the maximum value of the algebraic connectivity is equal to the number of vertices. The algebraic connectivity $\lambda_1$ can be used also to calculate the convergence ratio of the swarm to the centroid. It holds:

$$\|x(t) - \mathbf{1}\| \leq \|x(0) - \mathbf{1}\| e^{-\lambda_1 t} \tag{3.12}$$

It's easy to understand why the convergence ratio is upper bounded by a value related to the algebraic connectivity: in fact, $\lambda_1$ is the lower eigenvalue of the system and corresponds to the higher time constant, i.e. the part of the system related to $\lambda_1$ is the slower.



(a) $\lambda_1 = 0.2679$ (b) $\lambda_1 = 0.3249$ (c) $\lambda_1 = 6.000$

Figure 3.4: Examples of different values of $\lambda_1$ for different edge sets on the same vertex set

## 3.2 Control in Graph Theory: the consensus algorithm

Literally, *consensus* means that an interconnected system is able to reach an agreement regarding a certain quantity of interest. The first formulation of the problem appeared in 1960's (see [29]) and a *statistical* formulation of the problem was developed to manage information collected using sensor networks working under uncertainties [9].

The agreement problem was later solved by Borkar and Varaiya [15] for distributed decision making systems. This work was summarized in [11] and was widely applied to parallel computing.

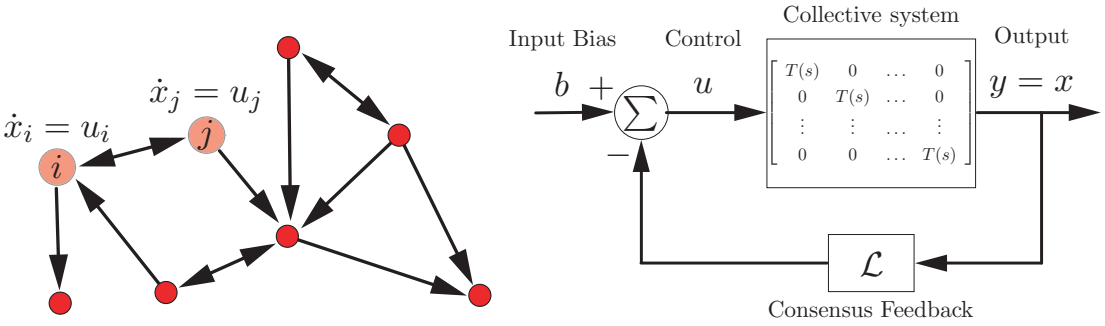
In case of *agents* (or dynamic systems) the agreement reached by the system depend on the state of all the agents. A typical way to solve this problem is to define interaction rules in order to explicit the information exchange between an agent and all its neighbors. In our case, the idea of using graph to represent an agent network sounds natural: in this case, in fact, it will be possible to use all the results collected in graph theory that are summarized in the previous section. Moreover, the spectral analysis of matrices used to describe a graph (overall the Laplacian matrix) can be automatically applied to the dynamic analysis of a group of agents

Given a group of $N$ agents, the simplest model that can be used is the single integrator, i.e. $u_i(t) = \dot{x}_i(t)$, and the goal of the consensus algorithm is to drive all the agents to a final common value $x_f$.

If we model the connections between agents as a connected graph $G = (V, E)$, a simple algorithm regarding the state of the system can be found in [2] and can be expressed as a $n$-th order linear system:

$$\dot{x}_i(t) = -\sum_{j \in N(i)} (x_i(t) - x_j(t)) + b_i(t), \text{ where } \begin{cases} x_i(0) &= x_{i,0} \in \mathbb{R} \\ b_i(t) &= 0 \end{cases} \tag{3.13}$$

where $N(n_i)$ is the *neighbor set* of agent $x_i$ (see def 3.5). The role of the input value




(a) (b)

Figure 3.5: Two equivalent forms of the consensus algorithm: 3.5(a) a networks of agents modeled as integrators and 3.5(b) the feedback loop that performs the algorithm on a MIMO system.

$b_i(t)$ will be described later. Note that, setting $b_i(t) = 0$, the collective behavior can be summarized in:

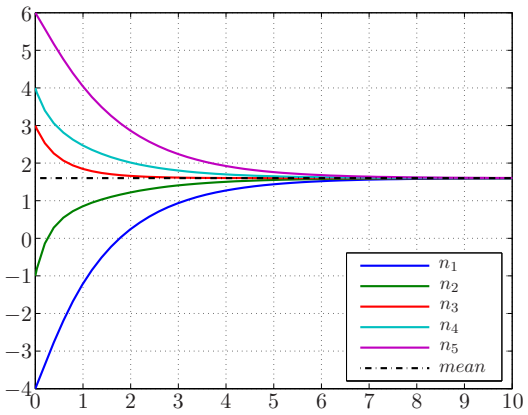
$$\dot{x} = -\mathcal{L} \cdot x \tag{3.14}$$

The two equivalent formulations of the consensus algorithm explained in this section are reported in Fig. 3.5. In particular it is possible to see that there is a direct correspondence between a network of agents modeled as single integrators where the $i$-th node receive information about the state of all the neighbors node $x_j \in N(i)$, and the block diagrams of interconnected dynamic systems with identical transfer functions $T(s) = \frac{1}{s}$. The *collective system* is a MIMO (Multi In - Multi Out) system represented with a diagonal matrix. This algorithm is of interesting in robotics because of its decentralized nature: each agent performing the consensus act only based on information received from local neighbors and does not need a supervisor.

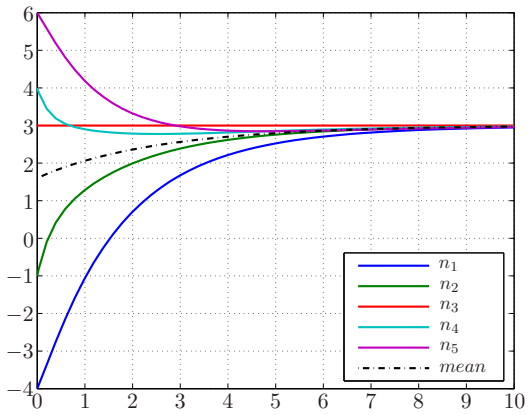
Recalling the definition of the Laplacian matrix of a graph (Def 3.12) and the theorem related to the eigenvalues of this matrix (Thm 3.1), as $\mathbf{1}$ belongs to the null space of $\mathcal{L}$ ($\mathcal{L} \cdot \mathbf{1} = \mathbf{0}$), the zero-eigenvalue $\lambda_0$ corresponds to the eigenvector $\mathbf{1} = [1 \ \dots \ 1]^T$. This means that the consensus feedback is globally asymptotically stable and drives the system to an equilibrium point that, in the state-space, is a $N$-dimensional vector

$$\bar{x} = \begin{bmatrix} \alpha \\ \vdots \\ \alpha \end{bmatrix} = \alpha \cdot \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \tag{3.15}$$

Once all the agents reach the equilibrium, the time-derivative in Eq 3.13 is null, i.e. $\dot{x}(t) = 0$. Starting from this consideration, it is easy to demonstrate that the $\alpha$ value coincides with the mean value of the agents starting positions, i.e. $\alpha$ is a function of the




(a) Time evolution of a five-integrators system performing consensus.




(b) Evolution of a four-integrators system and a constant: note that the mean value tends to the constant agent

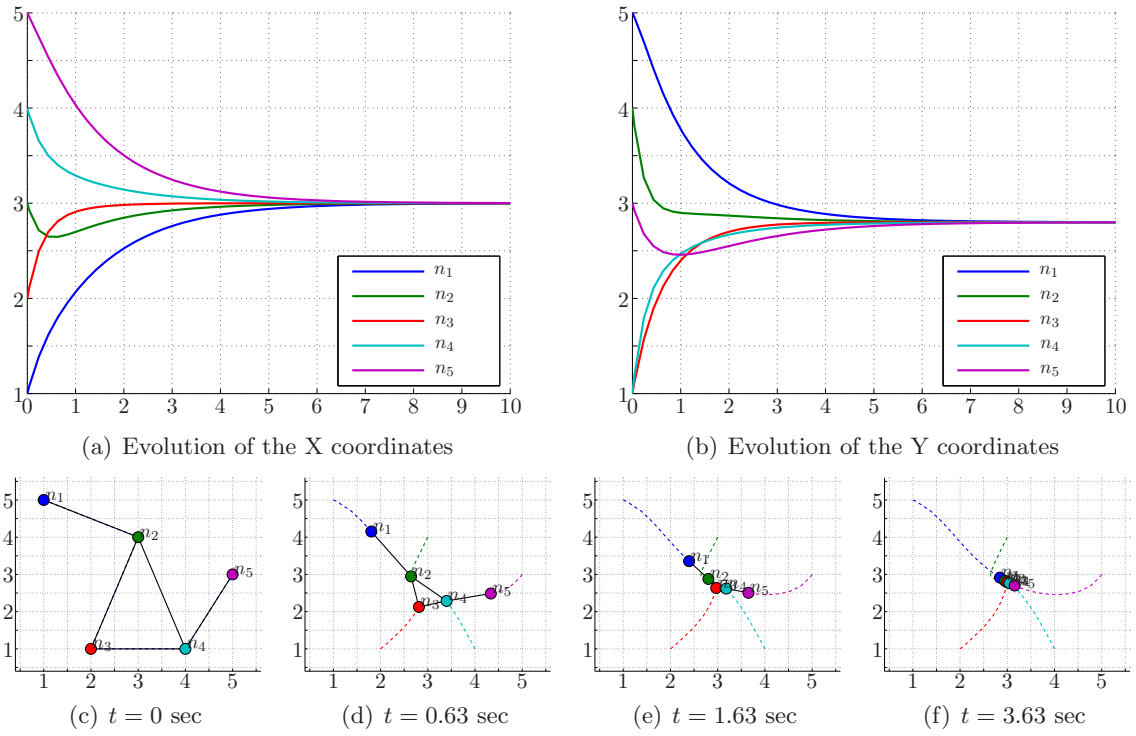
Figure 3.6: Evolution of system performing consensus

starting position vector:

$$\alpha(x_{i,0}) = \frac{1}{N}\sum_{i=1}^{N} x_i(0) \tag{3.16}$$

If we think about the agents as points without mass, till this point we can imagine that the state vector $x(t) = [x_1, \ \dots \ x_N]^T$ represents their positions on a line, i.e. we have supposed that all the agents can move on single dimension. In Fig. 3.6(a) an example of the evolution of the states of a system with five agents versus time is depicted. Note that the mean value (dash-dotted line) is always constant in time. In Fig. 3.6(b) the same example is depicted, but the third agent (starting position is 3) is fixed: in this case, as a consequence of the property expressed in eq 3.15-3.16, all the states tend to the locked one.

Because of the decentralized nature of this algorithm, we are interested in using it to guarantee the convergence of a group of robots modeled as particle points without mass. Let's see now how to extend the Laplacian-based consensus algorithm to a $m$-dimensional space. The key point is that a particle point can be modeled as single integrator in all dimensions. The control loop depicted in Fig. 3.5(b) can be slightly modified: the *collective system block* is a MIMO system with $(m \cdot N) \times (m \cdot N)$ transfer function matrix with $T(s) = \frac{1}{s}$ on diagonal; the *consensus feedback block* is a $(m \cdot N) \times (m \cdot N)$ matrix that is created using the *Kronecker product* (denoted by $\otimes$), that is $\bar{\mathcal{L}} = \mathcal{L} \otimes I_m$, where $I_m$ is the $m \times m$ identity matrix. In particular, if we set $m = 2$ or $m = 3$, we perform consensus for a group of agents moving in a two-dimensional or three-dimensional space respectively. In the rest of this chapter, if no extra information is provided, we mean for simplicity of exposition $m = 2$.




(a) Evolution of the X coordinates

(b) Evolution of the Y coordinates

(c) $t = 0$ sec

(d) $t = 0.63$ sec

(e) $t = 1.63$ sec

(f) $t = 3.63$ sec

Figure 3.7: Evolution of system performing consensus

**Definition 3.15** (Kronecker product)**.** *: given a $m-$ by $-n$ matrix $A$ and a $p-$ by $-q$ matrix $B$, the Kronecker product $A \otimes B$ is $mp-$ by $-nq$ block matrix*

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix} \tag{3.17}$$

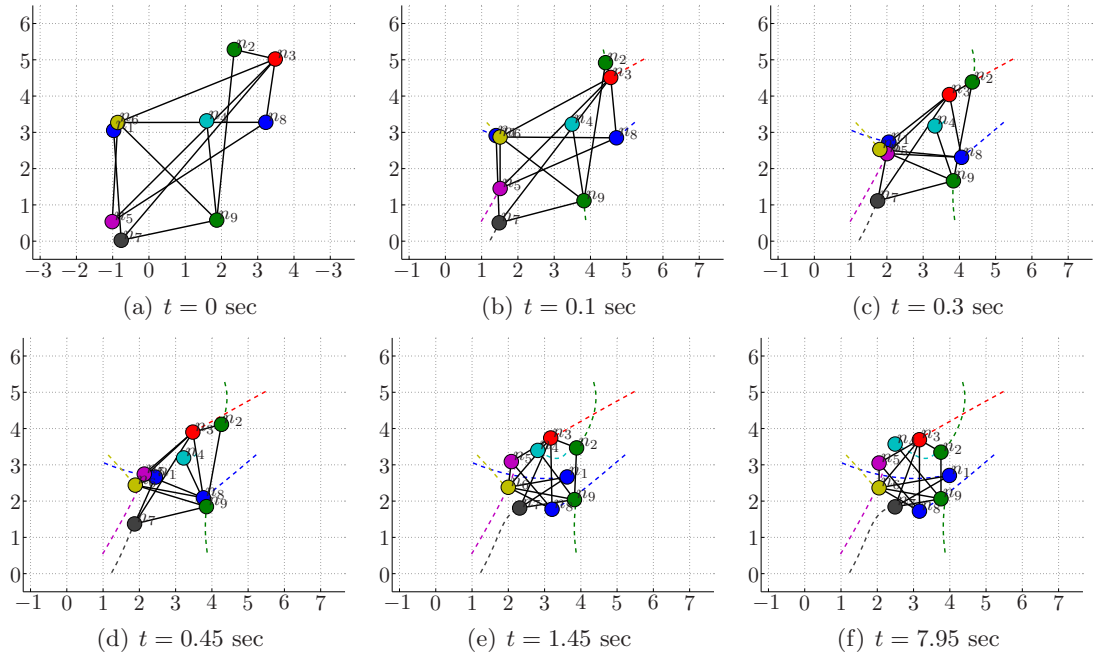
**Note 3.1.** *The Kronecker product enjoys the following properties:*

- $A \otimes (B + C) = A \otimes B + A \otimes C$
- $(A + B) \otimes C = A \otimes C + B \otimes C$
- $(kA) \otimes B = A \otimes (kB) = k(A \otimes B)$
- $(A \otimes B) \otimes C = A \otimes (B \otimes C)$

**Note 3.2.** *The Kronecker product is not commutative ($A \otimes B \neq B \otimes A$), even if two permutation matrices $P$, $Q$, can be found such that:*

- $A \otimes B = P(B \otimes A)Q$

An example of a group of agents moving in a two-dimensional space is depicted in Fig. 3.7, where their behavior is shown.



(a) $t = 0$ sec (b) $t = 0.1$ sec (c) $t = 0.3$ sec

(d) $t = 0.45$ sec (e) $t = 1.45$ sec (f) $t = 7.95$ sec

Figure 3.8: Evolution of system performing consensus

### 3.2.1 Formation control

There are two main paradigms to achieve formation for multi-vehicle systems: the first approach is based on the idea of using a rigid structure to represent the desired formation and to control the robots behavior basing on inter-vehicle potential fields; the second one, instead, is based on the idea of representing the group of vehicles as a graph (where communication links are represented by edges) performing consensus with an *input bias* other than zero. The second paradigm is analyzed in this section, focusing on the possibility of using the inter-agents distances to calculate the bias vector.

Suppose now to have the two vectors $x_d$ and $y_d$ with the desired final positions. If we define a new state $z = x - x_d$ (the same for the second dimension), the eq 3.14 becomes:

$$z_i(t) = x_i(t) - x_{i,d}, \ \forall i = 1 \dots N \tag{3.18}$$

Follows, for the single agent:

$$\dot{z}_i(t) = -\sum_{j \in N(i)} (z_i(t) - z_j(t)) = -\sum_{j \in N(i)} ((x_i(t) - x_j(t)) - (x_{i,d} - x_{j,d})) \tag{3.19}$$

If we define $r_{ij} = x_{i,d} - x_{j,d}$ as the desired relative distance between agent $n_i$ and agent $n_j$, $\forall n_j \in N(i)$), from Eq 3.13 it follows

$$b_i(t) = -\sum_{j \in \mathcal{N}(i)} (x_{i,d} - x_{j,d}) = -\sum_{j \in \mathcal{N}(i)} r_{ij} \tag{3.20}$$

that is the *bias input* required for agent $n_i$ to reach the desired position asymptotically. In Fig. 3.8 the behavior of a swarm of nine particles is depicted: starting from random positions, they converge in a regular polygon configuration.

## 3.3 Applications of the consensus algorithm

Due to its simplicity, the consensus algorithm explained in the previous section has been widely studied and applied to solve problems arisen in many different research fields. In particular, it has been studied for commercial and military applications.These problems can be summarily divided into two main branches, that are explained in this section.

### 3.3.1 Leader Networks

The key point behind the so called *leader networks* is the idea of finding some special nodes called *leaders* that can be used to drive all the group toward a predefined task [58] [89] [3] [92]. The ability to characterize a given network as a *leader network* is the starting point for the solution of two problems that are strictly related with the possibility of controlling only some agents of the group while the others perform the consensus algorithm.

Given a swarm of $N$ agents described by a graph $G = (V, E)$, suppose that in the vertex set there is a subset of $N_l$ agents that can be considered the leaders. In particular, to deal with the problems related to a leader network, we have to assume that:

- the leaders have access to global information;
- a static graph $G$ represents the network topology
- the graph $G$ is connected and undirected

In a leader-based network represented by a graph $G = (V, E)$, the vertex set can be divided in two disjoint (but connected) subsets, the *follower* subset $V_f$ and the *leader* subset $V_l$, such that $V_f \bigcup V_l = V$ and $V_f \bigcap V_l = 0$. Without loosing of generality, we can suppose that the first $N - N_l$ agents are follower while the last $N_l$ agents are leaders, i.e. $x_f = [x_1 \dots x_{N_f}]^T$, $x_L = [x_{N_f+1} \dots x_N]^T$, where $N_f = |V_f|$.

As a consequence of this division, the incidence matrix and the Laplacian matrix can be partitioned as:

$$\mathcal{I} = \left[\begin{array}{c} \mathcal{I}_f \\ \hline \mathcal{I}_l \end{array}\right] \Rightarrow \mathcal{L} = \left[\begin{array}{c|c} \mathcal{L}_f & \mathcal{L}_{fl} \\ \hline \mathcal{L}_{fl}^T & \mathcal{L}_l \end{array}\right] = \left[\begin{array}{c|c} \mathcal{I}_f\mathcal{I}_f^T & \mathcal{I}_f\mathcal{I}_l^T \\ \hline \mathcal{I}_l\mathcal{I}_f^T & \mathcal{I}_l\mathcal{I}_l^T \end{array}\right] \tag{3.21}$$

where $\mathcal{L}_f \in \mathbb{R}^{N_f \times N_f}$, $\mathcal{L}_l \in \mathbb{R}^{N_l \times N_l}$ and $\mathcal{L}_{fl} \in \mathbb{R}^{N_f \times N_l}$.

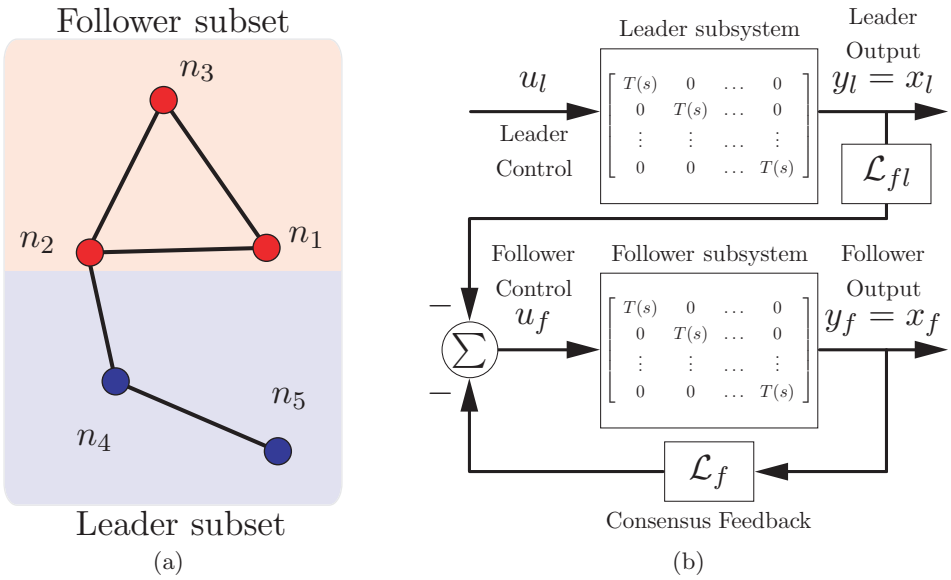



Figure 3.9: Example of a leader-follower network: graph (Fig. 3.9(a)) and control loop (Fig. 3.9(b))

.

If we suppose to be able to control the *leader* nodes, the dynamic of the controlled system can be expressed as:

$$\begin{bmatrix} \dot{x_f} \\ \dot{x_l} \end{bmatrix} = - \begin{bmatrix} \mathcal{L}_f & \mathcal{L}_{fl} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_f \\ x_l \end{bmatrix} + \begin{bmatrix} 0 \\ I \end{bmatrix} u \tag{3.22}$$

The system for the *followers* is

$$\dot{x}_f = -\mathcal{L}_f x_f - \mathcal{L}_{fl} x_l \tag{3.23}$$

As pointed out in [58], we have:

**Theorem 3.4.** *Given a connected graph $G = (V, E)$ and a subset $V_f \subset V$, the submatrix $\mathcal{L}_f$ is positive definite ($\mathcal{L}_f \succ 0$)*

**Theorem 3.5.** *Given a connected graph $G = (V, E)$ with the vertex set partitioned in* leaders *and* followers *(performing consensus in Eq 3.23), for a fixed leader position, the equilibrium point of the follower subsystem is*

$$x_f = -\mathcal{L}_f^{-1} \mathcal{L}_{fl} x_l \tag{3.24}$$

*that is globally asymptotically stable.*

An example of leader-follower graph and the corresponding control loop is reported in Fig. 3.9. In Fig. 3.9(a) leader nodes are blue and follower nodes are red.

Before going on with the control of leader-based networks, some conditions must be verified in order to ensure the controllability of the *follower* system.

**Theorem 3.6.** *If we call $A = -\mathcal{L}_f$ and $B = \mathcal{L}_{fl}$, the system $\dot{x} = Ax + Bu$ is controllable if and only if one of this propositions holds:*

1. $\text{rank}(C) = \text{rank}(A)$, *where $C$ is the controllability matrix $C = [B\ AB\ \dots\ A^{N_f}B]$;*
2. $\text{rank}(W_c) = \text{rank}(A)$, *where $W_c$ is the controllability Gramian;*
3. $\text{eig}(\mathcal{L}) \cap \text{eig}(\mathcal{L}_f) = 0$;
4. $\text{eig}(\mathcal{L}_f)$ *are distinct and* $eigv(\mathcal{L}_f) \cap B^{\perp} = 0$ *(see [92]).*

**Example 3.7.** *Let's consider the graph reported in Fig. 3.9(a). It is:*

$$\mathcal{L} = \left[\begin{array}{ccc|cc} 2 & -1 & 0 & -1 & 0 \\ -1 & 3 & -1 & -1 & 0 \\ 0 & -1 & 2 & 0 & -1 \\ \hline -1 & -1 & 0 & 2 & 0 \\ 0 & 0 & -1 & 0 & 1 \end{array}\right] \quad \text{with : } \text{eig}(\mathcal{L}) = \begin{bmatrix} 0.0 \\ 0.5188 \\ 2.3111 \\ 3.0 \\ 4.1701 \end{bmatrix}, \quad \text{eig}(\mathcal{L}_f) = \begin{bmatrix} 1.0 \\ 2.0 \\ 4.0 \end{bmatrix}.$$

*As the proposition 3 of theorem 3.6 is satisfied, the system is controllable.*

At the two extremity of the spectrum connectivity of connected graph, we have:

- the graph is complete but is not controllable;
- the graph is a *path* graph and is controllable.

This means that having a low connectivity (i.e. the algebraic connectivity $\lambda_1$ is low) is not always a bad characteristic for graphs. More generally, once a connected graph is given, one of the hardest problem is to find a partition of the vertex set in order to have a controllable system. In fact, given a connected graph, a random search for a controllable configuration can be very expansive in terms of time and resources. In Fig. 3.10 two examples are depicted: in Fig. 3.10(a) the probability of a node of being connected is 20%, in Fig. 3.10(b) the probability is 2%. It is possible to note that as long as the number of the graph vertices and the probability of a node of being a *leader* increase, the mean probability over 200 iteration of finding a controllable leader-follower configuration drastically increases.

It is easy to note that for high leader probability and for a high number of vertexes there is a peak in the number of iteration requested to find at least one controllable leader-follower configuration.

An important theorem that allows to find a subset of leader/follower agents in order to have controllable system is in [58]:

**Theorem 3.7.** *The system $(\mathcal{L}_f, \mathcal{L}_{fl})$ is controllable if $G$ is connected and* $\text{null}(\mathcal{I}_l) \subseteq \text{null}(\mathcal{I}_f)$, *where $\mathcal{I}_f$ and $\mathcal{I}_l$ are defined in Eq 3.21.*

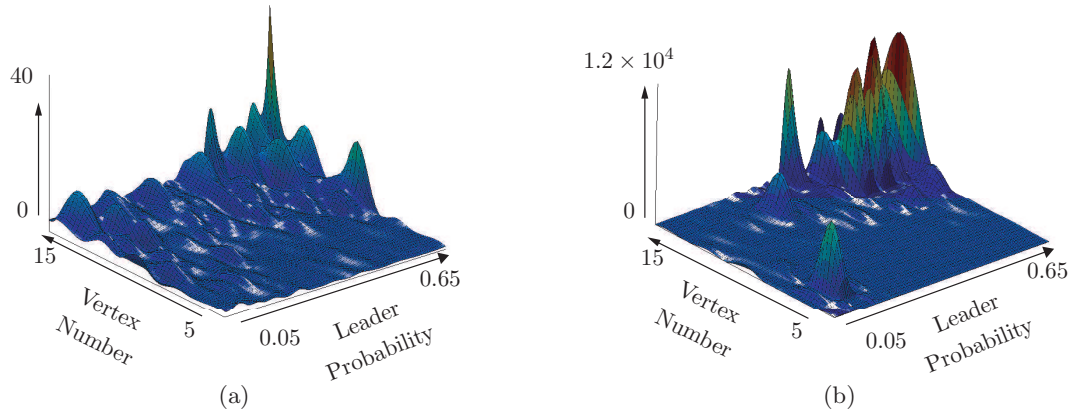



Figure 3.10: Probability of finding a controllable configuration with a node probability connection of 20% (3.10(a)) and 2% (3.10(b)).

Now that we have analyzed the main problems related to the use of a leaders subset to control the system, let's see two applications of these results.

### Optimal Control

The optimal control problem applied to a leader/follower graph can be formulated as follows: given a network of agents modeled as a connected graph $G = (V, E)$ and given a subset of nodes that acts as leaders such that the system in Eq 3.23 is controllable, find the control input for the leader set in order to drive the followers from a position $x_0$ at time $t_0$ to a final position $x_1$ at time $t_1$. This formulation is widely referenced as the *optimal point-to-point transfer* and the conditions $x(T_0) = x_0$ and $x(T_1) = x_1$ are called *boundary states*.

Let's consider the system in Eq 3.23 and, for the sake of notational convenience, equate $x_f$ with $x$, $x_l$ with $u$, $-\mathcal{L}_f$ with $A$ and $-\mathcal{L}_{fl}$ with $B$, so that it can be rewritten as

$$\dot{x} = Ax + Bu \tag{3.25}$$

As the theory regarding optimal control has been widely studied in literature, in this subsection we present only an application of the optimal control theory applied to a leader-follower network of agents.

The optimal control is solved if a given cost index can be minimized; in general the problem can be formulate as:

$$\min J \tag{3.26}$$

The cost index $J$ can be a function of the input and of the system state, or simply a

function of the input. The typical example used in optimal control theory is

$$J = \int_{t_0}^{t_d} u^T(t)u(t)dt \tag{3.27}$$

that minimize the control energy.

The solution of this problem is well-known in literature [53] [19], and it is:

$$u(t) = B^T e^{A^T(t_1-t_0)} W_c(t_0,t_1)^{-1} \left(x_d - e^{A(t_1-t_0)}x_0\right) \tag{3.28}$$

where $W_c(t_0,t_1)$ is the *controllability Gramian* defined as:

$$W_c(t_0,t_1) = \int_{t_0}^{t_1} e^{sA} B B^T e^{sA^T} ds \tag{3.29}$$

Recall that, if the system in Eq 3.25 is controllable, the controllability Gramian is always invertible. As an example, consider a network of five agents represented as a path graph, where nodes $n_1 \dots n_4$ are the follower and node $n_5$ is the leader. The boundary states are

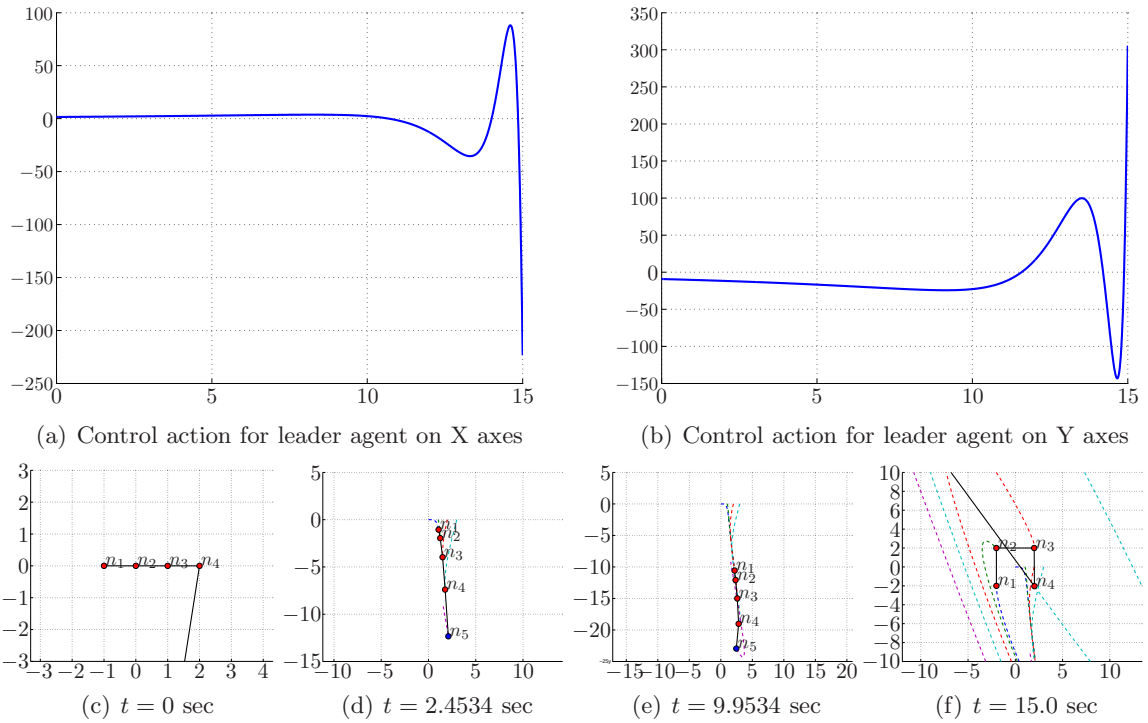
$$\begin{cases} x(0) = x_0 = [0,\ 1,\ 2,\ 3]^T \\ y(0) = y_0 = [0,\ 0,\ 0,\ 0]^T \end{cases} \quad \text{and} \quad \begin{cases} x(15) = x_1 = [-2,\ -2,\ 2,\ 2]^T \\ y(15) = y_1 = [-2,\ 2,\ 2,\ -2]^T \end{cases}$$

i.e. starting from a line configuration we want to drive the followers at time $t = 15$ sec. to a square configuration with side 4.

In Fig. 3.11 the control signal is reported with some snapshots of the simulation. It is important to recall that even if the optimal control of a group of agents allows to drive the agents between two boundary states in a finite time by acting with an exogenous control action only on the *leaders*, it may present some feasibility problems: first of all the control action can saturate the actuators (i.e. the energy supplied to the system is too high); secondly there can be numerical problems. In fact, as the Gramian is calculated by integrating a matrix multiply, the final result can be a matrix $W_c(t_0,t_1)$ that is bad conditioned.

### Containment Algorithm

The second application that concerns the use of the leader-follower paradigm is the so called *containment problem*. In this problem we have the followers subset performing the consensus and a the leader subset controlled by an exogenous signal. The leaders, in particular, have to move from a zone of the plane (in 2D space) to another predefined zone keeping all the followers inside the convex polytope spanned by them (a polygon in 2D and a polyhedron in 3D). This problem is frequently indexed as *herding problem*, because it is the typical problem that shepherds have to face when they have to keep a herd of animals compact while moving from a pasture to another. To face the solution of the herding problem we have to make some starting assumptions:



(a) Control action for leader agent on X axes

(b) Control action for leader agent on Y axes

(c) $t = 0$ sec

(d) $t = 2.4534$ sec

(e) $t = 9.9534$ sec

(f) $t = 15.0$ sec

Figure 3.11: Evolution of system with optimal control

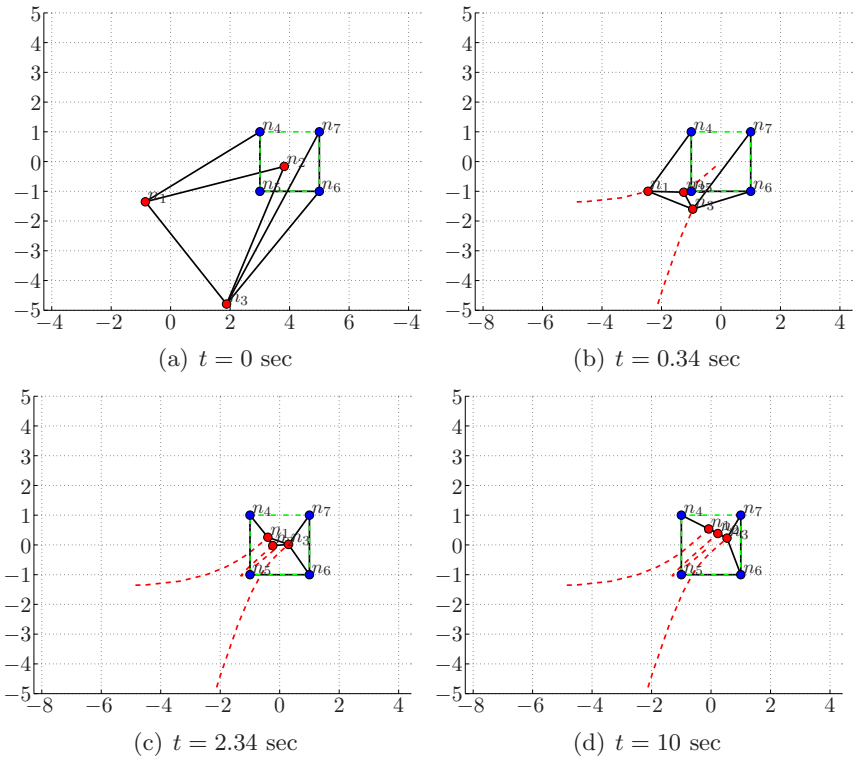
- the graph $G$ is fixed and connected;
- the *leader subgraph* is connected;
- the *followers* execute the consensus algorithm;
- the *leader velocities* are unbounded;
- leaders are able to detect polytope-perimeter crossing conditions.

The polytope spanned by one or two leaders is a point or a segment respectively and both cases are considered as degenerate polytopes. In this section we suppose that in our leader-network there are at least three leader agents.

The main work about this problem is addressed in [33], where two important theorems are enunciated and demonstrated.

**Theorem 3.8** (Follower convergence)**.** *Given a graph $G = (V, E)$, with $V$ partitioned in leaders and follower subsets such that the previous assumptions hold, the follower agents converge to the convex polytope spanned by leader agents.*

**Theorem 3.9.** *Given a graph $G = (V, E)$, with $V$ partitioned in leaders and follower subsets, once the followers are inside the polytope spanned by the leaders, they never point away.*



(a) $t = 0$ sec (b) $t = 0.34$ sec

(c) $t = 2.34$ sec (d) $t = 10$ sec

Figure 3.12: Four static leaders *attract* followers.

An example of graph that can perform the containment algorithm described in this section is shown in Fig. 3.12, where the follower nodes (red) converge to the center of the polygon spanned by the leaders (blue). Some problems arise when the leaders move to execute a point-to-point transfer of the followers. The following corollary point out the first problem related to the convergence time of the followers:

**Corollary 3.10.** *Given a graph $G = (V, E)$, with $V$ partitioned in leaders and follower subsets such that the previous assumptions hold, if a follower is connected only with one or two leaders, it will asymptotically converge to a vertex or to a side of the convex polytope spanned by leader subset.*

This particular configuration can drive the system into a deadlock situation, where the follower converges into the polytope only asymptotically. In Fig. 3.13 an example of these two possible deadlock configurations are shown. These problems, addressed as *static problem* because arises when the leaders are static and are waiting for followers to converge, can be circumvented by transmitting to the followers *false information* about leaders position: in other words, the leaders see a new convex polytope that is *smaller* than the real one. With this trick, the followers with critical connections will tend asymptotically to the border of the inner polytope and will go into the original polytope spanned by leaders in a finite time. Now that we are sure that all the followers converge in a finite time

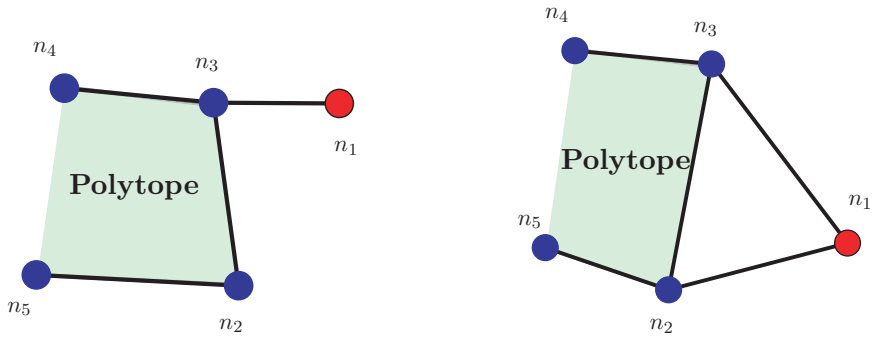
into the polytope spanned by the leaders, we can assume that the leaders start moving to transfer the followers from the starting zone to a final one. At this point, another problem arises: as the leaders velocities are unbounded, it can happen that the leaders are faster than the followers, thus one or more follower agents can lie out of the polytope spanned by leaders. To solve this problem, a hybrid control has been used to move the leaders and at the same time ensure that all the followers never lie out of the polytope. The hybrid control automaton is described in Fig. 3.14 with snapshots that explain the meaning of the two states. The *guard* conditions are defined as:

$Guard_1$ (**STOP2go**): all the followers are inside the polytope spanned by leaders;

$Guard_2$ (**GO2stop**): at least one follower is outside the polytope spanned by leaders;

The hybrid control used to move leaders is useful, but it can be improved if the crossing condition is applied not on the real polytope but on the *false* polytope used to avoid deadlock situations: in this case, in fact, the leaders will stop before one follower fall out of the leader-spanned polytope, ensuring that once the stating condition is true, they never go out of the polytope, even if their dynamic is slower than the leaders one. This trick can be very important if, for example, the leaders are escorting vehicles that transport dangerous materials (the polytope can be interpreted as the *safety zone*). The problem with the automaton in Fig. 3.14 is that the system, in some particular configuration or if the leaders are faster than the followers, can switch frequently between the two states, thus slowing down the system performances. The last trick that one can use is to change dynamically the dimension of the inner polytope, such that it expands while leaders are in *GO* mode and reduces in *STOP* mode.

Let's conclude about the containment algorithm with an example where the hybrid algorithm described in this section has been applied using all the improvements (inner polytope, varying radius). The law used for varying inner polytope radius is a saturated




(a) (b)

Figure 3.13: Critical configurations in herding problem.

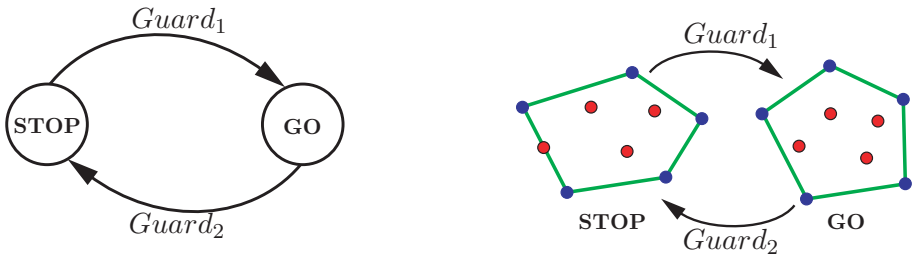



Figure 3.14: Automaton for the Stop-and-Go hybrid control for herding.

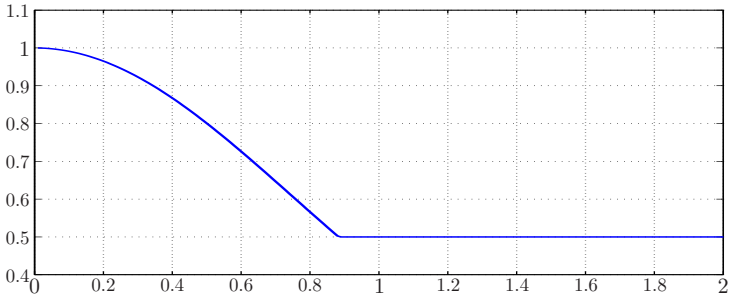



Figure 3.15: Safety polytope parameterized varying radius

Gaussian-like function:

$$r(d) = \max\{0.5\ ,\ \frac{1}{M\rho\sqrt{2\pi}} e^{-\frac{(d-\mu)^2}{2\rho^2}}\} \tag{3.30}$$

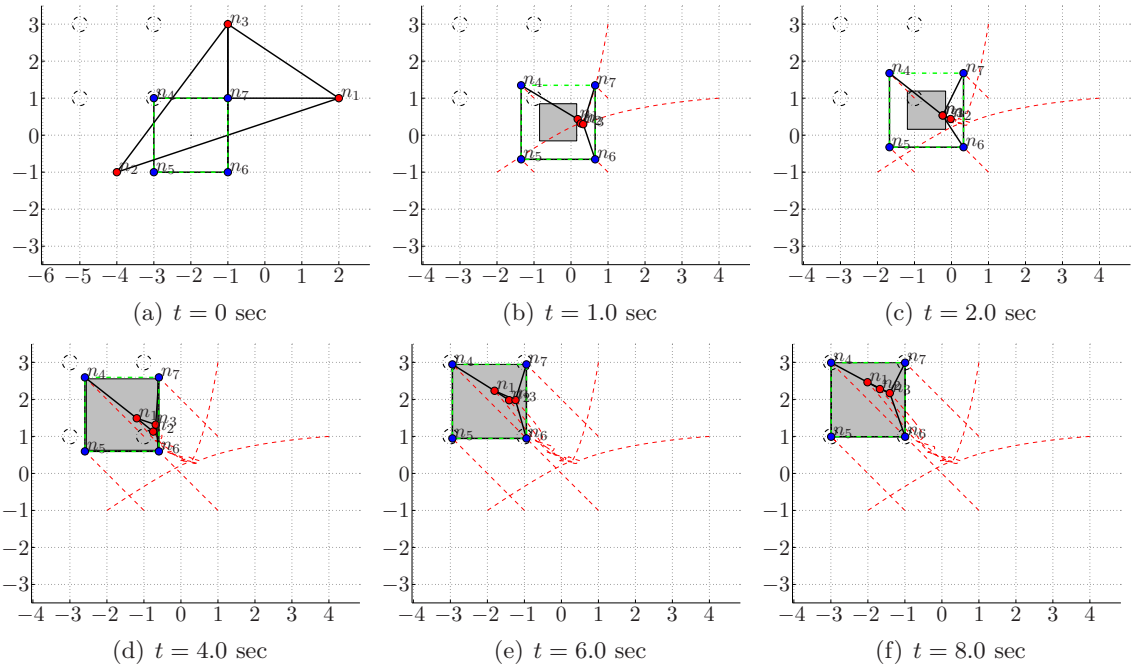
where $M = \frac{1}{M\rho\sqrt{2\pi}} e^{-\frac{\mu^2}{2\rho^2}}$ normalize $r(d)$ such that $0 \leq r(d) \leq 1$ and $d$ is a weighted function of leaders position errors (mean the distance to the target position) and the leader velocity. In Fig. 3.15 an example of the eq 3.30 is depicted.

The graph depicted in Fig. 3.16 is a seven-agents network where three follower agents ($V_f = \{n_1, n_2, n_3\}$) start in random positions and four leader agents ($V_l = \{n_4, n_5, n_6, n_7\}$) start in a square formation. The goal is to move leaders to the positions marked with dotted circles while herding followers. The safety polytope is in grey.

In Fig. 3.17 the leaders control action is compared with the solution of the same example without using safety polytope. It is easy to see that using a safety polytope the system moves slowly toward the final positions but, on the other hand, the hybrid control avoids jitter.

### 3.3.2 Δ-disk Graphs

So far we have considered the networks as a *static networks*, i.e. the starting communication graphs do not change in time. If we introduce the possibility of having networks with



(a) $t = 0$ sec (b) $t = 1.0$ sec (c) $t = 2.0$ sec

(d) $t = 4.0$ sec (e) $t = 6.0$ sec (f) $t = 8.0$ sec

Figure 3.16: Snapshots of a simulation of a group of agents performing herding task.

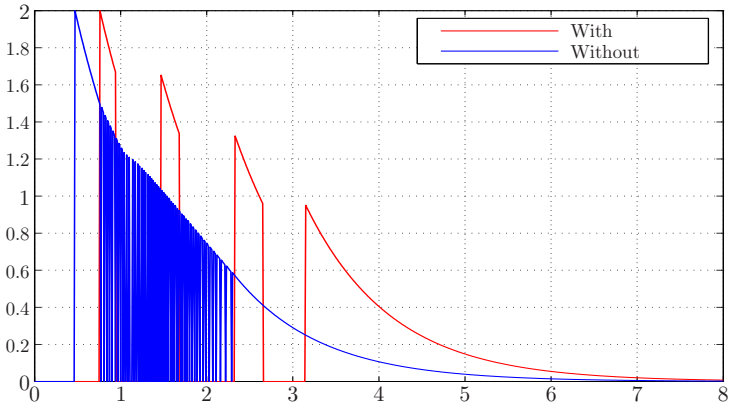



Figure 3.17: Comparison between leaders control action with (red) or without (blue) the safety polytope.

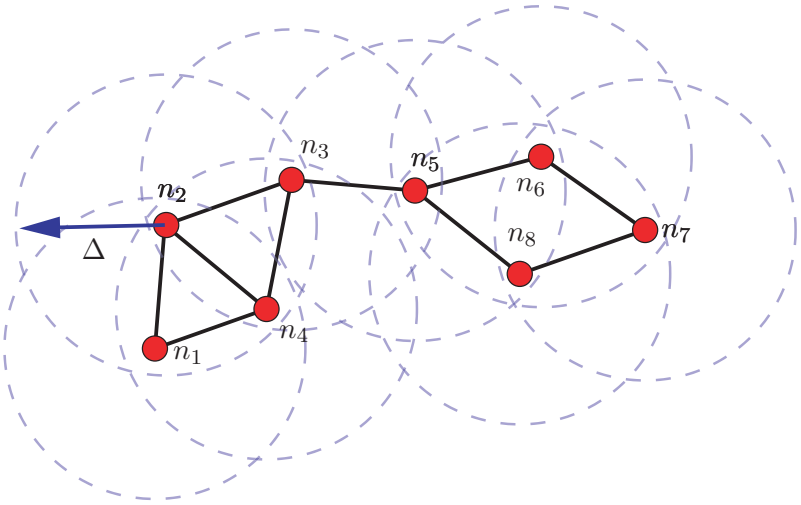



Figure 3.18: Example of a $\Delta$ disk graph. The edge $e_{3,5} = (n_3, n_5)$ is the critical connection of the swarm.

connections that can change in time, we deal with *dynamic networks*. This improvement of the classical network model can be considered as a step toward the reality. In fact, in reality agents have to deal with adverse environmental conditions (like obstacles) and we have to take into account the possibility for an agent to lose the connection with the others. A typical example of a graph model used to study dynamic networks is the so called $\Delta$-disk graph. A $\Delta$-disk graph $G_\Delta(V, E)$ is defined by a couple of sets as in Def 3.1, where the *edge set* is slightly different.

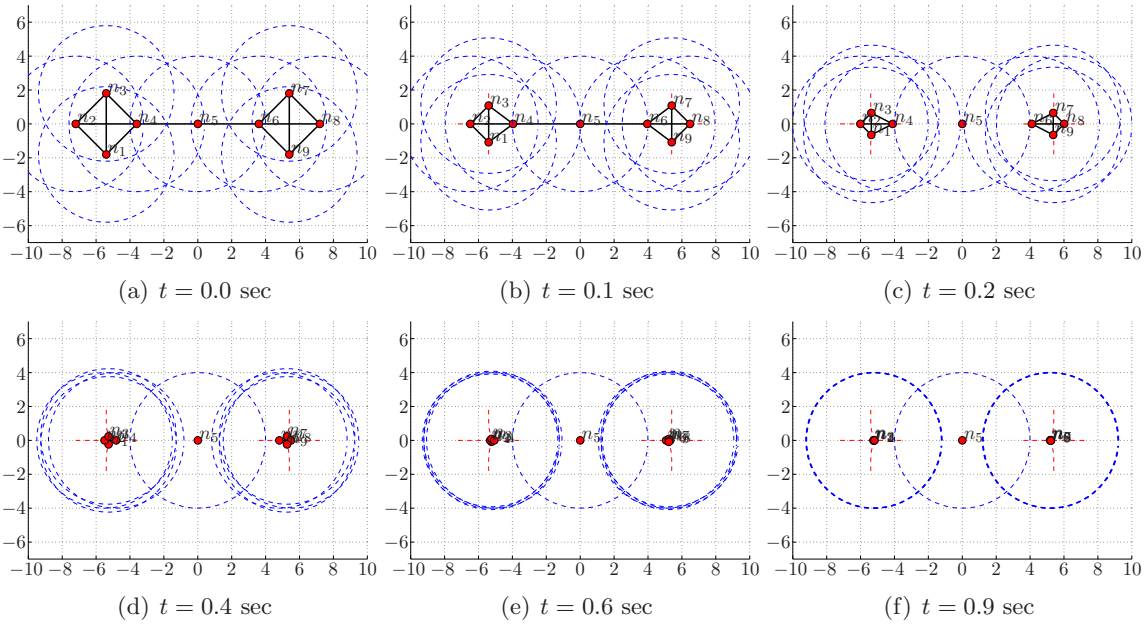
**Definition 3.16.** *$\Delta$ edge set Given a graph $G$ with $N$ nodes, the $\Delta$ edge set is defined as:*

$$E_\Delta \subseteq V \times V : \; E_\Delta = \{(n_i, n_j) : d(n_i, n_j) < \Delta\} \tag{3.31}$$

*where $d(n_i, n_j)$ is the Euclidean distance between two agents and $\Delta$ is the communication distance. Specifying for the 2D problem, it is $d(n_i, n_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.*

In case of an heterogenous set of nodes is used in the same network, it is possible to define $E_\Delta$ using different communication range. For simplicity, we will consider that all the agents have the same communication range. Note that having a edge set dynamically defined imply that all the matrices previously defined to describe the graph properties (Adjacency matrix, Degree matrix, Incidence matrix and Laplacian matrix) are no longer static but change dynamically. In Fig. 3.18 an example of a $\Delta$-disk graph is reported. As can be pointed out from the figure, there is a critical connection between nodes $n_3$ and $n_5$ as, if the edge $e_{3,5}$ goes down, the graph is no more connected.

One of the most interesting problem related to $\Delta$-disk graph is to find a way to control the swarm in order to preserve connectedness while performing consensus algorithm or



(a) $t = 0.0$ sec (b) $t = 0.1$ sec (c) $t = 0.2$ sec

(d) $t = 0.4$ sec (e) $t = 0.6$ sec (f) $t = 0.9$ sec

Figure 3.19: Nine agents $\Delta$-disk swarm performing consensus.

formation keeping [57] [56] [55] [40]. Indeed, while the agents move, they can go too far from a subset of neighbors and then broke the network connectivity. An example is reported in Fig. 3.19, where some snapshots of a $\Delta$-swarm performing consensus are depicted, where it is $\Delta = 4$. Note that at time $t = 0.1$ sec. the node $n_5$ is at the limit of the communication range of neighbors $n_4$ and $n_6$. After this time instant, the connection is lost and the graph is no longer connected. The phenomena can be easily explained. Let's consider what happens for agent $n_4$: as long as the swarm is performing consensus, $n_4$ is "attracted" to two opposite neighbors subsets $N_A = \{n_1,\ n_2,\ n_3\}$ $N_B = \{n_5\}$. As the cardinality of subset $N_A$ is grater than the cardinality of $N_B$, the agent $n_4$ is strongly attracted to the first subset.

The elemental solution for this problem is based on the use of a *edge-weighted graph* as defined in point 1 of Def 3.9. In particular, our weight matrix $\mathcal{W} \in \mathbb{R}^{|E|\times|E|}$ is a diagonal matrix that can be used to change dynamically the Laplacian matrix $\mathcal{L}$. In fact, it is:

$$\mathcal{L}_{\mathcal{W}} = \mathcal{I} \cdot \mathcal{W} \cdot \mathcal{I}^T \quad \text{where } \mathcal{W} = \text{diag}(w_k) : e_k \in E \tag{3.32}$$

and in particular the single agent dynamic in Eq 3.13 becomes:

$$\dot{x}_i(t) = -\sum_{j\in N(i)} w_{i,j}(x_i(t) - x_j(t)) + b_i(t), \text{ where } \begin{cases} x_i(0) &= x_{i,0} \in \mathbb{R} \\ b_i(t) &= 0 \end{cases} \tag{3.33}$$

where $w_{i,j}$ is the weight of the edge (if exists) connecting nodes $n_i$ and $n_j$. Till now, we

have always considered

$$w_{i,j} = \begin{cases} 1 & \text{if } (n_i, n_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

The $\mathcal{W}$ matrix is defined as a diagonal matrix where each elements correspond to the weight of an edge. As changing the elements of $\mathcal{W}$ it is possible to change the behavior of the swarm, our goal is to find a function to weight edges such that as long as the starting graph is connected, the swarm performing consensus will never becomes disconnected.

In [56] an edge tension $V_{ij}(\Delta, x)$ is defined between each couple of nodes as:

$$V_{ij}(\Delta, x) = \begin{cases} \dfrac{\|l_{ij}(x)\|^2}{\Delta - \|l_{ij}(x)\|}, & \text{if } (n_i, n_j) \in E \\ \\ 0, & \text{otherwise} \end{cases} \tag{3.34}$$

It follows:

$$\frac{\partial V_{ij}(\Delta, x)}{\partial x_i} = \begin{cases} \dfrac{2\Delta - \|l_{ij}(x)\|}{(\Delta - \|l_{ij}(x)\|)^2}, & \text{if } (n_i, n_j) \in E \\ \\ 0, & \text{otherwise} \end{cases} \tag{3.35}$$

where $l_{ij} = x_i(t) - x_j(t)$ is the edge-vector between agents $n_i$ and $n_j$. The control law for weighted-consensus in Eq 3.33 becomes:

$$\dot{x}_i(t) = -\sum_{j \in N(i)} \frac{2\Delta - \|l_{ij}(x)\|}{(\Delta - \|l_{ij}(x)\|)^2}(x_i(t) - x_j(t)) \tag{3.36}$$

The weight matrix is:

$$\begin{array}{rcl} \mathcal{W}(\Delta, x) & = & \text{diag}(w_k(\Delta, x)), \quad k = 1, 2, \ldots, |E| \\ & \Updownarrow & \\ w_k(\Delta, x) & = & \dfrac{2\Delta - \|l_{ij}(x)\|}{(\Delta - \|l_{ij}(x)\|)^2} \end{array} \tag{3.37}$$

In the same work, authors clearly demonstrate that as long as the starting graph is connected, with the control law in Eq 3.36 any connection will be lost. A problem arises when a new connection is created: as long as the starting graph is connected but not complete, while the consensus algorithm is executed, new connections are created (i.e. two nodes that are disconnected at time $t = 0$ can become connected in finite time) because the distance between a couple of nodes became equal to $\Delta$. But with $\|l_{ij}(x)\| = \Delta$ both $V_{ij}(\Delta, x)$ and $\frac{\partial V_{ij}(\Delta, x)}{\partial x_i}$ go to infinity, and an infinite energy is introduce into the system by the control law. To bypass this problem, a hybrid control can be introduced using a two-states automaton as reported in Fig. 3.20, where $\epsilon > 0$ is a predefined *switching threshold*.

So, if we define a *state set* $\mathbb{D}^{\epsilon}_{\mathcal{G},\Delta} = \left\{x \in \mathbb{R}^{m \times N} : \|x_i - x_j\| \leq (\Delta - \epsilon)\right\}$, formally it is demonstrated that:

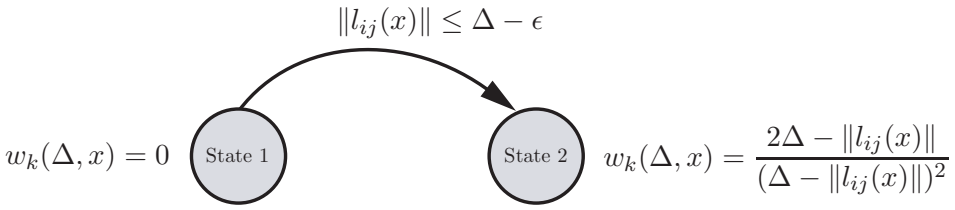



Figure 3.20: Two states automaton for preserving connectivity in a $\Delta$-disk graph.

**Theorem 3.11.** *Given a connected graph $\mathcal{G}$ with initial condition $x_0 \in \mathbb{D}^{\epsilon}_{\mathcal{G},\Delta}$ for a given $\epsilon > 0$, the dynamic system performing* consensus algorithm *converges to the static centroid.*

Considering two nodes $n_i$ and $n_j$, the behavior of the new controller can be described as follow:

- if the two nodes are disconnected or if the distance between them is greater then $(\Delta - \epsilon)$, then $w_{ij}(\Delta, x) = 0$;
- otherwise the value of $w_{ij}(\Delta, x)$ is defined in Eq 3.37.

Using a predefined threshold greater than zero, for each new connection the control will inject in the system a quantity of energy that is always lower than infinity, but is higher as long as the value of $\epsilon$ decrease.

In Fig. 3.21 some snapshots of the same simulation setup depicted in Fig. 3.19 are shown. Let's note the the connectivity is preserved. As the global control energy of the system is defined by

$$V = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} V_{ij}(\Delta, x), \tag{3.38}$$

in Fig. 3.22 the peaks correspond to the creation of a new connection and the value converges to zero asymptotically.

### $\Delta$-graph and formation control

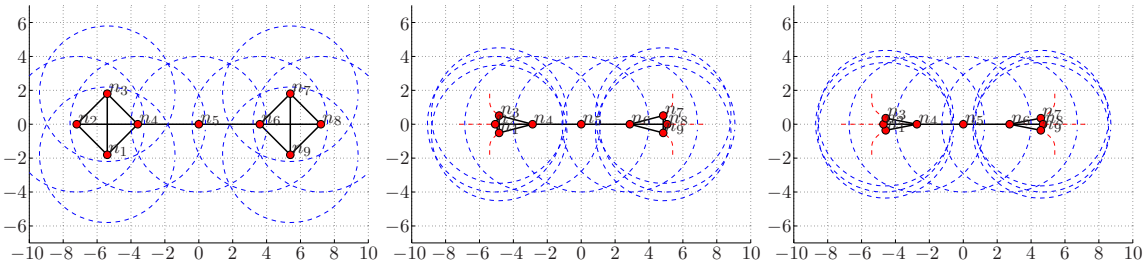
Now that the problem of maintain connectivity in a $\Delta$-disk graph performing consensus is solved, it is useful to see how this solution can be applied to the formation control. As explained in Sec 3.2.1, a connected swarm achieves formation by using an exogenous signal (*bias*) as a reference. Its expression is defined in Eq 3.20. If we suppose that all the desired inter-agent distances $r_{ij}$ are known, the weights defined in Eq 3.37 can be modified as follows:

$$w_{ij}(\Delta - r_{ij}, x) = \begin{cases} \dfrac{2(\Delta - r_{ij}) - \|l_{ij}(x)\|}{(\Delta - r_{ij} - \|l_{ij}(x)\|)^2} & \text{if } (n_i, n_j) \in E_d \\ \\ 0 & \text{otherwise} \end{cases} \tag{3.39}$$

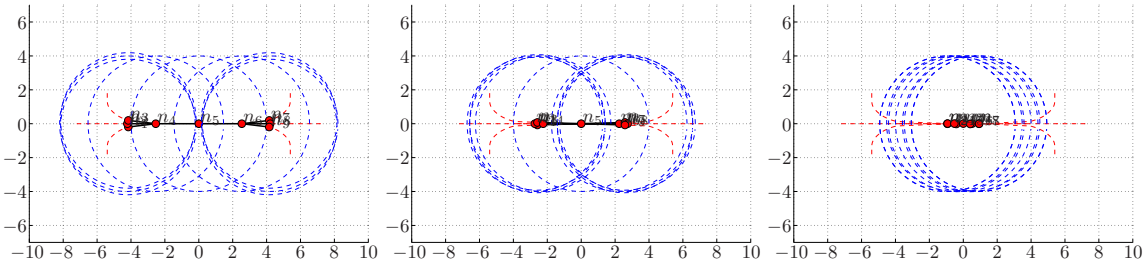
where $E_d$ is the *desired edge set*. In fact, as usually the $\Delta$-graph is not complete in the beginning, the new weights $w_{ij}(\Delta - r_{ij}, x)$ can not be applied till the system recognizes to be complete. This means that the automaton shown in Fig. 3.20 must be modified. It becomes as reported in Fig. 3.23.

Some snapshots where the weight matrix is defined using Eq 3.39 are reported in Fig. 3.24. Here, nine agents starting from random positions move to a regular nonagon formation.

As a final remark, it is important to notice that the control law applied to preserve connectivity presented in this section, can be applied also to all the other control problems that exploit consensus algorithm to control swarms of agents. It is particularly adapted to manage the herding problem as it can prevent leaders and followers going too far from each other.



(a) $t = 0.0$ sec (b) $t = 0.05$ sec (c) $t = 0.20$ sec



(d) $t = 0.30$ sec (e) $t = 3.45$ sec (f) $t = 0.90$ sec

Figure 3.21: Nine agents $\Delta$-disk swarm performing consensus using the weighted matrix defined in Eq 3.37, with $\Delta = 4$ and $\epsilon = 0.2$.

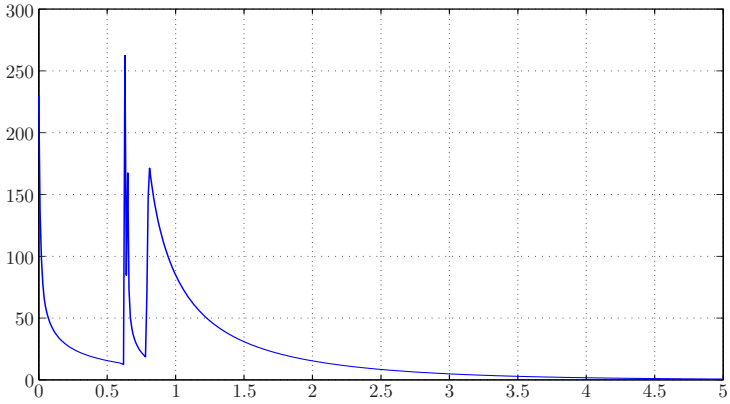


Figure 3.22: Global control energy. Peaks correspond to new connections.

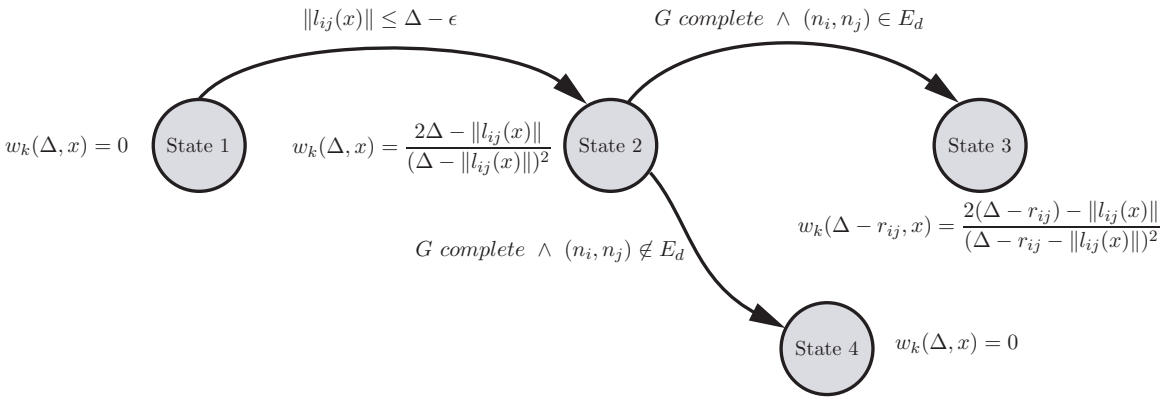
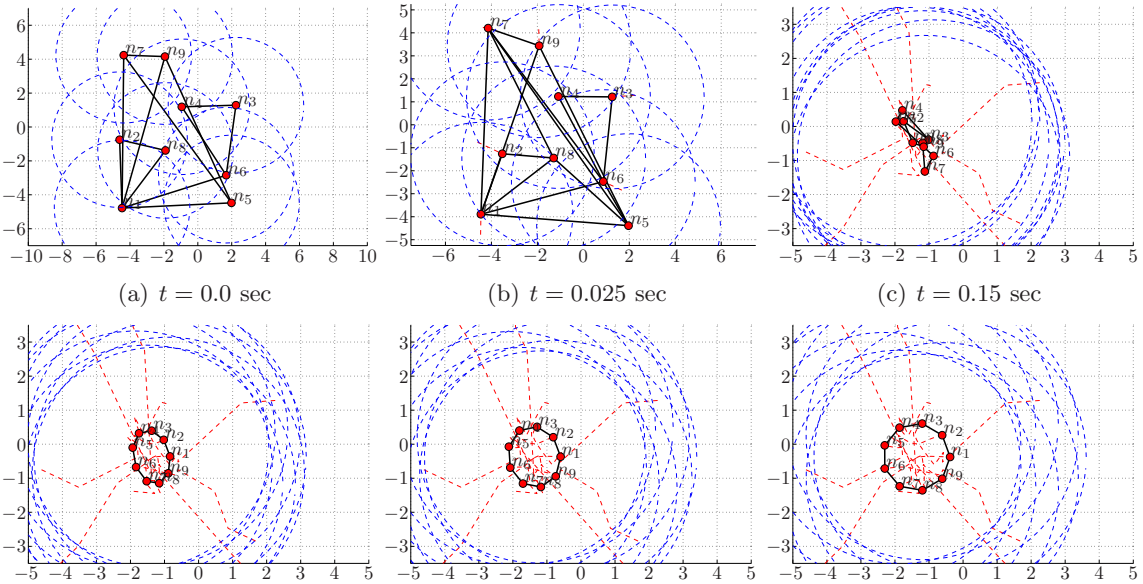



Figure 3.23: Automaton for preserving connectivity in a $\Delta$-disk graph engaged in a formation keeping mission.



(a) $t = 0.0$ sec (b) $t = 0.025$ sec (c) $t = 0.15$ sec

(d) $t = 0.975$ sec (e) $t = 1.275$ sec (f) $t = 5$ sec

Figure 3.24: Nine agents $\Delta$-disk swarm achieving regular nonagon with radius 2 using the weighted matrix defined in Eq 3.39, with $\Delta = 4$ and $\epsilon = 0.2$.

# Chapter 4

# A Graph-Based Distributed Control for Non-Holonomic Vehicles

In Chapter 3, basic notions on graph theory have been introduced in order to solve the *consensus problem* for a system of agents modeled as $n$-dimensional integrator, where a graph is used to represent the communication topology. Then, the attention has been focused on the possibility of using the *Laplacian feedback* solution of the consensus problem in order to solve a wide spectrum of problems, including formation keeping. In this chapter, we will show how the graph theory can be applied on agents modeled as real robots, focusing on differential-wheeled robots.

## 4.1 From holonomic agents to real robots

After introducing some basic definitions and notions about graph theory, in the previous chapter we have focus our attention on the so called *consensus problem* that, in simple terms, can be formulated as follows:

*"Given a system, find a control law to drive the states of the system to a final common value."*

For a system where the agents are modeled as single integrators (i.e. $n$-dimensional points without mass), the problem has a well known solution called *Laplacian feedback* (see Sec 3.2), that ensures the convergence of the state vector to a final common value.

If we consider our agents as points moving in a $n$-dimensional space, the solution can be interpreted as the ability of the system to converge to a final common point. A particular property of such systems is that the final point coincides with the centroid of the group and is static during the evolution of the system.

Our idea now is to extend this approach to a system where the particles are *real* robots: the problem that arises from the application of this theory to a group of real robot is that usually they can not be modeled as points (with or without mass). Even if in literature many works deal with the control of groups of omnidirectional robots, most of them are developed exclusively for research purposes. We decide to focus our attention on robots modeled as differential-wheeled robots due to the fact that this type of vehicles represent the better trade off between robot complexity (i.e. cost of the robot and of the maintenance) and the kinematic abilities of the vehicles. In fact, a differential wheeled robot is able to move straight, to bend and, most important, is able to turn on the spot, reducing the room needed for maneuvering. This characteristic is very important as allows the vehicle to accesses to areas forbidden to the others. Moreover, as seen in Chap 1, there are many vehicles modeled as differential wheel robots that are able to accomplish outdoor *real* missions, like de-mining, map building, planetary exploration or industrial surface cleaning.

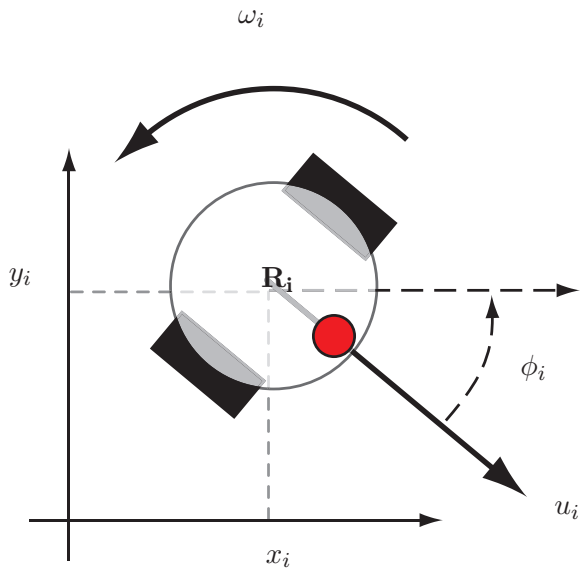



Figure 4.1: Differential wheel robot model.

The kinematic model of a differential-wheeled robot is shown in Fig 4.1, where the red dot over the body of the robot represents the front side. The kinematic equations of the

model are reported in Eq 4.1

$$\begin{cases} \dot{x}_i &= u_i \cos(\phi_i) \\ \dot{y}_i &= u_i \sin(\phi_i) \\ \dot{\phi}_i &= \omega_i \end{cases} \tag{4.1}$$

where $u_i$ is the linear speed, $\omega_i$ the rotational speed and $x_i$, $y_i$ and $\phi_i$ form the triplet defining the absolute coordinates.

The problem of state agreement for group of vehicles has been already faced in many works that can be find in literature [77] [95] [96] and, in particular, in [30] [12] the authors' attention is focused on the rendezvous problem for non holonomic vehicles modeled as differential-wheeled robots. In [44] authors deal with the problem of controlling formation and in [27] they consider also the possibility of having varying communication links.

In this Chapter we will deal with the problem of driving the position vectors $[x_i,\ y_i]$ to a final common value, without considering the orientation of the robots $\phi_i$. Part of the work exposed in this chapter was presented in [81].

## 4.2 Control Algorithm

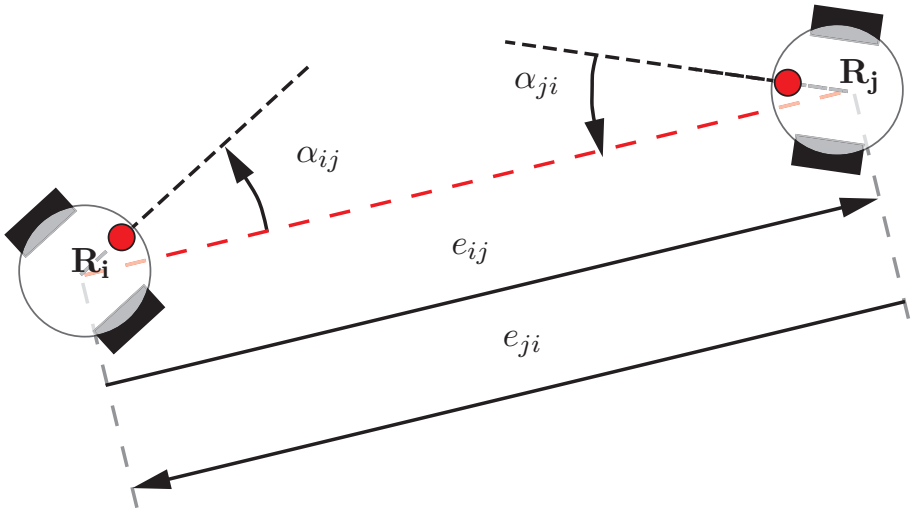



Figure 4.2: Example of local sensing.

The innovation introduced in this work is the fact that our graph-based control algorithm is based only on local sensing, means that we have supposed that each robot is able to detect the distance and the relative angle of the other members of the swarm within a given range. This introduces another limitation: a robot can not access to the global position and orientation data of its neighbors. Furthermore, occlusions between robots

can occur and, as explained in Sec 4.4, the system can became instable. The relative angles are positive defined in counter-clockwise: $\alpha_{ij} \in [-\pi, \pi]$. As an example, consider the system depicted in Fig. 4.2: $\alpha_{ij}$ is the relative angle where $R_i$ sees $R_j$ with respect to its front direction and $e_{ij}$ is the distance where robot $R_i$ senses robot $R_j$.

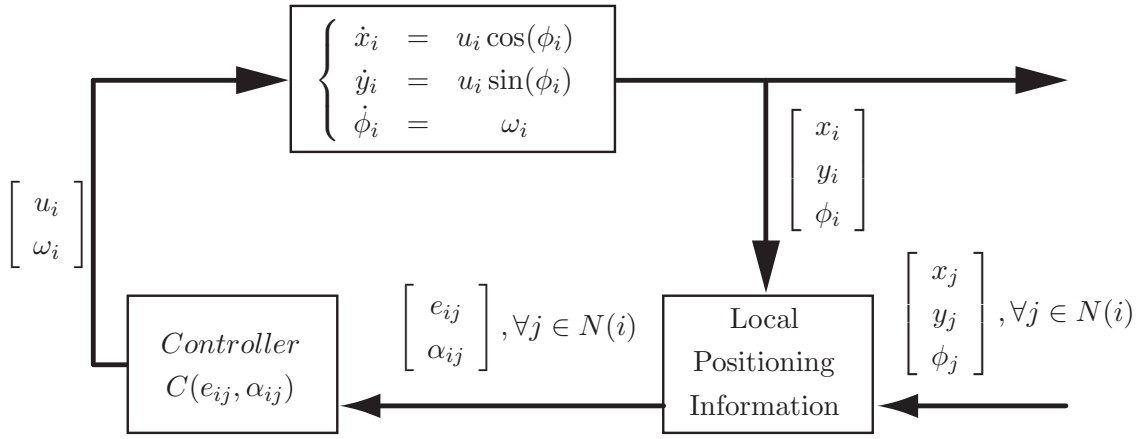



Figure 4.3: Feedback scheme for the consensus algorithm for a group of differential-wheeled robots using local sensing.

As long as we use vehicles with non-holonomic constraints, the feedback scheme for the $i$-th robot is reported in Fig. 4.3. Our goal is to create a controller $C(e_{ij}, \alpha_{ij})$ that is able to drive all the robots to a final common point exploiting only local sensing information. As we have seen in Section 3.2, in a system with agents modeled as massless particles, the convergence point is the centroid of the group. So, it sounds natural to choose this point as the convergence point for a group of differential-wheeled robots, in order to preserve a first match with the graph theory. As defined in Chapter 3, the set $N(i)$ represents the *neighbor set* of the $i$-th robot and $deg(n_i) = |N(i)|$ its cardinality.

With reference to the Fig. 4.4, we introduce now the following

**Definition 4.1** (Global relative error)**.** *For a robot* $\mathbf{R_i}$ *with* $|N(i)| \neq 0$*, we define:*

$$\begin{aligned} \bar{e}_{x,i} &= \frac{1}{|N(i)|+1} \sum_{j=1}^{|N(i)|} [-\mathcal{L}_{i,j} \cdot e_{i,j} \cdot \cos(\alpha_{i,j})] \\ \bar{e}_{y,i} &= \frac{1}{|N(i)|+1} \sum_{j=1}^{|N(i)|} [-\mathcal{L}_{i,j} \cdot e_{i,j} \cdot \sin(\alpha_{i,j})] \end{aligned} \tag{4.2}$$

*where* $e_{i,j}$ *is the Euclidean distance between* $\mathbf{R_i}$ *and* $\mathbf{R_j}$*, and* $\alpha_{i,j}$ *is the azimuth of* $\mathbf{R_j}$ *with respect to* $\mathbf{R_i}$*.* $\mathcal{L}$ *is the* Laplacian matrix *used to represent the connectivity topology of the swarm. The* global relative error *is defined as* $c_i = [\bar{e}_i,\ \bar{\alpha}_i]^T$*, where:*

$$\bar{e}_i = \sqrt{\bar{e}_{x,i}^2 + \bar{e}_{y,i}^2} \tag{4.3}$$

$$\bar{\alpha}_i = atan2(\bar{e}_{y,i}, \bar{e}_{x,i}) \quad \Rightarrow \quad \bar{\alpha}_i \in [-\pi, \pi] \tag{4.4}$$

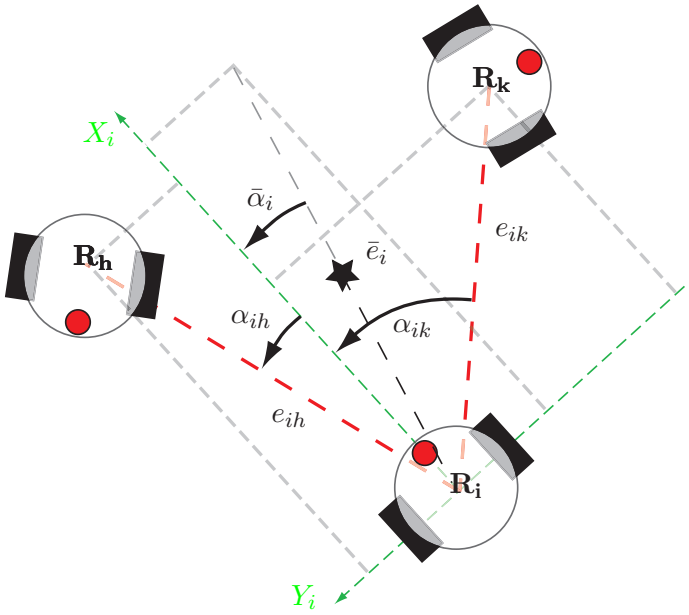



Figure 4.4: Definition of the relative position of the centroid of the group (black star) with respect to the robot $\mathbf{R_i}$.

In this section we assume that all the terms $\mathcal{L}_{i,j}$ are equal to 1 if robot $\mathbf{R_j}$ is in $N(i)$, 0 otherwise. As we assume that a couple of robots can communicate each other only when there are no obstacles occluding the line of sight, we will refer indistinctly to *connectivity topology* or *communication topology*. The demonstration that all the robots see to the same centroid if and only if the graph is connected is trivial. A first important difference with respect to the case of holonomic agents is that, except the trivial case where all the robots start pointing to the centroid, the centroid position is not static but change dynamically due to the nonlinearity of the robots model. In this section we consider the case where the communication graph is *always* complete, i.e. no occlusions occur between robots.

The control law for the $i$-th robot is given by:

$$\begin{cases} u_i &= K_1 \cdot \bar{e}_i \\ \omega_i &= K_2 \cdot \bar{\alpha}_i \end{cases} \quad \text{where } K_1, K_2 > 0 \tag{4.5}$$

The stability of this control law is first proofed for a group of two robots, and we will see that in this particular case (due to geometric properties) the two controls can be decoupled. Then, we will demonstrate the stability of a group composed by at least three robots. Due to simplicity of notation, for a generic $k$-th node $\bar{\alpha}_k$ and $\bar{e}_k$ will be substituted by $\alpha_k$ and $e_k$ respectively.

**Theorem 4.1.** *Given a couple of robots kinematically modeled as in Eq 4.1 and suppose that no occlusion is present in the line of sight between them, then the control law in Eq 4.5*

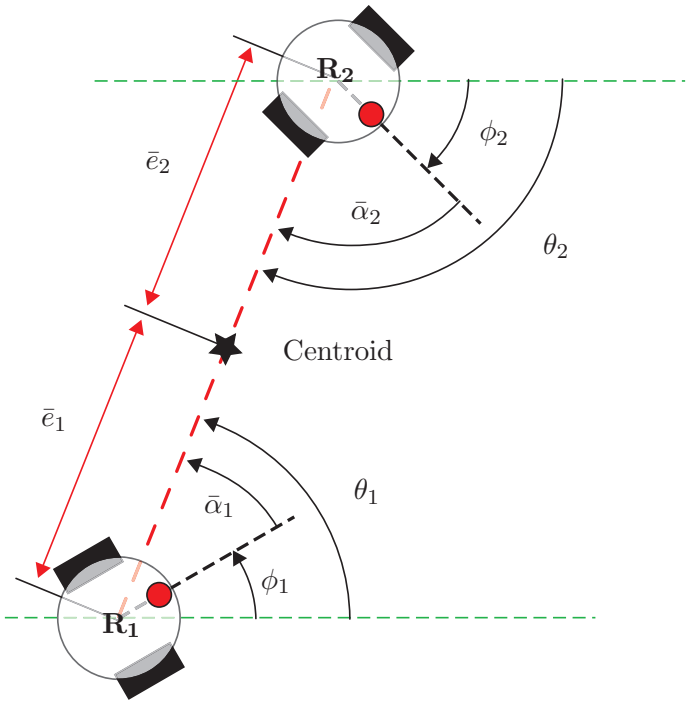



Figure 4.5: Couple of differential robots.

*drives the robot to their mean point.*

**Proof:** Consider the candidate Lyapunov function

$$V(e,\alpha) = \frac{1}{2}(e_1^2 + e_2^2 + \alpha_1^2 + \alpha_2^2) = \sum_{i=1}^{2} \frac{1}{2}(e_i^2 + \alpha_i^2) \tag{4.6}$$

As $V(e,\alpha)$ is quadratic in the relative range and bearing with respect to a calculated mean point, it is

$$\begin{cases} V_i(e_i,\alpha_i) & > & 0, \ \forall e_i \neq 0 \ \wedge \ \forall \alpha_i \neq 0 \\ V_i(e_i,\alpha_i) & = & 0, \ \forall e_i = 0 \ \wedge \ \forall \alpha_i = 0 \end{cases}$$

Deriving Eq 4.6, it is:

$$\dot{V}(e,\alpha) = e_1 \cdot \dot{e}_1 + e_2 \cdot \dot{e}_2 + \alpha_1 \cdot \dot{\alpha}_1 + \alpha_2 \cdot \dot{\alpha}_2 = \sum_{i=1}^{2} (e_i \cdot \dot{e}_i + \alpha_i \cdot \dot{\alpha}_i) \tag{4.7}$$

We can split the Eq 4.7 in two subparts:

$$\dot{V}(e,\alpha) = \begin{cases} \dot{V}_e & = & e_1 \cdot \dot{e}_1 + e_2 \cdot \dot{e}_2 \\ \dot{V}_\alpha & = & \alpha_1 \cdot \dot{\alpha}_1 + \alpha_2 \cdot \dot{\alpha}_2 \end{cases} \tag{4.8}$$

For the centroid, w.r.t. a global reference frame, it is:

$$\begin{cases} x_c &= \dfrac{1}{2}(x_1 + x_2) \\ \\ y_c &= \dfrac{1}{2}(y_1 + y_2) \end{cases} \tag{4.9}$$

where $[x_c, y_c]^T$ denotes the coordinates of the centroid. It follows:

$$\begin{cases} x_c - x_1 &= \dfrac{1}{2}(x_2 - x_1) &= e_1 \cos\theta_1 \\ \\ y_c - y_1 &= \dfrac{1}{2}(y_2 - y_1) &= e_1 \sin\theta_1 \end{cases} \tag{4.10}$$

The demonstration of stability and convergence is divided in two steps.
**Step 1:** consider the term $\dot{V}_\alpha$. To calculate $\dot{\alpha}_i$ we have to consider that $\alpha_i = \theta_i - \phi_i$:

$$\dot{\alpha}_i = \dot{\theta}_i - \dot{\phi}_i = \dot{\theta}_i - \omega_i$$

where, for $i = 1$:

$$\dot{\alpha}_1 = \dot{\theta}_1 - \omega_1 = \frac{d\theta_1}{dt} - \omega_1 = \frac{d}{dt}\left(\arctan\left(\frac{y_c - y_1}{x_c - x_1}\right)\right) - \omega_1 \tag{4.11}$$

The value of $\dot{\theta}_1$ can be calculated as follows:

$$\begin{aligned} \dot{\theta}_1 &= \frac{1}{1 + \left(\dfrac{y_2 - y_1}{x_2 - x_1}\right)^2} \cdot \frac{(x_2 - x_1)(\dot{y}_2 - \dot{y}_1) - (y_2 - y_1)(\dot{x}_2 - \dot{x}_1)}{(x_2 - x_1)^2} = \\ &= \frac{\cos\theta_1(u_2 \sin\phi_2 - u_1 \sin\phi_1) - \sin\theta_1(u_2 \cos\phi_2 - u_1 \cos\phi_1)}{e_1 + e_2} = \\ &= \frac{u_1 \sin\alpha_1 - u_2 \sin(\theta_1 - \phi_2)}{e_1 + e_2} \end{aligned}$$

Calculating $\dot{\alpha}_2$, it is:

$$\dot{\alpha}_1 = \frac{u_1 \sin\alpha_1 - u_2 \sin(\theta_1 - \phi_2)}{e_1 + e_2} - \omega_1 \tag{4.12}$$

$$\dot{\alpha}_2 = \frac{u_1 \sin\alpha_2 - u_1 \sin(\theta_2 - \phi_1)}{e_1 + e_2} - \omega_2 \tag{4.13}$$

From Fig. 4.5 it is easy to see that $e_1 = e_2 = \bar{e}$ and $\theta_1 - \theta_2 = \pi$, follows that $\theta_1 - \phi_2 = \alpha_2 - \pi$ and $\theta_2 - \phi_1 = \alpha_1 - \pi$. Now, by substituting Eq 4.12, Eq 4.13 in $\dot{V}_\alpha$ and using these equalities, it is:

$$\dot{V}_\alpha = \alpha_1 \frac{u_1 \sin\alpha_1 + u_2 \sin(\alpha_2)}{2 \cdot \bar{e}} - \alpha_1\omega_1 + \alpha_2 \frac{u_1 \sin\alpha_2 + u_1 \sin(\alpha_1)}{2 \cdot \bar{e}} - \alpha_2\omega_2$$

and, considering a control law as in Eq 4.5 and considering that $|\sin x| \leq |x|$ and that $|x| = x \cdot sign(x)$:

$$\begin{aligned}
\dot{V}_\alpha &= \alpha_1 \left( \frac{K_1}{2} \sin\alpha_1 + \frac{K_1}{2} \sin\alpha_2 - \omega_1 \right) + \alpha_2 \left( \frac{K_1}{2} \sin\alpha_1 + \frac{K_1}{2} \sin\alpha_2 - \omega_2 \right) \leq \\
&\leq \alpha_1 \left( \frac{K_1}{2} |\sin\alpha_1| + \frac{K_1}{2} |\sin\alpha_2| - \omega_1 \right) + \alpha_2 \left( \frac{K_1}{2} |\sin\alpha_1| + \frac{K_1}{2} |\sin\alpha_2| - \omega_2 \right) \leq \\
&\leq \alpha_1 \left( \frac{K_1}{2} |\alpha_1| + \frac{K_1}{2} |\alpha_2| - K_2\alpha_1 \right) + \alpha_2 \left( \frac{K_1}{2} |\alpha_1| + \frac{K_1}{2} |\alpha_2| - K_2\alpha_2 \right) = \\
&= \alpha_1^2 \left( \frac{K_1}{2} sign(\alpha_1) - K_2 \right) + \alpha_2^2 \left( \frac{K_1}{2} sign(\alpha_2) - K_2 \right) + \frac{K_1}{2} \left( \alpha_1 \cdot |\alpha_2| + \alpha_2 \cdot |\alpha_1| \right)
\end{aligned}$$

Choosing $K_2 > \dfrac{K_1}{2}$, it is:

$$\dot{V}_\alpha \leq -K_2 \left( \alpha_1^2 + \alpha_2^2 \right) + \frac{K_1}{2} \left( \alpha_1 \cdot |\alpha_2| + \alpha_2 \cdot |\alpha_1| \right) \leq 0, \quad \text{for } K2 > \frac{K_1}{2} \geq 0. \tag{4.14}$$

This means that, with the control law in Eq 4.5, the angles $\alpha_1$, $\alpha_2$ goes to 0, i.e. the vehicles point to the centroid.

**Step 2:** consider the term $\dot{V}_e$. To calculate $\dot{e}_1$ let's consider that:

$$e_1 = \sqrt{(x_c - x_1)^2 + (y_c + y_1)^2}, \tag{4.15}$$

From Eq 4.10 follows:

$$\begin{aligned}
\dot{e}_1 &= \frac{1}{2} \frac{2(x_c - x_1)(\dot{x}_2 - \dot{x}_1) + 2(y_c - y_1)(\dot{y}_2 - \dot{y}_1)}{\sqrt{(x_c - x_1)^2 + (y_c + y_1)^2}} = \\
&= \frac{e_1 \cos\theta_1 (\dot{x}_2 - \dot{x}_1) + e_1 \sin\theta_1 (\dot{y}_2 - \dot{y}_1)}{e_1}
\end{aligned} \tag{4.16}$$

and, replacing $\dot{x}_i$ and $\dot{y}_i$ with Eq 4.1,

$$\begin{aligned}
\dot{e}_1 &= \cos\theta_1 (u_2 \cos\phi_2 - u_1 \cos\phi_1) + \sin\theta_1 (u_2 \sin\phi_2 - u_1 \sin\phi_1) = \\
&= -u_1 (\cos\theta_1 \cos\phi_1 + \sin\theta_1 \sin\phi_1) + u_2 (\cos\theta_1 \cos\phi_2 + \sin\theta_1 \sin\phi_2) = \\
&= -u_1 \cos(\alpha_1) + u_2 \cos(\theta_1 - \phi_2)
\end{aligned} \tag{4.17}$$

Similarly, deriving $e_2$, we get:

$$\begin{aligned}
\dot{e}_1 &= -u_1 \cos(\alpha_1) + u_2 \cos(\theta_1 - \phi_2) &&= -u_1 \cos\alpha_1 - u_2 \cos\alpha_2 \\
\dot{e}_2 &= -u_2 \cos(\alpha_2) + u_1 \cos(\theta_2 - \phi_1) &&= -u_2 \cos\alpha_2 - u_1 \cos\alpha_1
\end{aligned} \tag{4.18}$$

Follows, using the control law in Eq 4.5:

$$\begin{aligned}
\dot{V}_e &= e_1 (-u_1 \cos\alpha_1 - u_2 \cos\alpha_2) + e_2 (-u_2 \cos\alpha_2 - u_1 \cos\alpha_1) = \\
&= e_1 (-K_1 e_1 \cos\alpha_1 - K_2 e_2 \cos\alpha_2) + e_2 (-K_2 e_2 \cos\alpha_2 - K_1 e_1 \cos\alpha_1)
\end{aligned}$$

As $e_1 = e_2 = \bar{e}$,

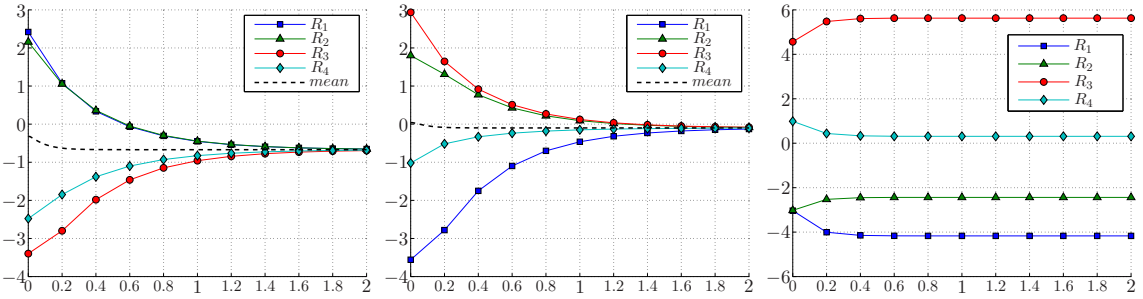
$$\dot{V}_e = -2K_1\bar{e}^2\left(\cos\alpha_1 + \cos\alpha_2\right) \tag{4.19}$$

In Eq 4.19 there can be stability problems because for $\alpha_1, \alpha_2 \in [-\pi, \frac{\pi}{2}]$ the function $\dot{V}_e$ is positive, but, as we have demonstrated in step 1, the angles converge to zero regardless the value of $e_1$ and $e_2$. So, after a finite time $T$ it is possible to say that Eq 4.19 is always lower then zero, for $K_1 > 0$. It follows that, after time $T$, Eq 4.7 is lower than zero, and the stability of the system is proved.

■

An improvement of the control can be introduced if we define $u_i = K_1 e_i \cos\alpha_i$. This changes Eq 4.19 in $\dot{V}_e = -2K_1\bar{e}^2\left(\cos^2\alpha_1 + \cos^2\alpha_2\right)$, that is always less than zero regardless the angles. Intuitively, this improvemten moves the $i$-th robot backward till it is perpendicular to the the centroid position, reducing the maneuvering space.

**Proposition 4.2.** *Given a group of $N$ robots modeled as in Eq 4.1, assuming that communication graph representing the communication topology of the swarm is complete, the control law in Eq 4.5 stabilizes the group to its the centroid.*



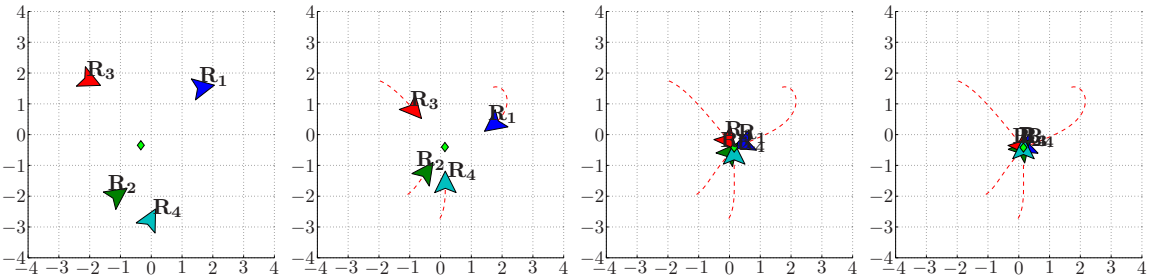

(a) Evolution of the $X$ coordinates.

(b) Evolution of the $Y$ coordinates.

(c) Evolution of $\phi_{i,i=1\ldots4}$.

Figure 4.6: Coordinates and angles evolution of a differential-wheel-four-robot group performing consensus.

In Fig. 4.6 the evolution of the $X$, $Y$ coordinates and $\phi$ angles of four robots of a group are depicted, while in Fig. 4.7 some snapshots of the same simulation are shown. Note that in Fig. 4.6(a)-4.6(b), the coordinates of the robots converge to a common point (green diamond in Fig. 4.7 represents the centroid), but the mean value is no more static (dotted line). In Fig. 4.6(c) the value of the orientations of the robots is reported: as we have supposed that any robot can access to global information, it is impossible to perform agreement also on angles, as this request to know the value of $\phi_i, i = 1 \ldots N$.




(a) $t = 0.0$ sec. (b) $t = 0.3$ sec. (c) $t = 1.0$ sec. (d) $t = 2.0$ sec.

Figure 4.7: Evolution of a four-robot group performing consensus.

## 4.3 Formation Keeping

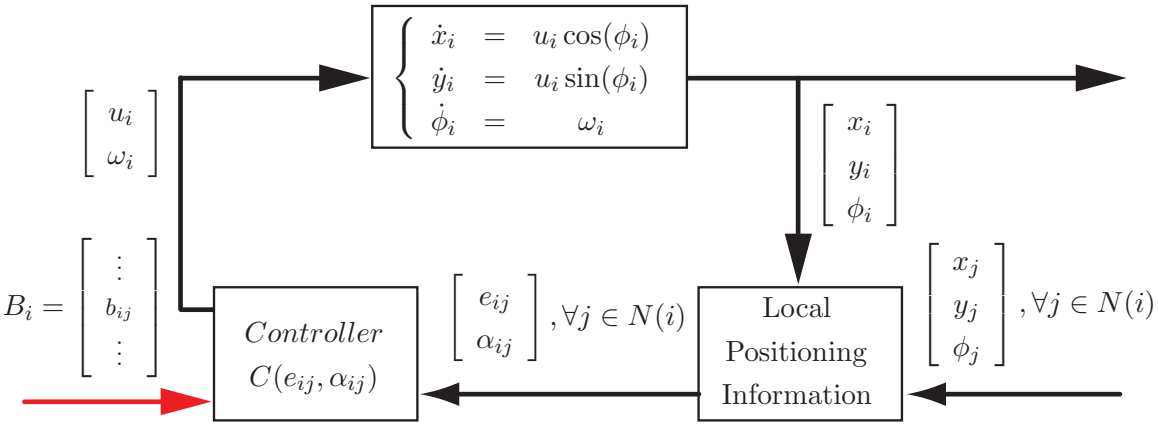



Figure 4.8: Feedback scheme for formation keeping using consensus algorithm for a group of differential-wheeled robots with local sensing.

As we have seen in Sec 3.2.1, the consensus algorithm can be used also to get a predefined formation. To introduce this possibility, the scheme in Fig. 4.3 should be slightly modified, and became as appear in Fig. 4.8. The arrow on the left side of *controller* block represents the input bias, that is a vector defined as

$$B(i,j) = \begin{cases} b_{i,j}, & \forall R_j \in \mathcal{N}_i \\ 0, & \text{otherwise} \end{cases}$$

where $b_{i,j}$ is the desired distance between robot $\mathbf{R_i}$ and robot $\mathbf{R_j}$. As we will see in Sec 4.6, this vector can be dynamically calculated based on the cardinality of set $N(i)$. To use the control law defined in Eq 4.5 in order to get a formation, the values defined in Def 4.1 are

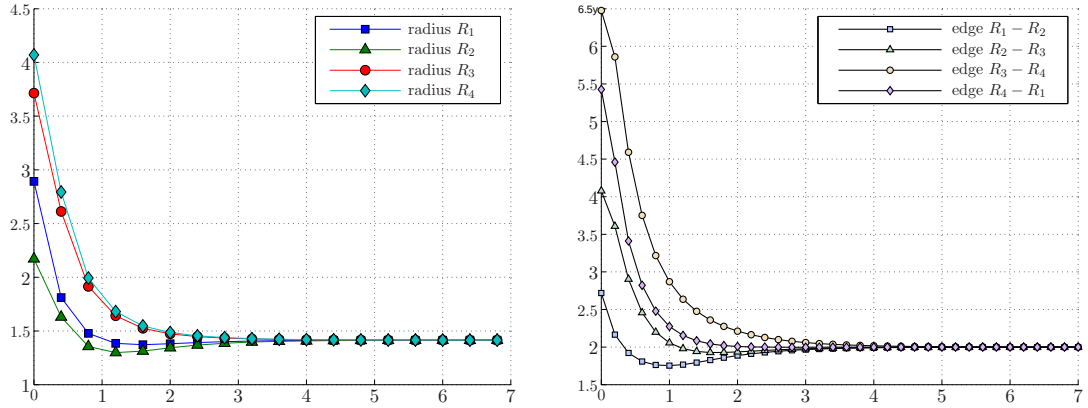
changed as follows:

$$\begin{aligned} \bar{e}_{x,i} &= \frac{1}{|N(i)|+1} \sum_{j=1}^{|N(i)|} \left[ -\mathcal{L}_{i,j} \cdot (e_{i,j} - b_{i,j}) \cdot \cos(\alpha_{i,j}) \right] \\ \bar{e}_{y,i} &= \frac{1}{|N(i)|+1} \sum_{j=1}^{|N(i)|} \left[ -\mathcal{L}_{i,j} \cdot (e_{i,j} - b_{i,j}) \cdot \sin(\alpha_{i,j}) \right] \end{aligned} \tag{4.20}$$

This means that robot $R_i$ will not point to the *real centroid* but to a *false centroid*, that depends on the desired configuration. For convenience of representation, for a group of $N$ robots all the column vectors $B_i$ can be collected in the *bias matrix* $B$:

$$B = \begin{bmatrix} \vdots & \dots & \vdots & \dots & \vdots \\ B_1 & \dots & B_i & \dots & B_N \\ \vdots & \dots & \vdots & \dots & \vdots \end{bmatrix} \tag{4.21}$$

As an example, let us consider a group of four robots achieving the square formation.




(a) Evolution of the radii

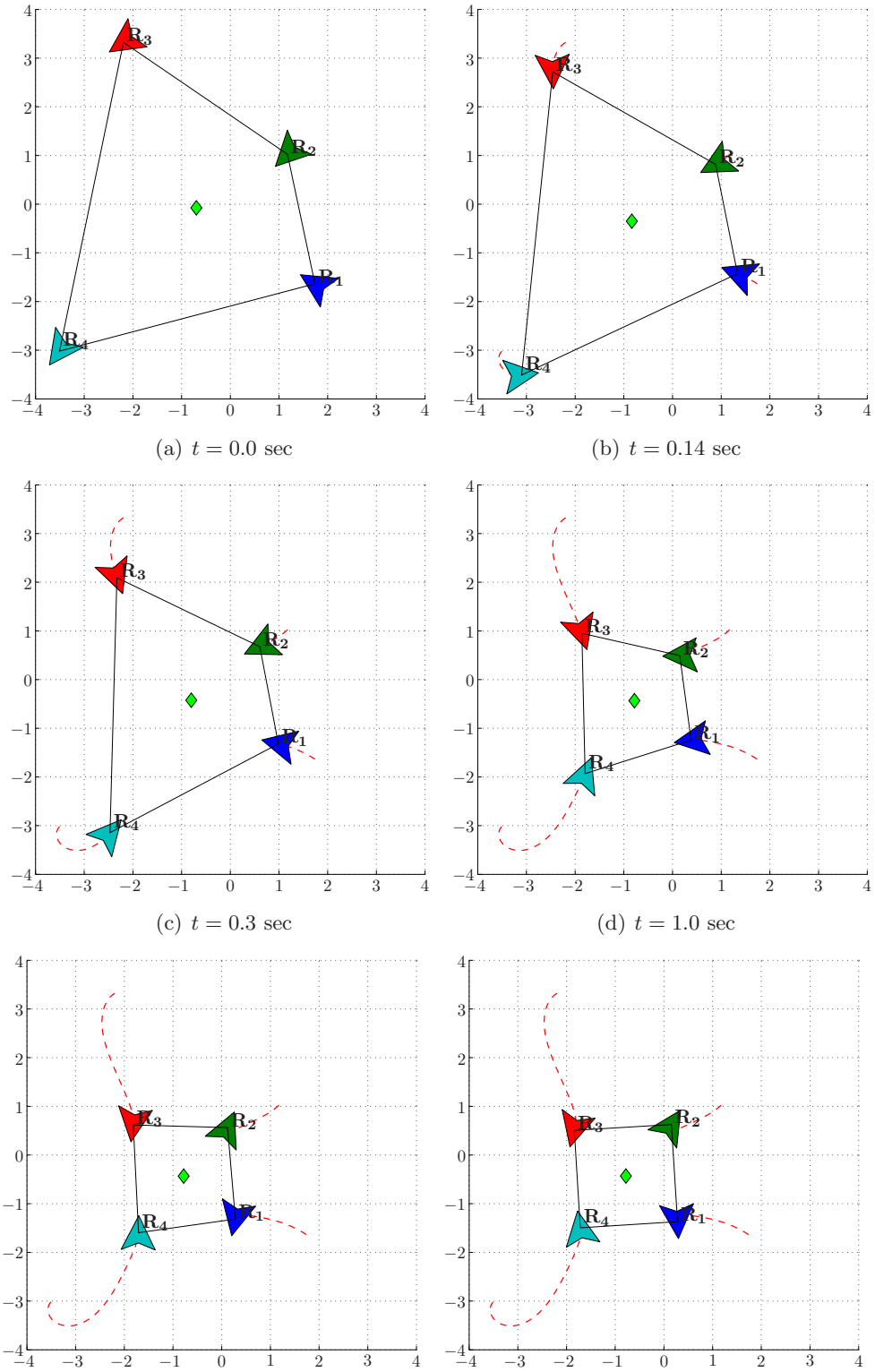
(b) Evolution of the square edges

Figure 4.9: Evolution of radii and sides of a group of four differential-wheel robots group performing consensus to achieve square formation.

The side of the desired formation is 2 and the diagonal is $2\sqrt{2}$. The corresponding $B$ matrix is:

$$B = \begin{bmatrix} 0 & 2 & 2\sqrt{2} & 2 \\ 2 & 0 & 2 & 2\sqrt{2} \\ 2\sqrt{2} & 2 & 0 & 2 \\ 2 & 2\sqrt{2} & 2 & 0 \end{bmatrix} \tag{4.22}$$

In Fig. 4.9 the evolution of radii and edges of the formation are depicted, where "radius $R_i$" represents the distance to the centroid. As expected, the edges converge to 2 while radii converge to $\sqrt{2}$. Some snapshots of the same simulation are depicted in Fig. 4.10.




(a) $t = 0.0$ sec

(b) $t = 0.14$ sec

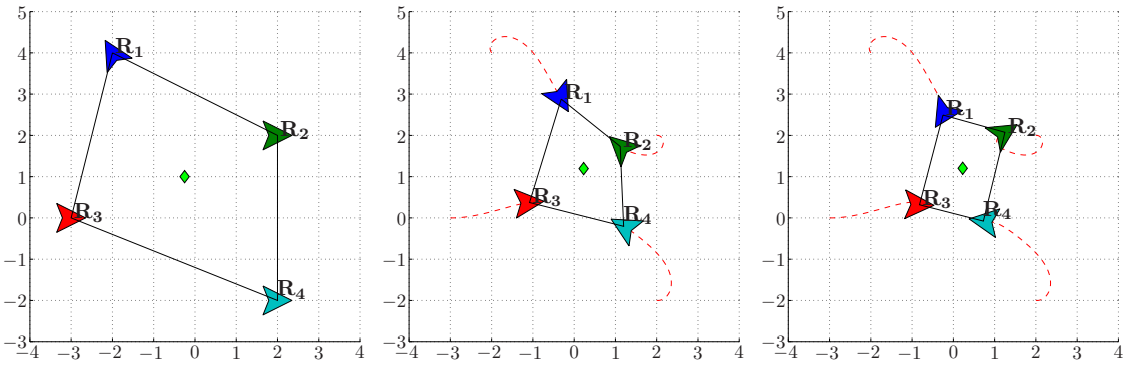
(c) $t = 0.3$ sec

(d) $t = 1.0$ sec

(e) $t = 2.0$ sec

(f) $t = 5.0$ sec

Figure 4.10: Evolution of a four-robot group achieving square formation.



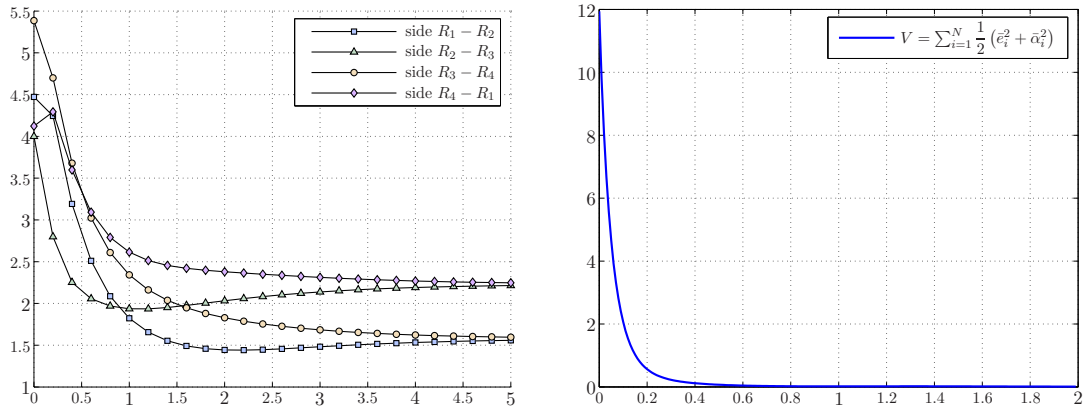

(a) $t = 0.0$ sec (b) $t = 1.0$ sec (c) $t = 2.0$ sec

Figure 4.11: Evolution of a four-robot group achieving square formation.

## 4.4 Additional Optimization

### Avoiding local minima in formation maneuvering

The use of this control law to achieve formation can present some interesting issues. The first problem arises directly from the definitions in Eq 4.20. In simulations reported in Figg 4.9-4.10 the counterclockwise order of the robots is $R_1$,$R_2$,$R_3$,$R_4$. If we suppose that the order is $R_1$,$R_3$,$R_4$,$R_2$, the system can stabilize in a local minima due to the fact that the $\bar{e}_i$ values are zero when Eq 4.20 are both zero in points other then the final configuration. In Fig. 4.11 a simulation illustrates the problem, while in Fig. 4.12 the evolution of the formation edges and of the Lyapunov function introduced in Theorem 4.1 and in Proposition 4.2 are shown. Note that even if the Lyapunov function goes to zero, the sides converge to different values.




(a) (b)

Figure 4.12: Evolution of the formation edges (4.12(a))and of the Lyapunov function (4.12(b)) introduced in Theorem 4.1.

A possible way to avoid this problem is to relabel each robot with a virtual label that can be different respect the real one. As a consequence, the rows of the bias vector $B_i$ have to be exchanged. In order to reach a consensus on the virtual labeling, all the robot of the swarm relabel the others in the same way. If each robot has a different ID, the relabel algorithm is:

**Algorithm 4.1** (Relabel algorithm for the $i$-th robot)**.**

```
while (1)
    find swarm centroid;
    Sort robots in CCW starting from the robot with lower ID;
    if (more robot are aligned)
        then sort based on distance w.r.t. centroid;
    endif
    exchange rows of B_i;
    perform consensus;
endwhile
```

Using the same initial conditions of the previous example, in Figg 4.13- 4.14 the behavior of the swarm is reported, where at time $t = 5.0$ sec. robots start reconfiguring to avoid local minima. Note that at time $t = 5.0$ sec. the Lyapunov function has a peak due to the relabeling algorithm.

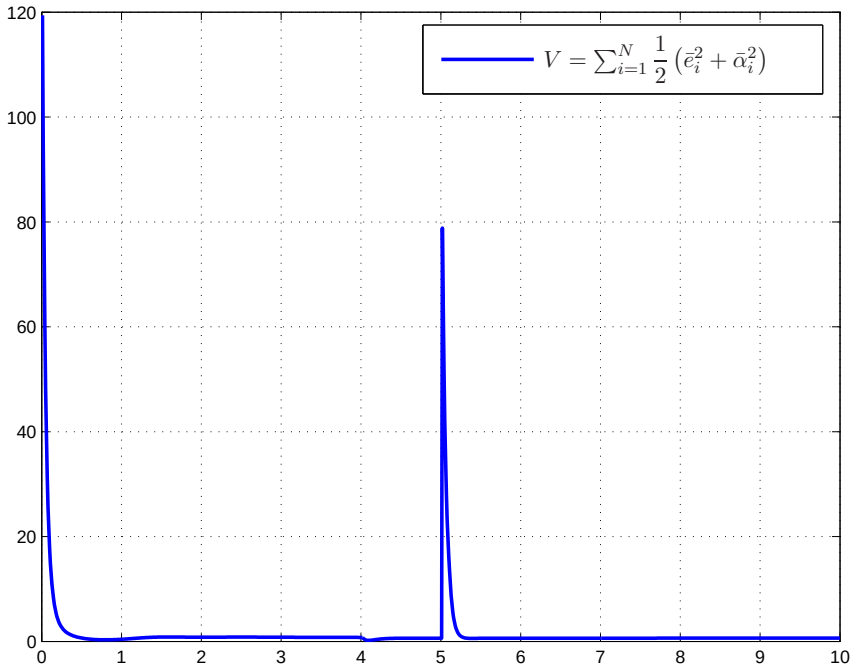
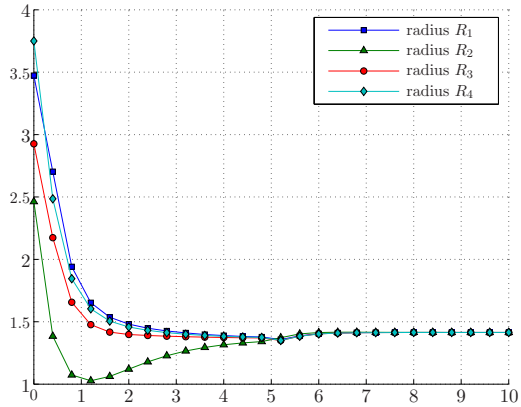
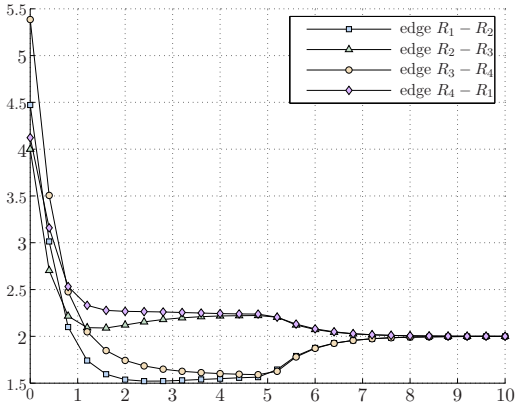



Figure 4.13: Evolution of the Lyapunov function in case of relabeling.




(a) Evolution of the radii in presence of relabeling.




(b) Evolution of the square edges in presence of relabeling.

Figure 4.14: Evolution of radii and edges of a four-differential-wheel-robots formation performing relabeling.

### Data Broadcasting

So far we have supposed that each robot can sense the others, i.e. the communication graph is complete. The problem is that robots navigating through a terrain can often encounter obstacles that can prevent point-to-point communication and drive the system to instable configurations. In fact, if two robots sense two different subsets of neighbors, they can point to different centroids. To make the system more stable and redundant we can introduce the ability for the robots to transmit information, broadcasting relative positioning data. Fig. 4.15 depicts a typical situation where broadcasting is useful. In this figure, the line-of-sight between robot $\mathbf{R_0}$ and robot $\mathbf{R_2}$ is blocked by a wall. Fortunately robot $\mathbf{R_1}$ can be used as a relay to compute the missing information about the position of $\mathbf{R_2}$ with respect to $\mathbf{R_0}$. The idea is to have $\mathbf{R_1}$ broadcasts its relative positioning information (in our case $\alpha_{10}$, $e_{10}$, $\alpha_{12}$ and $e_{12}$, where $e_{ij}$ is the distance sensed by $\mathbf{R_i}$ to $\mathbf{R_j}$). This information enables $\mathbf{R_0}$ to compute an estimation of $\alpha_{02}$ and $e_{02}$ with the following equations:

$$\begin{cases} \beta_{02} &= \pi + \alpha_{01} - \alpha_{10} + \alpha_{12} \\ e_{x,02} &= e_{01}\cos(\alpha_{01}) + e_{12}\cos(\beta_{02}) \\ e_{y,02} &= e_{01}\sin(\alpha_{01}) + e_{12}\sin(\beta_{02}) \\ e_{02} &= \sqrt{e_{x,02}^2 + e_{y,02}^2} \\ \alpha_{02} &= \text{atan2}(e_{y,02}, e_{x,02}) \end{cases}$$

This ability allow us to relax the *complete* constraint used in Theorem 4.1 and in Proposition 4.2 to a *connected* constraint, means that as long as each robot sees at least one robot of the group, the system converges. The data broadcasted by the range and

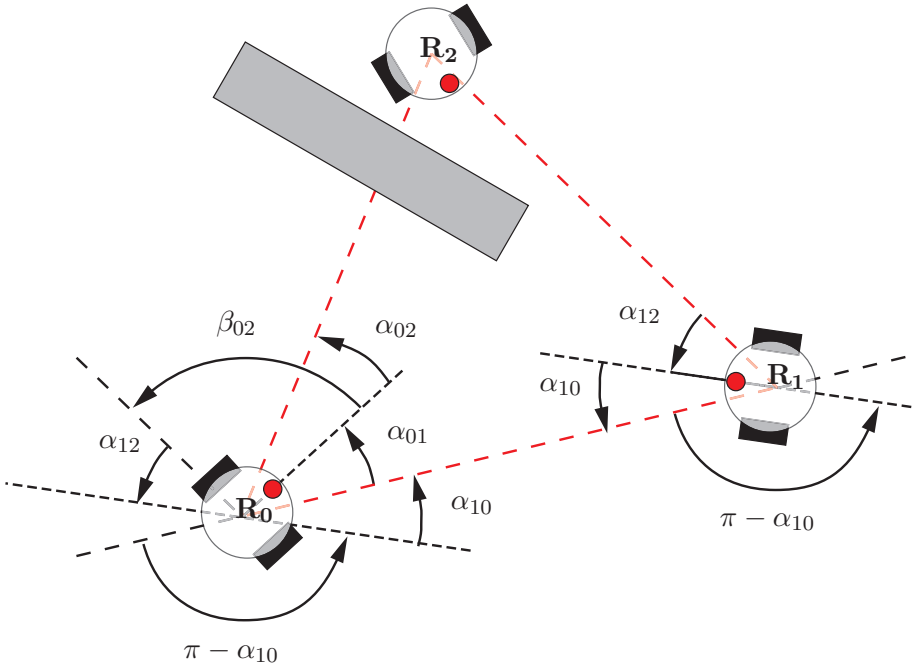



Figure 4.15: Example of a blocked line-of-sight where broadcasting is important.

bearing board is grouped into a data packet. Tabel 4.1 shows how the packet sent by $\mathbf{R_0}$ and $\mathbf{R_1}$ in our previous example looks like. This data packet not only contains the measurements ($\alpha_{ij}$ and $e_{ij}$, for all $j$ such that $\mathbf{R_i}$ is sensing $\mathbf{R_j}$), but also contains a *hop count* $h_{ij}$ that is incremented each time a robot broadcasts positioning data that it is not directly measuring. In Fig. 4.15, the *hop count* $h_{02}$ will be one, meaning that $\mathbf{R_0}$ used one relay to compute $\alpha_{02}$ and $e_{02}$. The data broadcasted can be ignored if the *hop count* is too high ($h_{ij} > h_{\max}$).

| Data packet sent by $\mathbf{R_1}$ |
|---|
| 0 (Sender ID) |
| 1 |
| $\alpha_{01}$ |
| $e_{01}$ |
| 0 ($= h_{01}$) |
| 2 |
| $\alpha_{02}$ |
| $e_{02}$ |
| 1 ($= h_{02}$) |

| Data packet sent by $\mathbf{R_2}$ |
|---|
| 1 (Sender ID) |
| 0 |
| $\alpha_{10}$ |
| $e_{10}$ |
| 0 ($= h_{10}$) |
| 2 |
| $\alpha_{12}$ |
| $e_{12}$ |
| 0 ($= h_{12}$) |

Table 4.1: Broadcasted data packet format.

### Obstacle Avoidance

To avoid obstacles, robots, tested later in our experiments, use a Braitenberg controller [17] on top of the formation control introduced in the previous sections. Sometimes, a robot would incur in a deadlock and oscillate undecided whether to avoid an obstacle or keep the formation. We introduce a counter that is increased by two when the robot is too close to an obstacle and reduced by one otherwise (values of the counter are between zero and a maximum value $C_{max}$). Whilst the counter is greater than zero, the robot follows the Braitenberg rules. This asymmetric counter makes sure that the robot gets more time to avoid the obstacle and avoids in practice these deadlock situations.

## 4.5 Range and Bearing board: relative positioning algorithm

To calculate the range and bearing of a transmitting robot using the Received Signal Strength Indication (RSSI) values, we employ a variation of the algorithm described by Pugh and Martinoli in [74] (in this case RSSI refers to the power present in a received infrared signal, in opposition with the common usage for the power present in a received radio signal). The original algorithm assumed a specific angular reception strength profile for photo-diodes and aggregated the RSSI from a series of receivers to calculate an estimate of both the range and bearing of the transmitting robot. We use a very similar approach, but with a different angular reception strength profile.

The algorithm requires that receivers are evenly-spaced around the perimeter of the robot. For any received transmission, approximately half these sensors will detect the signal. We can define a "sector of interest" as some sector of $n$ sensors with the highest received signal, where $n$ is at most half of the total sensors. We now only need to calculate the angle offset ($\theta$) from the center of this sector to find the bearing of the transmitting robot ($\phi$).

Upon measuring the signal intensity at the individual receivers for different receiver orientations, we discovered that the intensity is closely modeled by:

$$r' = r\cos(\theta)$$

where $r$, which we will call the range term, would be the signal intensity at a receiver which is directly facing the transmitting robot. Let us define $m = \lfloor \frac{n}{2} \rfloor$ as the number of sensors in one half of the receiving sector. Let $r'_{-1}, ..., r'_{-m}$ be values of the sensors which have a lesser angle than the center of the receiving sector, $r'_1, ..., r'_m$ be the values of the sensors which have a greater angle than the center of the receiving sector (with $r_0$ as the value of the center sensor if $n$ is odd), and $\beta_i$ be the angular offset of sensor $i$ from the

sector center (for a visual depiction of an example module with $n = 3$ and $m = 1$, see Fig. 4.16). The value $r'_i$ is given by:

$$r'_i = r\cos(\theta - \beta) \quad \text{with} \quad \beta_i = -\beta_{-i}$$

Therefore:

$$\begin{aligned} r'_i + r'_{-i} &= r\cos(\theta - \beta_i) + r\cos(\theta + \beta_i) \\ &= 2r\cos(\theta)\cos(\beta_i) \\ r'_i - r'_{-i} &= r\cos(\theta - \beta_i) - r\cos(\theta + \beta_i) \\ &= 2r\sin(\theta)\sin(\beta_i) \end{aligned}$$

Let:

$$a = \frac{\sum_{i=0/1}^{n} r'_i + r'_{-i}}{\sum_{i=1}^{n} 2\cos(\beta_i)} = r\cos(\theta)$$
$$b = \frac{\sum_{i=0/1}^{n} r'_i - r'_{-i}}{\sum_{i=1}^{n} 2\sin(\beta_i)} = r\sin(\theta)$$

so that:

$$\theta = \arctan(\frac{b}{a}), \quad r = (a^2 + b^2)^{\frac{1}{2}}$$

which exploits the trigonometric identity $A\cos^2(x) + A\sin^2(x) = A$. The initial term of the sums for $a$ and $b$ is 0 if $n$ is odd (center sensor should be included) and 1 if $n$ is even.

We can apply this algorithm to our current system. With some empirical trials, we determined that the best accuracy was obtained using three of the eight total sensors. Given the signal strength at all infrared receivers, we can find the sector of three sensors with the strongest total received signal; this provides us with the values $r'_{-1}$, $r'_0$, and $r'_1$. In this system, $\beta_1 = \frac{\pi}{4}$, and thus:

$$r'_{-1} = r\cos(\theta + \frac{\pi}{4}), \quad r'_0 = r\cos(\theta), \quad r'_1 = r\cos(\theta - \frac{\pi}{4})$$

Based on this, we get the formulas:

$$a = \frac{r'_1 + r'_{-1} + 2r'_0}{2\cos(\frac{\pi}{4}) + 2}, \quad b = \frac{r'_1 - r'_{-1}}{2\sin(\frac{\pi}{4})}$$

which we can use to get:

$$\begin{aligned} \theta &= \arctan(\frac{b}{a}), \quad r = (a^2 + b^2)^{\frac{1}{2}} \\ \phi &= \text{quadrant angle} + \theta \end{aligned}$$

where $\phi$ specifies the bearing and $r$ can be converted into range using a lookup table.

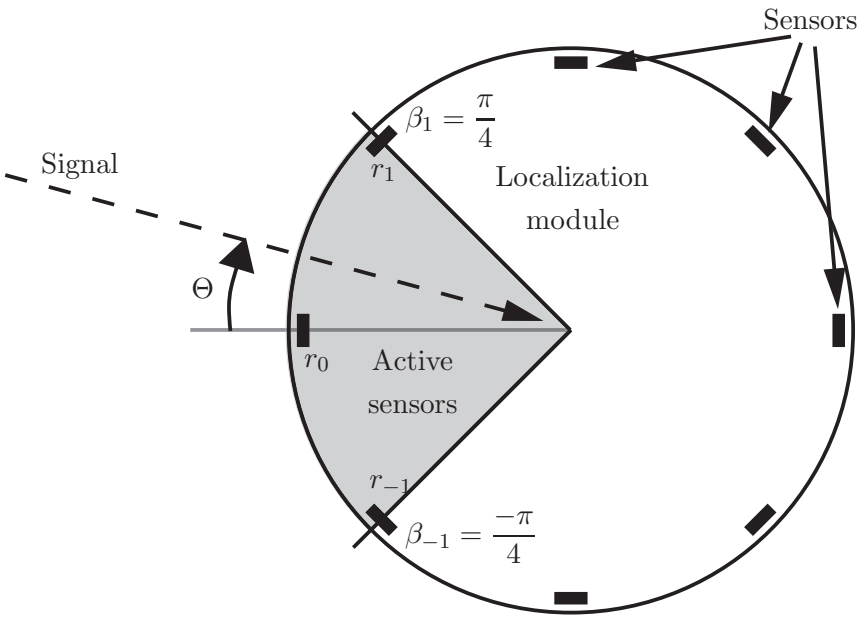



Figure 4.16: Relative Positioning Module Receiving Signal

## 4.6 Simulation Results

Experiments were conducted using Webots [64], a realistic mobile robotic simulator, and simulated Khepera III robots [91]. All sensors and actuators of the simulated robotic platform, including the range and bearing module introduced in Sec 4.5, were calibrated to match reality. In particular, the range and bearing module suffers from a 10% noise ratio in the estimation of the distance and a 0.1 radians noise in the relative angle.

### 4.6.1 Experimental Setup

At the beginning of each simulation run, four robots and either zero or ten obstacles are randomly placed in a 3 x 3 m area in the middle of a 4 x 4 m arena. Figure 4.17 shows an example of initial and final positions for the robotic nodes. Obstacles are represented by cylinders with a 10 cm radius.

The goal of the four robots is to converge to a square formation where the diagonal of the square is one meter. Thus, they use the *Bias* matrix $B$

$$B = \begin{bmatrix} 0 & 1 & 1\sqrt{2} & 1 \\ 1 & 0 & 1 & 1\sqrt{2} \\ 1\sqrt{2} & 1 & 0 & 1 \\ 1 & 1\sqrt{2} & 1 & 0 \end{bmatrix}$$

. The controller of the robot has $K_1 = 25000$ and $K_2 = 50000$. It is important to note

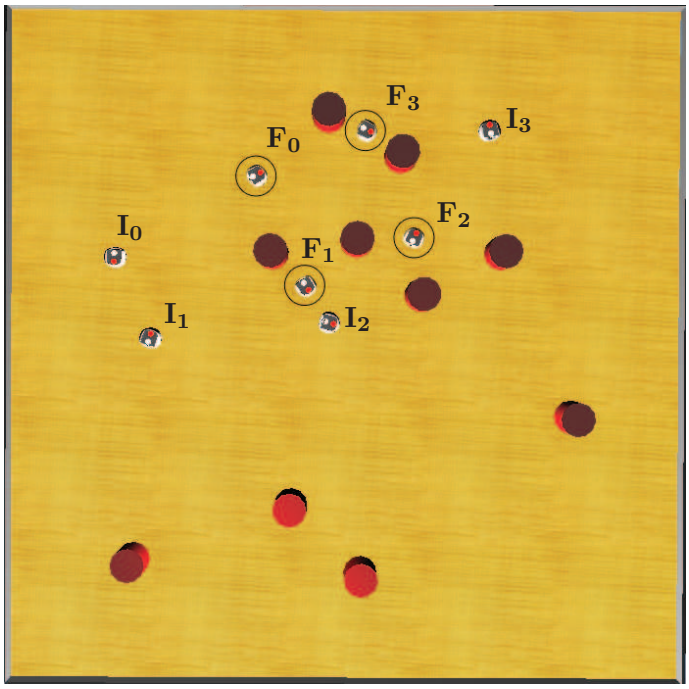



Figure 4.17: Example of an experiment with ten obstacles and four robots. Obstacles are in red, robots are white. $\mathbf{I_i}$ and $\mathbf{F_i}$ denote the initial and final position of robot $\mathbf{R_i}$ respectively.

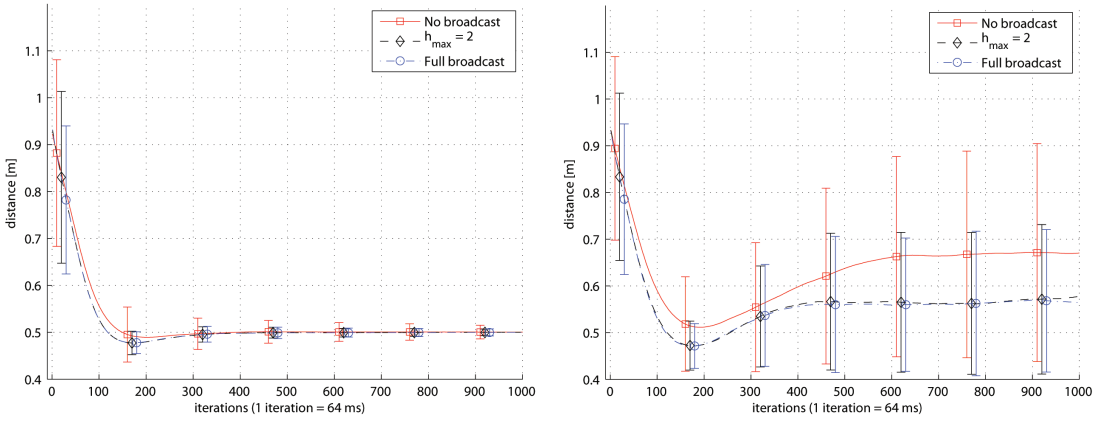
that if at any time $\|N_i\| = 0$, the robot $\mathbf{R_i}$ will start to move randomly to acquire the positioning information again.

### 4.6.2 Results

The position of each robot is monitored during a run. After 1000 runs, distances to the center of mass of the robots are computed to assess the convergence of the algorithm. Since the robots need to converge to a square configuration with a diagonal of one meter, their distances to the center of mass should converge to half a meter. Two main scenarios are tested, one without obstacles and one with ten obstacles. Each scenario is subdivided into two test cases: perfect links between robots (no packet loss) and intermittent connectivity. During the intermittent connectivity, or unstable connectivity, positioning and communication links are unstable and can go up or down with constant probability a rate corresponding to a Poisson distribution. We chose a mean time constant for the Poisson process of 10 sec. Furthermore we analyze the usefulness of the broadcasting algorithm by varying $h_{\max}$, the maximal hop count.

Figure 4.18(a) shows the evolution of the distance to the center of mass with perfect connectivity and no obstacle present. We can see that, even though there are no obstacles,

point-to-point communication can still break when another robot passes in between two other robots. Hence, a two hop broadcasting enables a faster convergence of the algorithm. This figure also shows that the approach used converges.




(a) (b)

Figure 4.18: Average and standard deviations of the distances to the center of mass depending on time without obstacles and using a perfect connectivity 4.18(a) and an unstable connectivity 4.18(b).

Figure 4.18(b) shows that, even though the communication links can easily go down, the system stays stables. Since a robot starts to move randomly when it loses all its communication links, the figure shows a mean of the distance to the center of mass a little higher than half a meter and large standard deviations. Again it is worth noting that a two hops broadcasting achieves similar performances to the full broadcasting and we can safely have the robots not send data with two hops or more, thus saving energy.

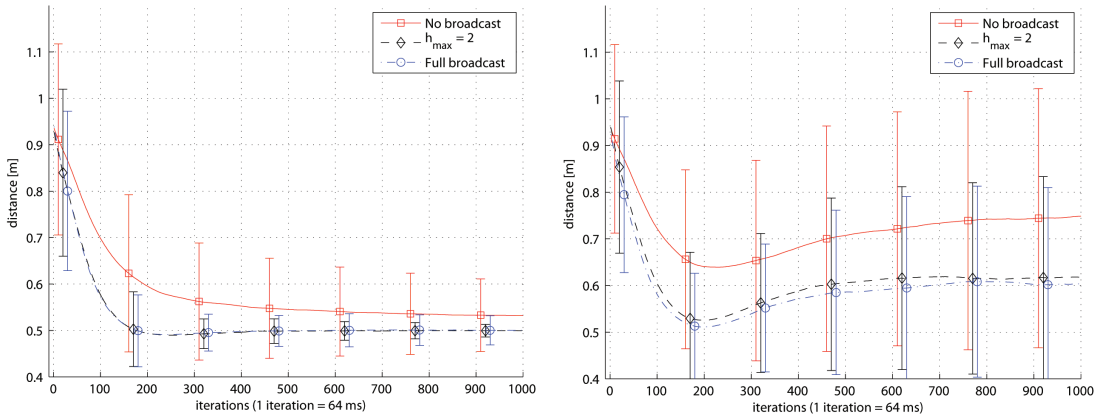
Figure 4.19(a) and Figure 4.19(b) really demonstrate the importance of broadcasting in real - not trivial - environments. The figures show that the control law used enables convergence even in more challenging scenarios. Again two hops broadcasting is achieving identical performances to the full broadcasting.

### 4.6.3 Application example: target hunting

Let us recall that in Eq 4.20 the values used to define the control law depend on the connectivity matrix $\mathcal{L}$. Till now, we have supposed that:

$$\mathcal{L}_{ij} = \begin{cases} |N(i)| & \text{if } i = j \\ g(N(i), t) & \text{if } R_j \in N(i) \\ 0 & \text{otherwise} \end{cases}$$

where $g(N(i), t) = 1$. The function $g(N(i), t)$ can be defined dynamically in order plan more complex tasks. The key point is to remember the definition of the *weighted Laplacian*




(a) (b)

Figure 4.19: Average and standard deviations of the distances to the center of mass depending on time with obstacles and using a perfect connectivity 4.19(a) and an unstable connectivity 4.19(b).

*matrix* $\mathcal{L}_{\mathcal{W}}$ given in Eq 3.32, where the terms $g(N(i))$ for a system able to preserve connectivity are defined in Eq 3.37.

The same approach can be use to coordinate a group of differential-wheeled robots. As an example where a weighted connectivity matrix is used, consider the *target hunting* task: given a target object that can be located by robots, the task is to surround it and achieve a regular polygon formation in order to reduce the escape possibilities. Moreover, we are interested in modeling a different behavior for the swarm when the target is too far to be surrounded. If we suppose that the target position is always known by robots, it can be considered as an extra vehicle. The incidence matrix and the Laplacian matrix can be defined as:

$$\tilde{\mathcal{I}} = \left[\begin{array}{ccc|ccc} & & & 1 & \cdots & 0 \\ & \mathcal{I} & & 0 & \cdots & 0 \\ & & & \vdots & \ddots & \vdots \\ & & & 0 & \cdots & 1 \\ \hline 0 & \cdots & 0 & -1 & \cdots & -1 \end{array}\right] \quad \Rightarrow \quad \mathcal{L} = \tilde{\mathcal{I}} \cdot \mathcal{W} \cdot \tilde{\mathcal{I}}^T \tag{4.23}$$

where $\mathcal{I}$ represent the incidence matrix of the group. The weight matrix is defined using two different coefficients $w_T$, $w_F$, that represent the weight of the robot-to-target and robot-to-robot communication links respectively, i.e. each robot performs the consensus algorithm sensing robots other than target (see Eq 4.24).

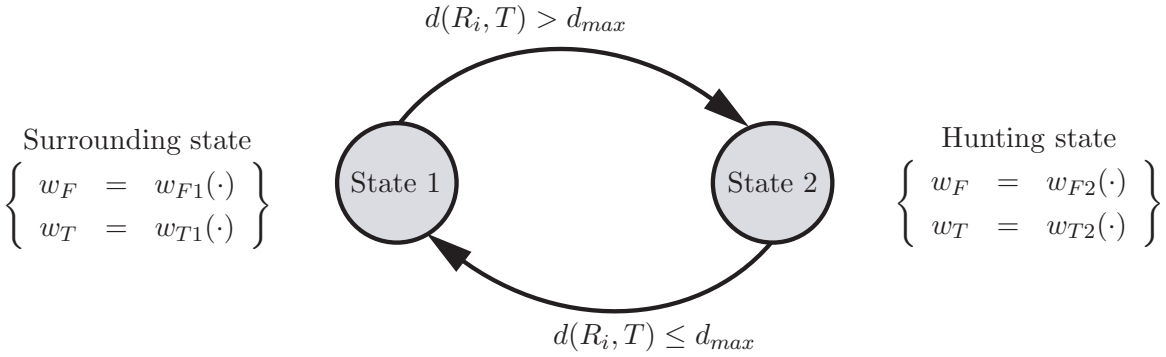



Figure 4.20: Automaton that summarizes complex behavior in target hunting task.

$$\begin{array}{rcl} \mathcal{W} & = & \text{diag}(w_k),\ k = 1, \ldots, |E| \\ & \Updownarrow & \\ w_k & = & \begin{cases} w_F & \text{if } e_k = (R_i, R_j) \\ w_T & \text{if } e_k = (R_i, T) \end{cases} \end{array} \tag{4.24}$$

where $E$ is the edge set for the given communication topology. Exploiting the use of different weights, a complex control law can be defined depending on the distance from the target. Defining $d(R_i, T)$ as the distance between the $i$-th robot and the target and $d_{max}$ as a predefined distance threshold, the control is summarized by the automaton in Fig. 4.20.

The weights $w_{F1}(\cdot)$, $w_{T1}(\cdot)$, $w_{F2}(\cdot)$, $w_{T2}(\cdot)$ can be continuous function or constants.

In Fig. 4.24 snapshots of a simulation are reported, where

$$\begin{cases} w_{F1}(\cdot) = 1 \\ w_{T1}(\cdot) = 1 \end{cases} \qquad \begin{cases} w_{F2}(\cdot) = 0.05 \\ w_{T2}(\cdot) = 1 \end{cases}$$

The blue ball represents the target. Obstacles and target are placed randomly in the arena and the regular polygon formation is an octagon with radius $d = 0.5$. Note that in Figg 4.24(a)-4.24(c) the group do not care about formation but flocks toward the target, while in Figg 4.24(d)-4.24(f) the target is close enough and the group surrounds the target.

Using nine robots with the same setup introduced in Sec 4.6.2, simulations were performed in an environment with five obstacles. In Figg 4.22-4.23 the evolution of the mean value and the standard deviation of distance to the target over 1000 simulations are shown, respectively in case of perfect connectivity and of instable connectivity (mean with 50% of probability of loosing connection and 50% probability of re-establish it). In both cases it is $h_{max} = 8$, i.e. the system is fully broadcasted.

In particular, all the robots start from a free-obstacles zone in the upper part of the arena, and the target is placed randomly. Moreover, at time $t = 32$ sec. the position of

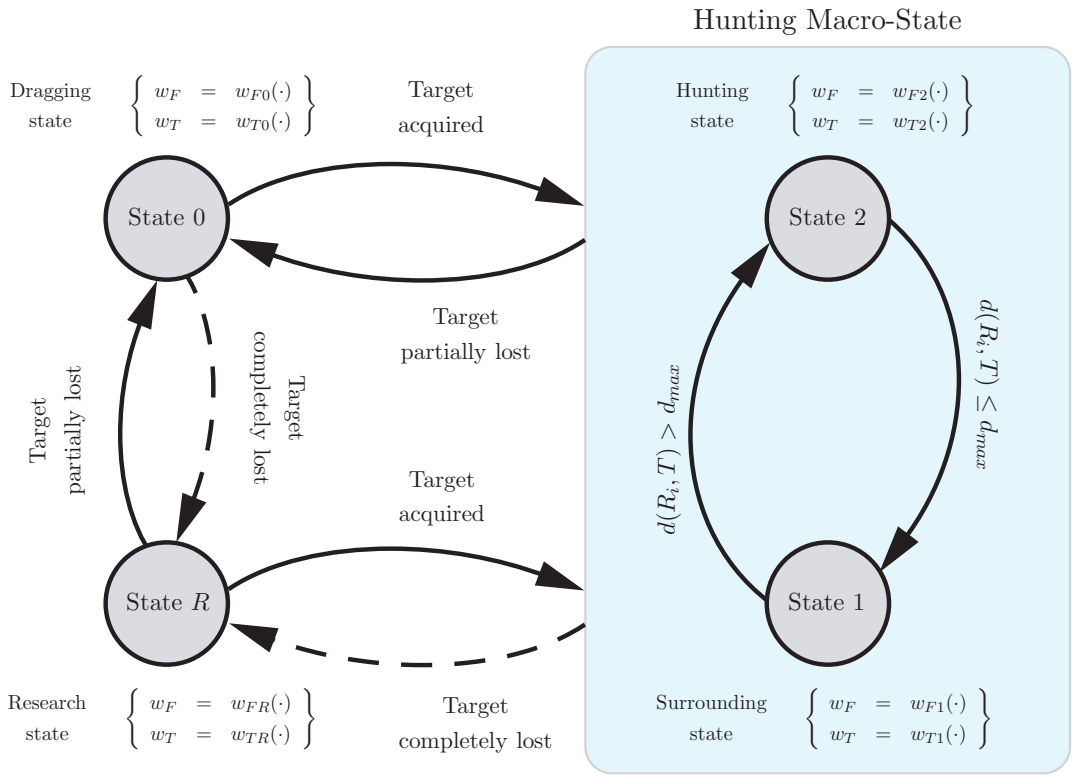



Figure 4.21: Automaton that summarizes complex behavior in target hunting task with *research* state.

the target is randomly changed instantaneously (during the same simulation run). This justifies the peak in the graphs.

It is possible to see that, in case of perfect connectivity using 9 robots, the system converges to a distance to the target $d = 0.5$ in both cases with full broadcast and with $h_{max} = 2$. In case of no broadcast the mean value tends to $d = 0.5$ but the standard deviation is too high.

The last improvement of the algorithm in case of hunting tasks is summarized by the automaton in Fig. 4.21, where the possibility that the group looses the target is considered and

$$\begin{cases} w_{F0}(\cdot) = 1 \\ w_{T0}(\cdot) = 0 \end{cases} \qquad \begin{cases} w_{FR}(\cdot) = 0 \\ w_{TR}(\cdot) = 0 \end{cases}$$

The basic idea is that when a robot sees others but, due to a lower number of $h_{max}$ or to a bad communication link, do not know the target position, tries to perform the formation task instead of moving randomly in order to find the target. This because as long as at least one robot senses the target, it move towards it dragging the whole group.

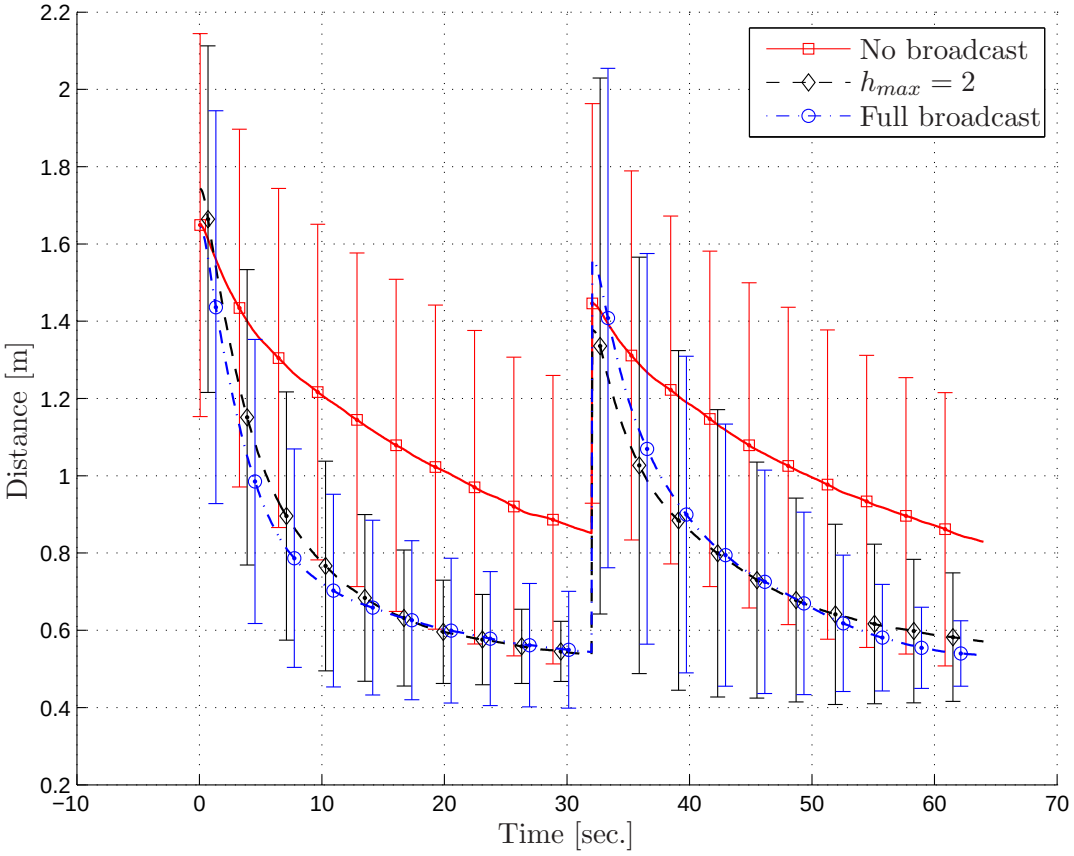



Figure 4.22: Average and standard deviations of the distances to the target depending on time with obstacles and using a perfect connectivity. The peak at $t = 32$ sec. means that the target is moved instantaneously to another random location.

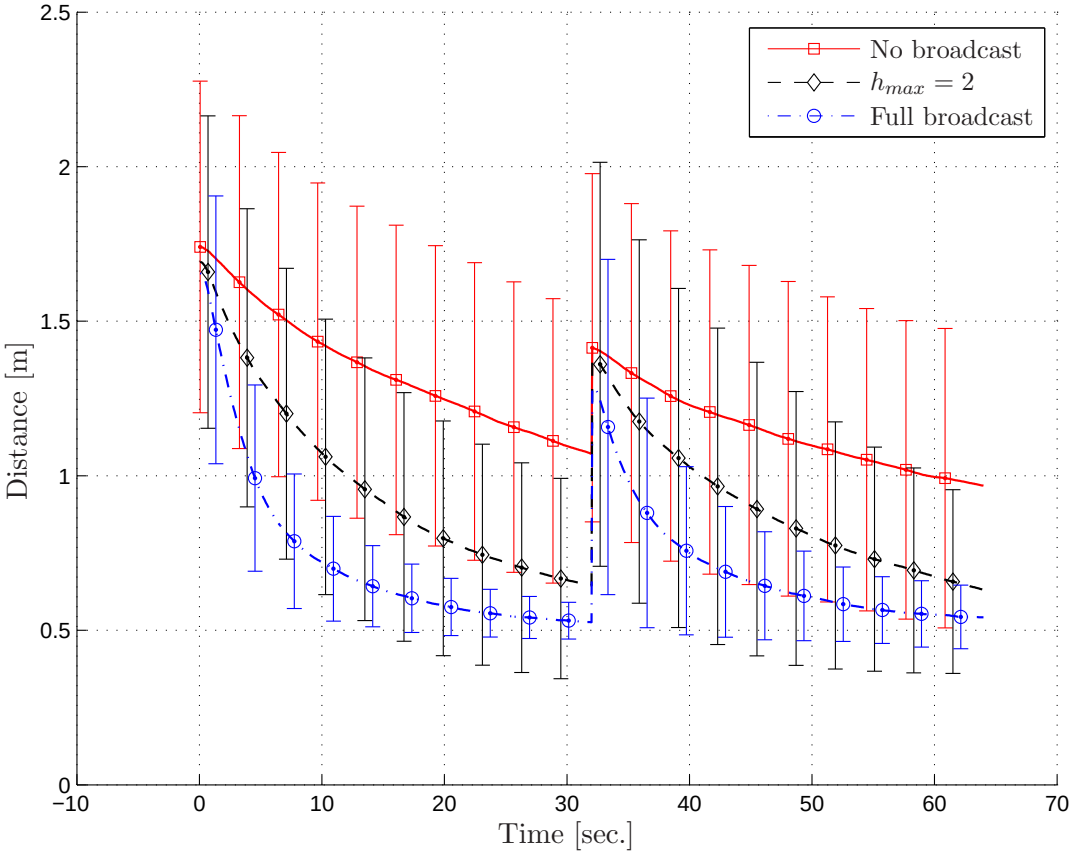
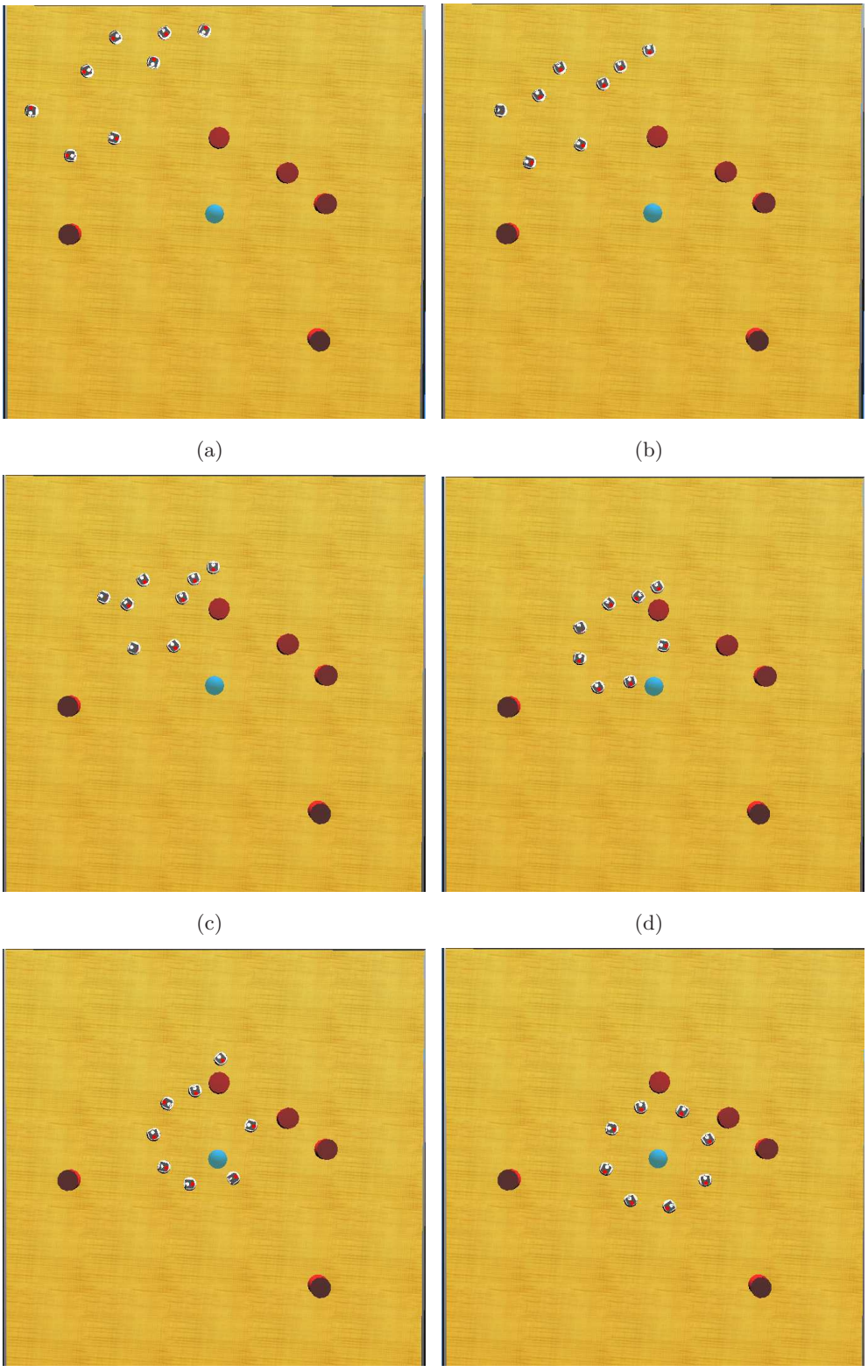



Figure 4.23: Average and standard deviations of the distances to the target depending on time with obstacles and using bad connectivity. The peak at $t = 32$ sec. means that the target is moved instantaneously to another random location.



(a) (b) (c) (d) (e) (f)

Figure 4.24: Group of robots using weighted Laplacian matrix to surround a target.

### 4.6.4 Real robots

In the end, some experiments were performed on real robots. For our experiments we used the Khepera III robotic platform (see Fig. 4.25) produced by K-Team Corporation that, with a diameter of 12 cm., make it suitable for indoor experiments. As explained

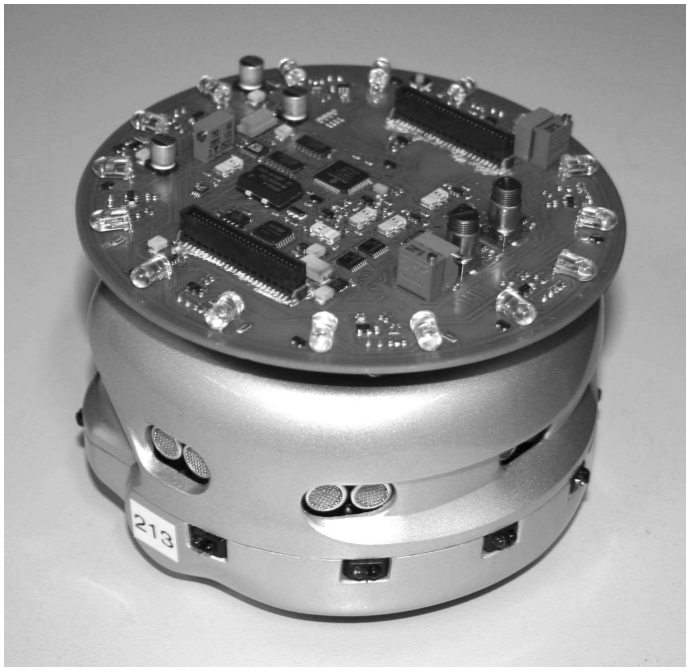



Figure 4.25: Khepera III robot with relative positioning module attached

in [45], the range and bearing module has the possibility of transmit data packet, thus in our experiments we were able to implement the data broadcasting.

# Chapter 5

# Conclusions and final remarks

This thesis deals with the problem of implementing different algorithms in order to coordinate a group of robots moving in unknown environments. More precisely, our work concerns solution of two different type of problems, in particular the coordination of a robotic swarm moving while hunting a target using a set of global information and the possibility of achieve a predefined formation using only local sensing. In Chapter 2, after a brief introduction on the basic concepts about the behavioral control of robots, we focus on a decentralized implementation of the so called *null space behavioral control* introduced in Sec 2.1.3. Basically, by exploiting the theory originally originally developed for the calculation of inverse kinematic of redundant manipulators, the possibility of merging different tasks with assigned priority level is shown, and a decentralized implementation based only on local communication is explained. This approach allow us to avoid all the problems related to conflicting tasks: in fact, if at least two of them try to satisfy reference functions that should drive the robot in different directions, the lower tasks are partially satisfied only insofar as they are not in contravene with the higher one, that is completely satisfied. As an example we choose a system composed by robots kinematically modeled as holonomic vehicles with non-null diameter. Moreover, each robot has a ring of eight proximity sensors modeled like infrared sensors, a limited ability to transmit data and a limited transmission range; the target visibility range, that allows each robot to see the target, is supposed limited. In particular, we have supposed that the data transmitted by the robots concern only their position, the data sensed by their on board sensors, and, eventually, the position of the target. Exploiting this limited set of information, we have created an algorithm able to move the robots toward a target or to follow it. More exactly, the algorithm is able to move the *centroid* of the group toward the target position; once the target is reached, low priority tasks can move the agents in order to surround the target and limiting the escape possibilities. This complex behavior is pursued by the implementation of three tasks, that are: match group centroid with target position, achieve

a predefined distance from the target (i.e. the robots must be on a circle surrounding the target) and maintain a fixed distance between a robot and its two nearest neighbors. The main problem for robots that moves in unknown environments is to avoid obstacles and, in case, to avoid configurations where they get stuck in local minima such that the group can not accomplish the desired mission. The obstacle and local minima avoidance algorithm implemented in our control is based on the cooperative paradigm as introduced in Sec 2.1.2 in order to mix the contributions generated by potential field applied to the data regarding obstacles detected by on board sensors, obstacles detected by other robots (whose positions are broadcasted) and the position of other robots in the same *communication* group. In particular, the algorithm is able to create a virtual target that can drive the single robot (or the entire swarm) out of a critical situation. The possibility of applying the obtained results to autonomous agents modeled as differential-wheel vehicles is shown Sec **??**, where this algorithm was developed to coordinate a group of Autonomous Underwater Vehicles (AUV), . In Sec 2.4, our approach is verified with many simulations where a single robot and a group of agents are moving in an arena with randomly placed obstacles in both cases with fixed and moving target.

The second part of this work is centered on the control theory applied to systems where the interconnections between agents are modeled as *interconnected graphs*. In Chapter 3 the basic notions about graph theory are summarized. The chapter introduces the reader to the basic notations necessary to handle control concepts applied to graph theory, in particular focusing on the possibility of using this approach to solve the so called *consensus problem*. This problem, whose basic solution is explained in Sec 3.2, deals with the idea of driving the states of a system to a final common value. In particular, if applied to a $N$-dimensional problem where the states represent the positions of mobile agents, the solution of the consensus problem allows to solve many other problems, such as formation keeping. Moreover, assuming that some agents are directly controlled from a high level control while others move solving the consensus, problems related to optimal control and herding/containment can be faced. The solution to these problems with holonomic agents is reported in the same chapter. In Sec 3.3.2 a different problem is faced: it is considered the possibility of having a limited communication range and this hypothesis is the starting point to analyze the behavior of systems where the connectivity is not fixed but time dependant.

The next step was to find a way to exploit the knowledge about graph theory to control groups of non-holonomic vehicles, in particular vehicles modeled as differential-wheel robots. As the problem has been already partially faced in the last years, we decided to introduce a new constraint to our system: basically, inspired by a range-and-bearing platform developed by DISAL lab at EPFL, we have supposed that robots can exploit only local information, eventually broadcasted in a short communication range, where "local"

means that each robot can sense the others but don't have access to global information. In Sec 4.1, the algorithm used to solve the consensus problem for differential-wheel robots is explained: after defining a decentralized control law, a demonstration of stability is provided in case of fully connected system (i.e.the communication graph is complete). As an example of application of this algorithm, we have performed the formation keeping. Furthermore, as the communication between robots is actuated using infrared, optimization like data broadcasting has been introduced in order to avoid instability due to the presence of environmental obstacles. The result is that the system is still stable also if the communication graph is no more complete but only connected. The goodness of our result has been investigated also with realistic simulations both on Matlab and Webots, where errors have been introduced in sensors reading in order to match with real set up. To conclude, this second part was tested on real robots using four Khepera III robots with range and bearing platform at DISAL lab, EPFL.

In Appendix A RobotiCad, an educational Matlab/Simulink toolbox for robotics has been presented. Even if it is not strictly related with the main work presented in this thesis, it is a 55 thousands line code program that was developed starting in 2005 and was programmed as a parallel activity in order to create a user-friendly toolbox to help students learn robotics basic concepts and to help engineers in fast prototyping. As example, two case studies are provided: the first one deals with the problem of programming a 8-degree-of-freedom robot to weld two objects; the second one exploits functionalities of RobotiCad for prototyping a robot to heal labirintolithiasis in motion-impaired persons.

## Future Developments

Future research activities will concern the possibility of finding new behavioral functions in order to create more complex swarm behaviors using the null-space based approach, in particular will consider the possibility of using only local data instead of using global data in order to achieve complex tasks. Another interesting research direction may be finding a real application field where the null-space behavioral approach can be applied. An example can be having robots able to carry an object: in that case it will be interesting to find a way to coordinate robots in order to move a common load in a point to point transfer task and, in particular, can be interesting to coordinate different subsets of robots in order to have explores, able to find a way to accomplish the assigned task, and carrier, that exploit the information broadcasted by explorers in order to avoid formation breaking or to prevent problems related to local minima.

On the other hand, an interesting direction for future research may be focus on the possibility of solving consensus algorithm using only local sensing to coordinate vehicles modeled as Dubins vehicles or car-like vehicles. In the first case, the algorithm can be

potentially useful to coordinate aerial vehicles like airplanes or Unmanned Aerial Vehicles (AUVs), while in the second case the consensus can be solved to face problems like vehicle routing on highways, where cars can not use full time data from GPS systems.

The topics of this thesis have been published or presented in [45], [81], [31], [32], [10], [79], [80]

# Appendix A

# RobotiCad, a Matlab/Simulink toolbox for robotics

RobotiCad is a user-friendly Matlab/Simulink toolbox for the modeling and simulation of robotic manipulators. With RobotiCad, starting from Denavit-Hartenberg parameters, it is possible to create the kinematic and dynamic models of any serial mechanical structure, together with its 3D graphical model. When a robot is created, it can be exported in a dedicated file that can be loaded in a Simulink scheme and easily interfaced with other block-sets. A robot, then, can be simulated and, eventually, an *AVI* file of the simulation can be obtained. Moreover, a rich collection of Matlab functions properly developed in order to study industrial robots is included in RobotiCad, e.g. functions for trajectory generation, manipulability analysis, control, and so on.

## A.1 Introduction to RobotiCad

Because of the complexity of robotic systems, several simulation tools devoted to robotics have been developed and proposed for solving problems ranging from control to trajectory planning, from design to programming and so on. Some of these tools are oriented to professional/industrial applications, while others are more specifically suitable for educational purposes. Many of them have been implemented for well defined problems and for defined classes of robots only, such as e.g. RoboOp [39], RoboWorks [70], or Easy-Rob [4]. Some of them are stand-alone packages, while others have been created as open source tools, see e.g. GraspIt [65], RoboMosp [43] or Webots [64]. Among the open source packages, several Matlab toolboxes have been developed, such as SimMechanics [6], or the Robotics

Toolbox by Peter Corke [25], to our knowledge the first tool of this type.
The Robotics Toolbox, basically, provides a set of Matlab functions and Simulink blocks for the simulation of the direct and inverse kinematics and of the dynamic model of user-defined robots. Although it can be probably considered as the most popular robotics toolbox, it reveals some limits:

- the Corke's Simulink blocks are related to the current Matlab workspace;
- only a type of trajectory is implemented, that represents the only manner to move a robot;
- a graphical user interface (GUI) is not available, and robots are represented as a collection of segments;
- the inverse kinematic function does not take into account singularity points;
- a robot is simulated without any other object in its environment;
- a robot cannot interact with external objects (e.g. grasped objects).

In particular, the last point is critical if the simulation of a robot within an industrial environment is required. These limitations were the main motivations for the development of RobotiCad, the Matlab/Simulink toolbox described in this chapter.

| **Robotics Toolbox** | **RobotiCad** |
|---|---|
| Text line interface | User friendly 3D dedicated interface |
| Create dedicate object classes | Compatible with Corke's toolbox |
| No workspace object | Several different workspace objects |
| Workspace simulation without objects | Workspace objects included in Simulink model |
| Fifth order polynomial trajectories | Twelve different types of trajectories; dedicated script tool |

Table A.1: Comparison between Robotics Toolbox and RobotiCad.

The main differences between the Robotics Toolbox and RobotiCad are summarized in Table A.1. It's worth to notice that some of the Corke's Matlab functions have been implemented and expanded in RobotiCad. Among them, all the functions allowing the definition of the robot and of the link object classes. This requirement was fundamental as we decided to keep software compatibility between models created in Robotics Toolbox and RobotiCad.

This chapter is organized as follows. In Section A.1 the main features of RobotiCad are introduced (the fundamental Matlab functions, the graphic user interface, the algorithms for the study of kinematics and dynamics, the RobotiCad Simulink Blockset library). An example of control implemented using RobotiCad features for an 8-DOF robot welding two perpendicular intersecting cylinders is discussed in Section A.2. In Section A.3 a fast prototyping example is shown.

As shown in Fig. A.1, the RobotiCad environment is composed of three fundamental modules:

- RobotiCad Matlab Functions.
- RobotiCad GUI.
- RobotiCad Blockset.

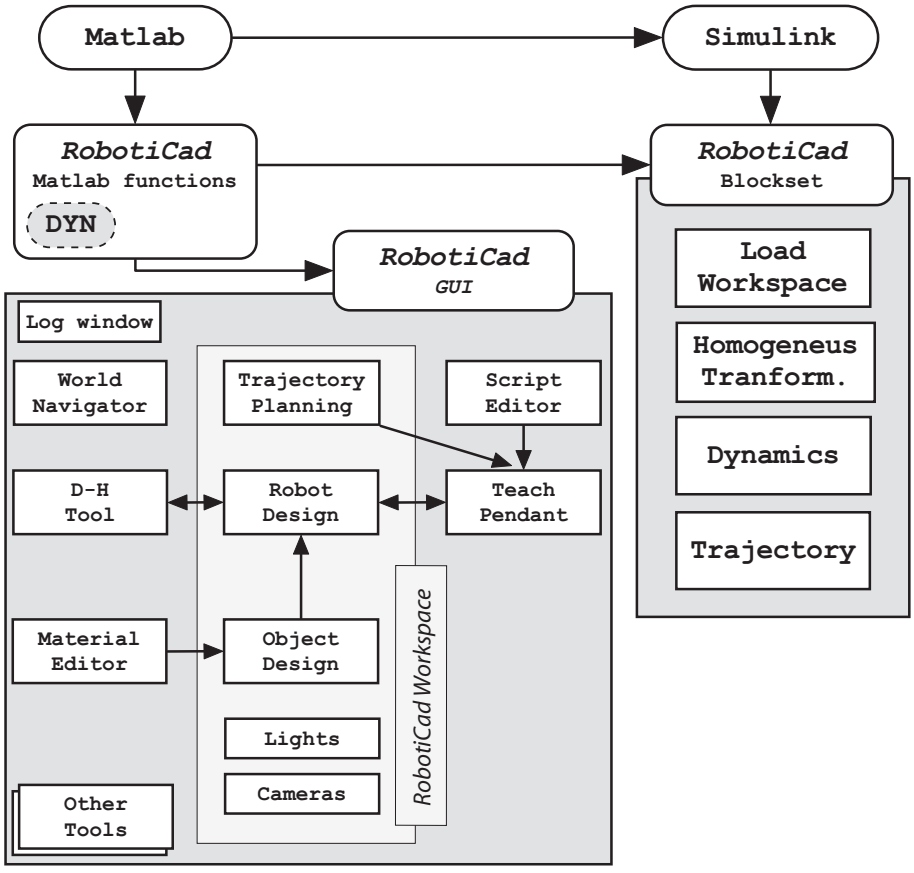



Figure A.1: RobotiCad functional blocks.

### A.1.1 RobotiCad Matlab Functions

This module can be considered the core of RobotiCad. Many functions used by the RobotiCad GUI (see Section A.1.2) and by the RobotiCad Simulink Blockset (see Section A.1.3) are implemented as *m*-functions and can be used at the Matlab prompt. This module allows the user to handle homogeneous transformation and rotation matrices, the creation of robot objects, and the study of its forward and inverse kinematics and dynamics. Moreover, this module allows to study singularity configurations and to handle robot's Jacobian matrices. Although many of these functions are improvements of Corke's Robotics Toolbox functions, the real innovation introduced in this module is the possibility of generating workspace and joint-space trajectories. A special mention concerns the *DYN* function subset: exploiting the Matlab Symbolic Toolbox, it allows the user to create the closed form solution to all the equations that concerns a particular manipulator. Starting from the kinematic and dynamic parameters of a robot, it creates all the matrices needed to analyze and control the robot, in particular it creates:

- the homogeneous-transform matrix for the position and orientation of the end-effector and of all the joints of the robot;
- the matrices involved in the closed-form dynamic equation of the manipulator, expressed as

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + D\dot{q} + g(q)q = \tau \tag{A.1}$$

  where all the matrices are parameterized based on the joint-space coordinates $q$.

As an extra options, this function set allows to create an *.m* files to be included in a Simulink scheme in order to have a more realistic simulations.

### A.1.2 RobotiCad Graphic User Interface

The GUI (Graphic User Interface) is probably the most important module of RobotiCad. By using it, the user can create a workspace, e.g. an industrial working cell, with many robots and objects to interact with. By typing "`>> RobotiCad`" at the Matlab prompt, the toolbox is started. The environment, shown in Fig. A.2, is composed of three main windows, that are:

**Log Window (a):** this window contains a single text field that is automatically updated every time the user executes an action on the workspace or on an object in the scene. By means of the Log Window, an history of the executed actions in the current workspace is then available to the user.

**World-Tree Window (b):** the World Navigator Window presents the hierarchical structure of the objects in the workspace, automatically updated when objects are created or modified. As example, in Fig. A.3 the structure of a Puma 560 robot is shown.

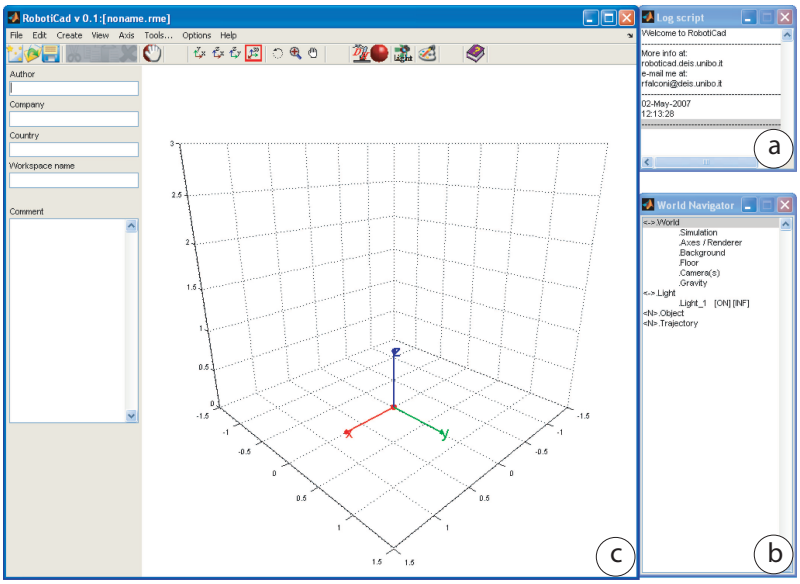



Figure A.2: RobotiCad main windows: a) Log Window. b) World Tree. c) RobotiCad Workspace.

**Workspace Window (c):** robots, workspace trajectories and other objects can be created in this window. The user can also insert lights and cameras to improve the representation of the scene. An absolute reference frame $\mathcal{F}_0 = \{O_0, x_0, y_0, z_0\}$ is present in the center of the new workspace.

In the *RobotiCad Workspace Window* the user can create its workspace with objects, robots, 3D workspace trajectories, lights and cameras.

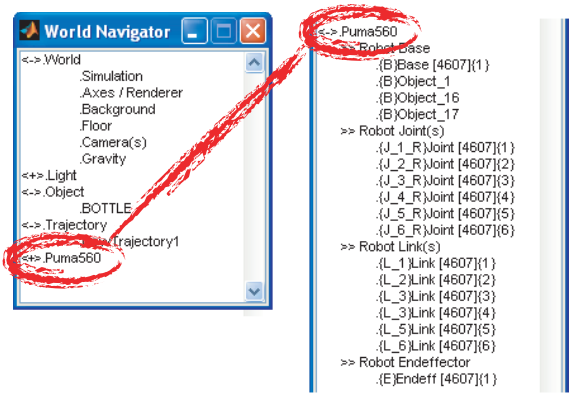



Figure A.3: Example of World Navigator Window: Unimation Puma 560 and its subtrees.

In order to create a new robot, the user can specify the Denavit-Hartenberg (DH) parameters or load a Corke-robot model if saved in a Matlab file. After defining the DH parameters, the user can fix the position and the orientation of the base of the robot base w.r.t. $\mathcal{F}_0$. Once all the parameters have been defined, the robot can be exported in the RobotiCad workspace. As a new robot is created, it is characterized by four categories of objects: *base*, *joint*, *link*, and *endeffector*. Each category may contain several objects, and their dynamic properties (i.e. mass and inertia matrix) can then be specified. As default, each category contains an object with default properties. By using the *World-Tree Window*, a workspace object can be moved to a robot's category and viceversa. Once the robot is created, all its properties can be analyzed (i.e.: Jacobian matrix, manipulability measures, force and velocity ellipsoids). Moreover, it can be programmed by using a virtual teach pendant (for joint-space trajectories), by using the *Script Editor* tool, or by assigning to the manipulator a workspace trajectory previously created. In Fig. A.4 a crane manipulator moving a box along a workspace trajectory is shown. The Trajectory

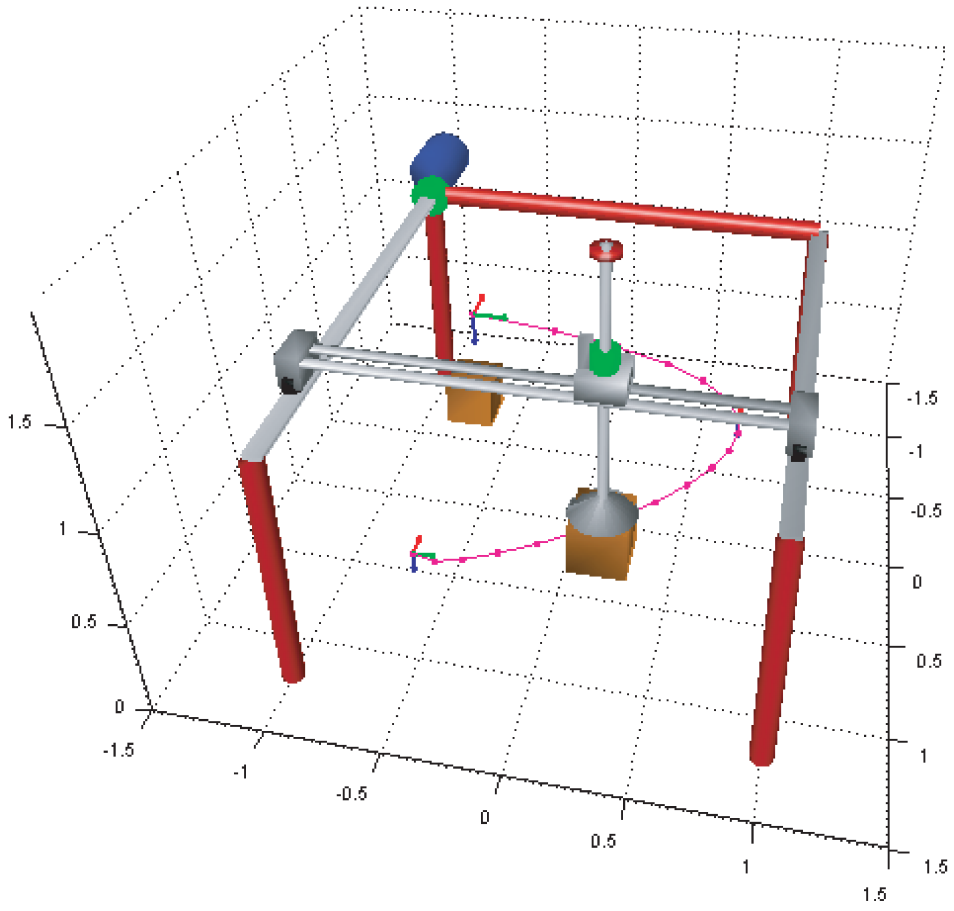



Figure A.4: A Crane moving a box along a 3D workspace trajectory.

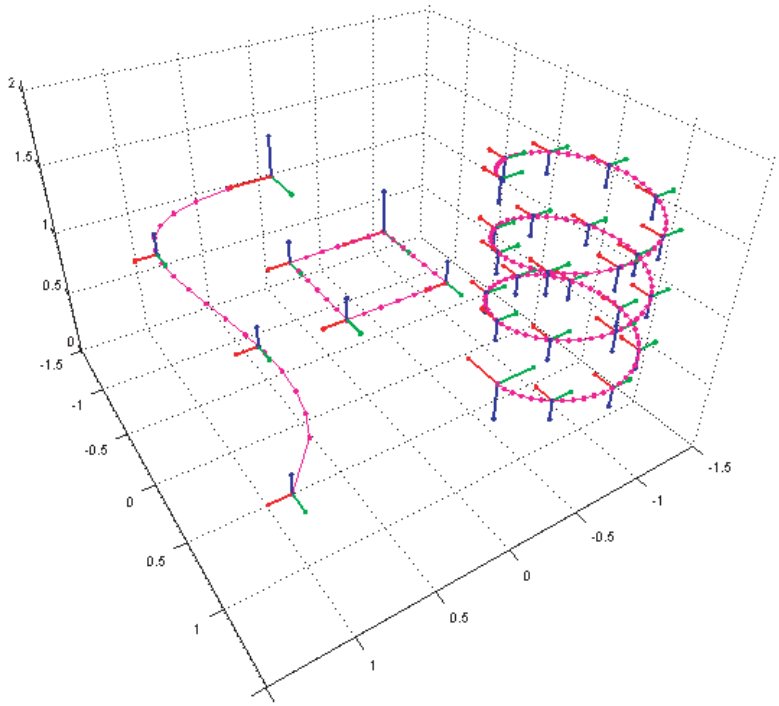



Figure A.5: Examples of workspace trajectories.

Planning module allows to define both Cartesian and Joint-Space trajectories. Many of the trajectories available for joint space motions can be found in [62], [63]. Cartesian trajectories can be specified in three ways:

- by using one of the predefined trajectories implemented in RobotiCad, such as circle and straight line;
- by the definition of a collection of via-points with specified position and orientation;
- by loading a collection of via-points from a text file **.txt* containing the trajectory description.

As example, in Fig. A.5 some workspace trajectories are shown.

For planning Cartesian trajectories in RobotiCad, several interpolation methods have been implemented (the default is based on cubic splines). Moreover, RobotiCad allows the user to combine together trajectories from workspace and joint space in order to achieve complex behaviors. To conclude, each workspace trajectory is associated with a semi-transparent bounding box (that suggests to the user the space covered by it) and a fixed frame positioned in the center of the trajectory. In this manner, scaling, rotation, and translation operations can be applied to the whole trajectory. Independently from the

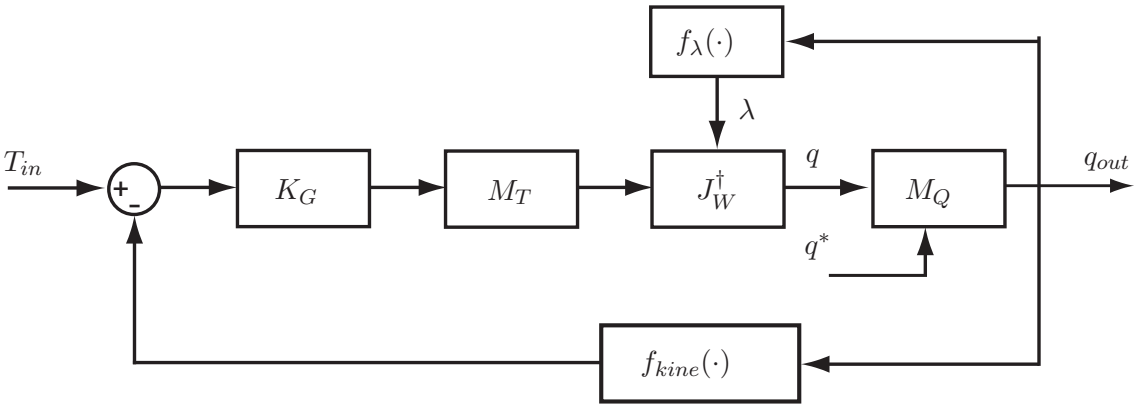



Figure A.6: Scheme of the inverse kinematic algorithm.

method chosen to program the robots, the obtained trajectories can be saved in a file that can be loaded in the RobotiCad GUI and in a Simulink model using a dedicated block (see Section A.1.3). During the execution of a movement, RobotiCad gives also the opportunity of controlling if the moving robot collides with one or more objects in the workspace.

### Kinematic and Dynamic Algorithms

The position and orientation in space of each object is provided by a reference frame rigidly connected with the object itself. Since the DH parameters are used to describe robot configurations, the position and orientation of each object in space results from a proper multiplication of homogeneous matrices. For the $i$-th joint (and related objects), the matrix providing position and orientation in a specified configuration is:

$$\mathbf{B}\ \mathbf{A}_1^2(q_1)\ \ldots\ \mathbf{A}_{i-1}^i(q_i)\ =\ \begin{bmatrix} & \mathbf{R} & & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $\mathbf{B}$ is the homogeneous matrix used to define the position and the orientation of the base of the manipulator w.r.t. $\mathcal{F}_0$, $\mathbf{R}$ is a rotation matrix and $\mathbf{p}$ is a vector.

The *inverse kinematic algorithm* is based on a modified calculus of the pseudo-inverse of the Jacobian matrix (see [48]). The general scheme to compute the robot's joint configuration starting from an homogeneous matrix or from a sextuple of elements like $[X_{pos},\ Y_{pos},\ Z_{pos},\ Roll,\ Pitch,\ Yaw]$ is shown in Fig. A.6, where:

- $T_{in}$ is the homogeneous matrix for the desired endeffector position and orientation, and $q_{out}$ is corresponding joints configuration;
- $K_G$ is a matrix gain that can be set by the user, whose value is critical for the

convergence of the algorithm; the *Inverse Kinematic block* (see Section A.1.3) allows to set the value of $K_G$ (the default value is 1) during simulation;

- $J_W^\dagger$ is the weighted pseudo-inverse of the Jacobian matrix computed as

$$J_W^\dagger = J^T \left(JJ^T - \lambda\right)^{-1},$$

where $J$ is the Jacobian matrix of the manipulator in the current configuration and $\lambda$ is a weight matrix defined in [48];

- $M_T$ and $M_Q$ are two filters customable by the user in order to mask some positions or directions in the workspace ($M_T$) or to lock some joints to a defined value ($M_Q$). In particular, the first filter can be used to improve algorithm performances for under-actuated robots (i.e.: for a planar robot, we can mask $[Z_{pos},\ Pitch,\ Yaw]$); the second filter can be used to select only proper subsets of the joints for the inverse kinematic function, leaving the possibility for the remaining joints to be programmed directly in joint space.

The dynamic algorithm used in RobotiCad is based on the standard Newton-Euler method and consists of a recursive algorithm to define the torques applied to the endeffector (for details, see [86]). Alternatively, the user can specify a *.m* file containing the closed-loop form of the dynamic equations calculated via the *DYN* function subset.

### A.1.3 RobotiCad Blockset

The second main part of the RobotiCad toolbox is a Simulink blockset collection, called RobotiCad Blocks. In Fig. A.7 a snapshot of this library is shown.

These eight fundamental blocks are summarized in the following list. Their properties are explained in the next Section with a case study.

**Workspace and Graphics** This main block, shown in Fig. A.8(a), contains all the blocks that allow the user to interface the model created in the RobotiCad GUI with a Simulink environment. In particular, the *Select Robot* (see Fig. A.8(b)) block allows to choose the robot to work with and, by a simple tick in a checkbox, can automatically create all the blocks that implement the fundamental functions, such as forward and inverse kinematics and dynamics, and so on.

**Trajectory** Contains all the blocks dedicated to the trajectory generation. The user can choose between many kinds of trajectories, or load the **.txt* files created by the RobotiCad GUI containing workspace trajectories. By means of these blocks, the user can combine many different trajectories to create complex motion profiles.

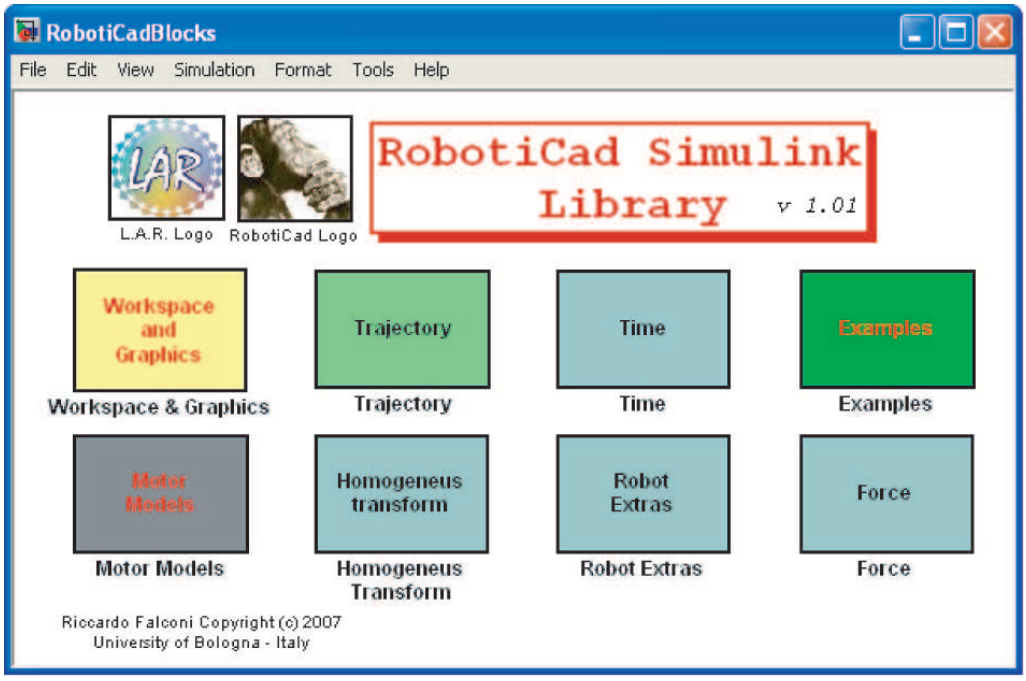
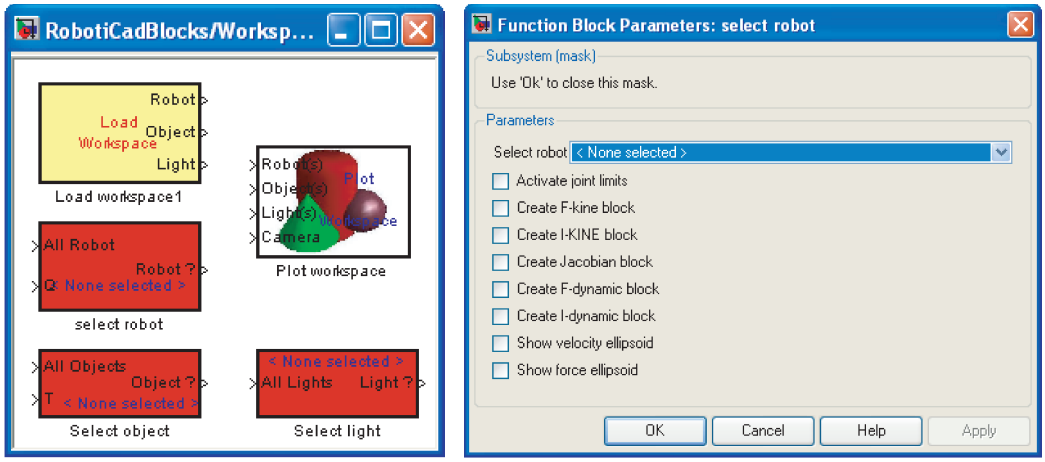



Figure A.7: RobotiCad Library for Simulink.




(a) Workspace and Graphics (b) *Select Robot* mask

Figure A.8: RobotiCad Workspace and Graphics Blocks and mask of the *Select Robot* block.

**Motor Models** Some basic motor models, such as DC motors and so on, are collected here. The user can add new models.

**Homogeneous Transformation** Contains all the blocks to handle homogeneous transformation.

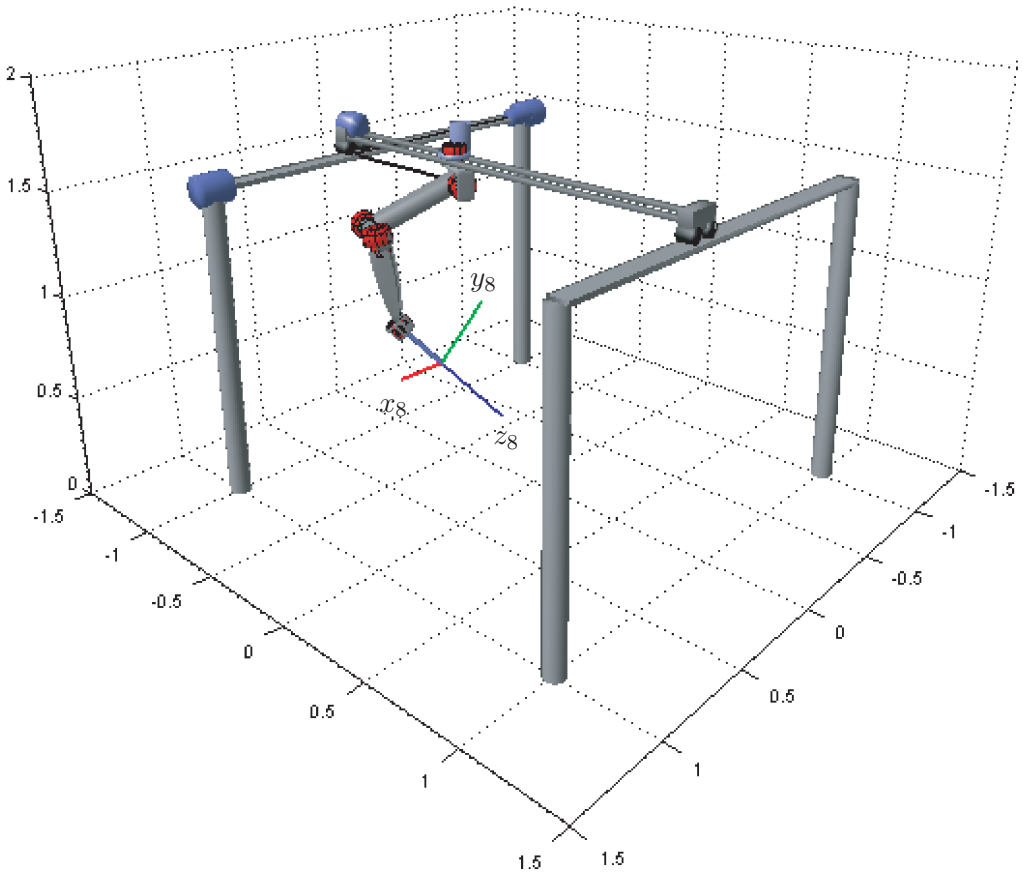



Figure A.9: The case study robot.

**Robot Extras** Once the user selects a robot, the blocks collected here allow the computation of the manipulability values, of the Jacobian matrix, of singularity conditions.

**Force** Contains two blocks to define a force vector and to load the gravity vector.

**Time** Contains two blocks that can be programmed as switches and that can be used to combine trajectories.

**Examples** As indicated by the block name, this block contains some examples of robotic systems.

## A.2 Case Study 1: welding profile

In this Section, in order to show how to create and simulate an industrial robot cell within RobotiCad, a case study is reported. In particular, an 8 DOF robot composed by a 2

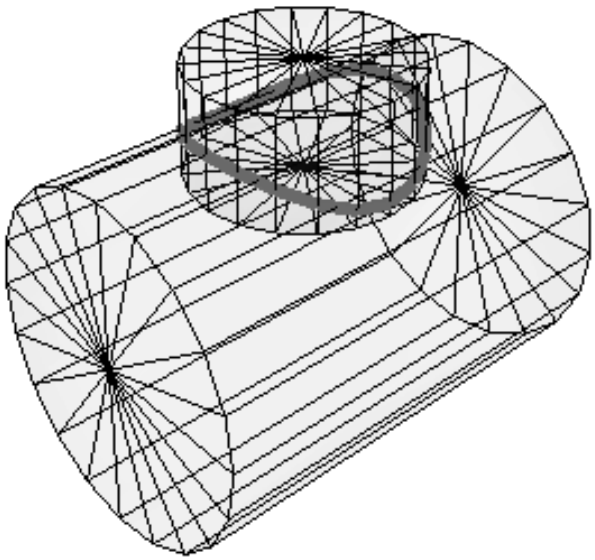


Figure A.10: Two orthogonal cylinders with desired welding profile.

DOF planar arm moving on the $XY$ plane and a Unimation Puma 560 (see Fig. A.9) is considered. The DH parameters are reported in Table A.2. In the simulation, this 8 DOF robot is used to weld two orthogonal and intersecting cylinders. The robot's tool trajectory is specified in the workspace, and the inverse kinematic algorithm is used to obtain the corresponding joints configurations. The workspace trajectory is described analytically and a subset of via-points is used and interpolated for the robot programming. Of course, for this kind of task, it is not necessary to use a manipulator as complex as this one. The dynamic parameters are not reported for the sake of brevity and because they are well known in the literature (see [5]). The position of the robot base in $\mathcal{F}_0$ is $[-1,\ -1,\ 1.5]^T$, while its *Tool* has an offset $d_t = 0.5$ along the $z_8$ endeffector axis.

### A.2.1 Trajectory Definition

The trajectory followed by the robot's tool is shown in Fig. A.10. To obtain the requested path, one can start from considering the locus obtained by intersecting two orthogonal cylinders, with radii $r_1 = 0.4$ and $r_2 = 0.75$ respectively. This locus is described by:

$$p(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} r_1 \cos(t) \\ r_1 \sin(t) \\ \sqrt{r_2^2 - r_1^2 \sin^2(t)} \end{bmatrix}, \tag{A.2}$$

| | $\alpha_i$ | $a$ | $\theta_i$ | $d$ | $R/P$ |
|---|---|---|---|---|---|
| $Joint_1$ | $\frac{\pi}{2}$ | 0 | 0 | $d_1$ | 1 |
| $Joint_2$ | $-\frac{\pi}{2}$ | 0 | $\frac{\pi}{2}$ | $d_2$ | 1 |
| $Joint_3$ | $\frac{\pi}{2}$ | 0 | $\theta_3$ | 0 | 0 |
| $Joint_4$ | 0 | 0.43 | $\theta_4$ | 0 | 0 |
| $Joint_5$ | $-\frac{\pi}{2}$ | 0.02 | $\theta_5$ | 0.15 | 0 |
| $Joint_6$ | $\frac{\pi}{2}$ | 0 | $\theta_6$ | 0.43 | 0 |
| $Joint_7$ | $-\frac{\pi}{2}$ | 0 | $\theta_7$ | 0 | 0 |
| $Joint_8$ | 0 | 0 | $\theta_8$ | 0 | 0 |

Table A.2: DH Welding Robot Parameters.

where $t \in [0 \dots 2\pi]$.

To define the Frenet frame associated to each via-point, $p(t)$ is derived:

$$\begin{aligned} \dot{p}(t) &= \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \end{bmatrix} = \begin{bmatrix} -r_1 \sin(t) \\ r_1 \cos(t) \\ -\frac{r_1^2 \sin(t) \cdot \cos(t)}{\sqrt{r_2^2 - r_1^2 \sin^2(t)}} \end{bmatrix}, \\ \ddot{p}(t) &= \begin{bmatrix} \ddot{x}(t) \\ \ddot{y}(t) \\ \ddot{z}(t) \end{bmatrix} = \begin{bmatrix} -r_1 \cos(t) \\ -r_1 \sin(t) \\ \frac{r_2^2(\sin^2(t) - \cos^2(t)) - r_1^2 \sin^4(t)}{\sin(t)\cos(t)} \end{bmatrix}, \end{aligned} \tag{A.3}$$

By normalizing $p(t)$, $\dot{p}(t)$, $\ddot{p}(t)$, one obtains $\bar{p}(t) = \frac{p(t)}{|p(t)|}$, $\dot{\bar{p}}(t) = \frac{\dot{p}(t)}{|\dot{p}(t)|}$, $\ddot{\bar{p}}(t) = \frac{\ddot{p}(t)}{|\ddot{p}(t)|}$ that represent the unit vectors of the Frenet frame parametrized in $t \in [0 \dots 2\pi]$. Now, by considering the 8 DOF robot, one can see that the robot's *approach* vector and $z$ vector of the Frenet frame differ of $\pi$ radians. To get the via point that describe the trajectory, one can get the Frenet frame for $t = i \cdot \frac{2\pi}{N}$, with $i = 0 \dots N$, and apply a rotation about $z$ of an angle $\theta = \pi$. In our case, the value $N = 20$ has been chosen to get a low position error using cubic spline interpolation (see Table A.3). The resulting workspace trajectory is shown in Fig. A.11.

### A.2.2 Robot Control Scheme

The control scheme used in this example is the well known *PD with gravity compensation*. The overall scheme is reported in Fig. A.12.

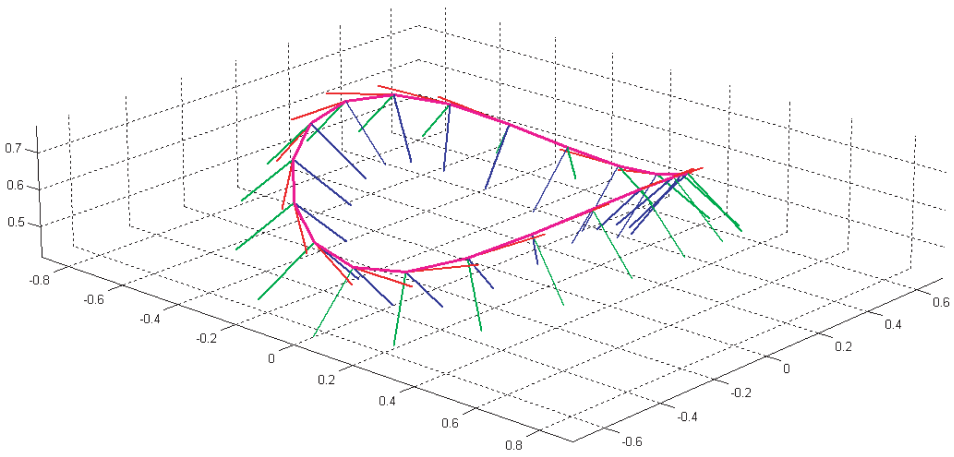



Figure A.11: Resulting trajectory with 20 via-points.

Table A.3: Number of trajectory via points versus approximation error.

| Number of via points | Error % |
|---|---|
| 3 | 2.4039 |
| 5 | 0.0882 |
| 20 | $7.4488 \cdot 10^{-5}$ |
| 100 | $5.7429 \cdot 10^{-13}$ |

In Fig. A.13 the analogous scheme created with the RobotiCad Simulink Library is shown. Note that in this scheme, a *Random Number* generator has been introduced to simulate the fact that the gravity term computed by the control algorithm in general does not match the real value. Moreover, note that the reference signal is created with two different blocks: the first one loads the **.txt* file containing the 20 via points used for the trajectory interpolation, the second one generates a joint space trajectory. In fact, the *Inverse Kinematic Block* is used to get joints trajectory reference for joints $3 \dots 8$, while the first two joints are moved by a circular trajectory specified directly in joint space. This allows to move the Cartesian part of the structure on a circle, while the tool of the robot is always maintained orthogonal to the intersection of the two cylinders.

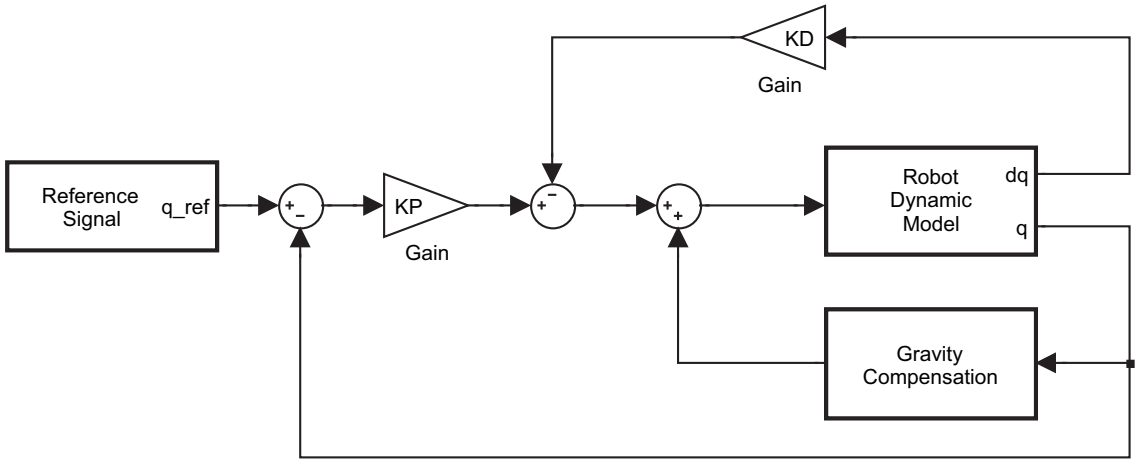



Figure A.12: PD with gravity compensation control scheme.

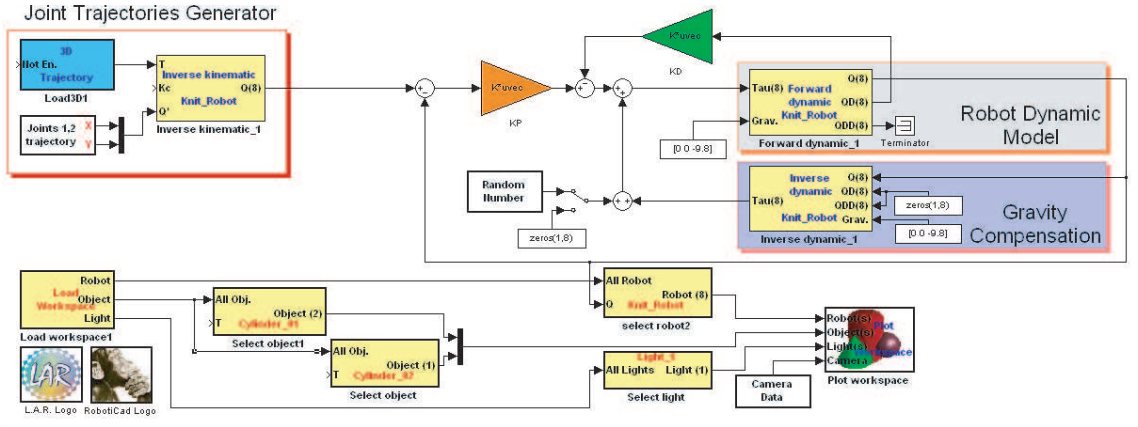
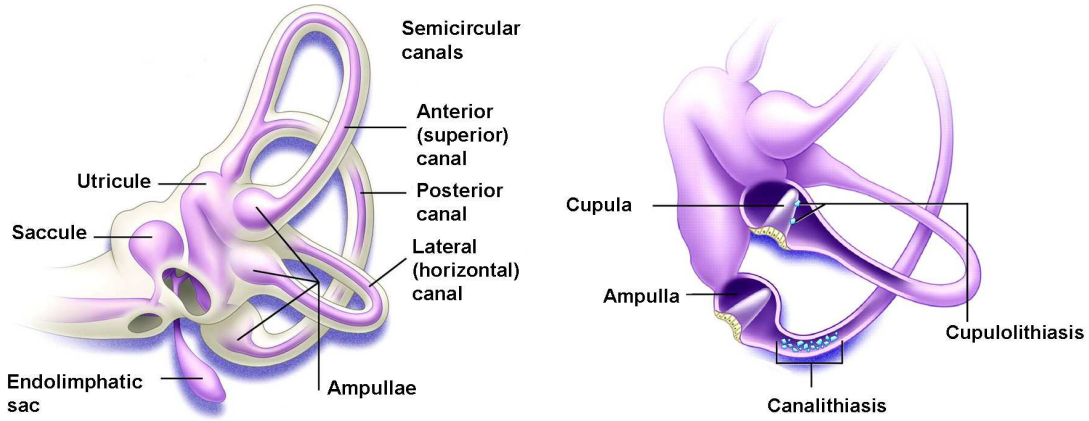



Figure A.13: PD with gravity compensation control scheme implemented with RobotiCad Simulink Library.

## A.3 Case Study 2: a real application in rehabilitation robotics

The first real application that exploits the RobotiCad Matlab Functions has been presented in [10], where the authors deal with the problem of projecting a serial manipulator able to heal motion-impaired patients affected by Benign Paroxymal Positional Vertigo (BPPV) and variants, collectively called vestibular lithiasis, that designate a common disorder caused by a malfunction of the inner ear. These pathologies are connected with the presence of dense particles within the semicircular canals which interfere with the sensing capabilities of angular velocity in the patient, causing nystagmus and vertigo. In




(a) Human vestibular system

(b) BPPV (Canalithiasis) and BPPV variants (Cupulolithiasis)

Fig. A.14(a) the human vestibular system is shown, and in Fig. A.14(b) the BPPV is depicted with variants.

Some of these conditions can be treated by repositioning maneuvers physically done by the doctor that moves the head of the patient along different poses in space. Despite the fact that the treatment shows a success rate up to 80-90%, the failure rate remains highly significant and it is proven that precision repeatability and unlimited 360° manoeuvrability can improve diagnostic and treatment potential for overcoming this kind of vertigo. Some example of manipulators able to move and orient a person following predefined trajectory already exists, such as the RoboCoaster from Kuka Roboticker GmbH [82] or the Omniax Positioning System designed and manufactured at the Portland Otologic Clinic by Eng. William Scott following Dr. J. M. Epley specifications [69], a robot specifically designed for the diagnosis and treatment of the BPPV that presents limited maneuverability. From these considerations, it follows the need to create a manipulator with high mobility able to move a patient replying known maneuvers and, eventually, with a workspace as small as possible in order to be used in an hospital environment. The kinematic design proposed in [10] is based on a simplified version of the task-based design technique exploited in [22] [23] [97] [51] to find an optimal structure-configuration to satisfy all the kinematic constraints for BPPV maneuvers. In particular, the optimization process make use of a progressive method witch meets consecutive constraints and progressive optimized solutions.

The overall evaluation procedure is depicted in Fig. A.14. Given a set of tasks we state that a reachability constraint (RC) must be satisfied for all task points. The RC constraint is said to be satisfied if there exists a solution of the inverse kinematic problem (IK) for the particular task presenting a positional and orientation error norm within a certain threshold and within a certain number of iterations. As the structure of the manipulator is not yet defined and the serial chain can assume a very high number of configurations,

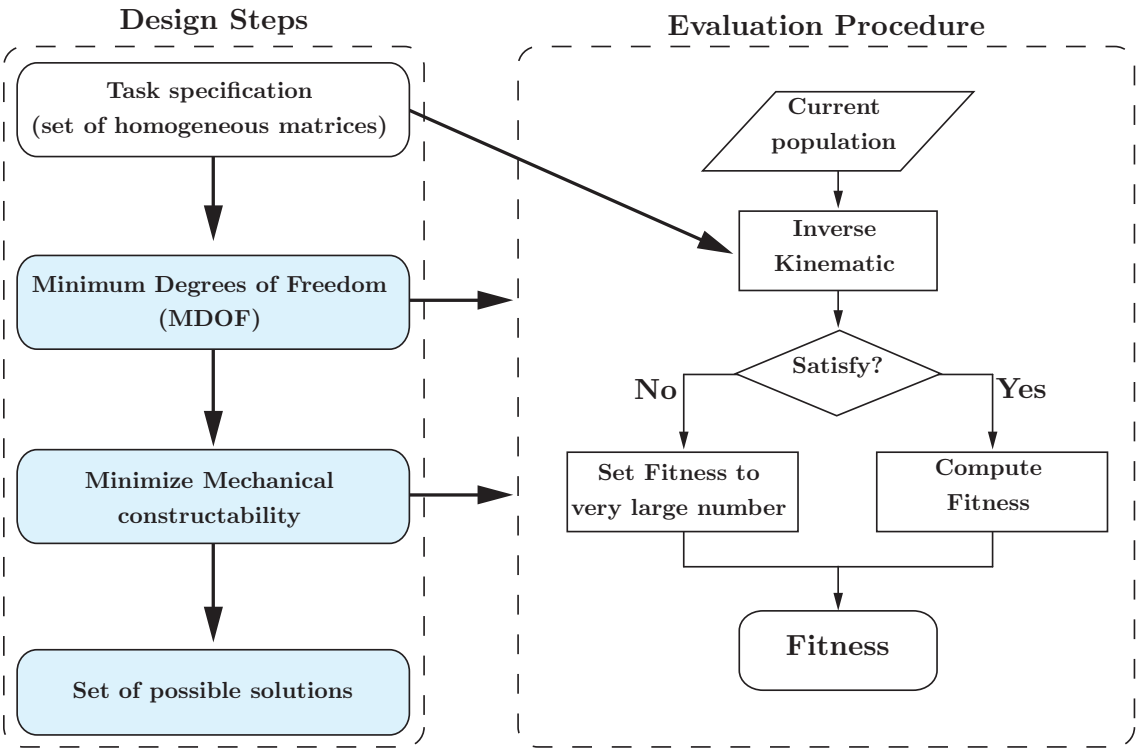



Figure A.14: Design steps and evaluation procedure.

a numerical method to solve the IK problem is used; reference is made to the singularity robust IK method proposed in [49]. If the RC is not satisfied the fitness value is set to a very large number. The design steps are briefly explained below:

A) ***Minimized degrees-of-freedom approach(MDOF)***: In the first design stage the fitness value is simply the number of manipulator's d.o.f. regardless the fact of joint being revolute or prismatic. IK is computed for every task and, if the RC is not satisfied, the fitness value is set to a very large number and successive tasks are not evaluated. Prior to IK calculation, which is the most time consuming procedure, it is verified that:

$$R_i = r_i - \sum_{j=0}^{n} l_j < 0 \text{ for } i = 1, ..., N_{TOT} \tag{A.4}$$

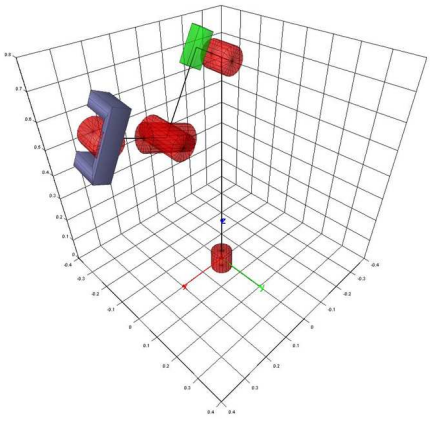
where $N_{TOT}$ is the total number of tasks, $r_i$ is the Euclidian distance of i-th task point from robot base and $l_j$ is the $j$-th link length. If the $j$-th joint is revolute, $l_j$ is a design parameter with a maximum allowed value set to $L_{\max}$. If $j$-th joint is prismatic, $l_j$ is set to $2L_{\max}$. After IK, $i$-th RC is considered not satisfied if:

$$\sqrt{a_j^2 + (\Delta d_{j_{max}}^2)} > 2L_{\max} \tag{A.5}$$

where $\Delta d_{j_{\max}} = |d_{j_{\max}} - d_{j_{\min}}|$ (i.e. during motion the prismatic joint has traveled a distance greater than $2L_{max}$, its initial length being set to $a_j$.) Result of this procedure

| D-H matrix and Link Type best solution | | | | |
|---|---|---|---|---|
| $\alpha$ | $a$ | $\vartheta$ | $d$ | $R/P$ |
| $\frac{\pi}{2}$ | 0,00 | $\vartheta_1$ | 1,20 | R |
| $\frac{3}{2}\pi$ | 0,10 | $\vartheta_2$ | 0 | R |
| $\frac{3}{2}\pi$ | 0,00 | $\pi$ | $d_3$ | P |
| $\frac{3}{2}\pi$ | 0,00 | $\vartheta_4$ | 0 | R |
| $\frac{3}{2}\pi$ | 0,00 | $\vartheta_5$ | 0,20 | R |
| $\pi$ | 0,00 | $\vartheta_6$ | 0 | R |

(a) Denavith-Hartenberg parameters



(b) Best solution link representation.

Figure A.15: Best found solution.

is the minimum number of d.o.f. necessary to perform task specifications; basically there exists a possible $n$-DOF structure defined by the individual $\overline{\mathbf{X}}$ capable to perform every pose.

B) ***Mechanical constructability minimization.*** Fitness function is set to:

$$F(X) = \sum_{j=0}^{n} l_j \tag{A.6}$$

where:

$$\begin{cases} l_j = \sqrt{a_j^2 + d_j^2} & \text{if j} - \text{th joint is revolute} \\ l_j = \sqrt{a_j^2 + (\Delta d_{j_{max}}^2)} & \text{ifj} - \text{th joint is prismatic.} \end{cases} \tag{A.7}$$

At this design stage fixed joints are not allowed. The aim of this step is the minimization of total link length and therefore of total robot's mass.

The algorithm is tested with a population size of 50 individuals - 200 generations. The IK solver and the obtained results visualization make use of the functionalities provided by RobotiCad. In table A.15(a) are reported the kinematic parameters of the best found solution, schematized in Fig. A.15(b). In Fig. A.16 a possible mechanical realization of the manipulator is depicted, and in Fig. A.17 two positions of task-executing are shown.

## A.4 Conclusions and future work

In this chapter RobotiCad, a new robotic toolbox for Matlab/Simulink, has been presented. It is based on a rich collection of Matlab functions, and can be used as a tool for educational purposes or industry robot prototyping and control.

The user-friendly interface allows to model a robot by specifying its Denavit-Hartenberg matrix and its physical properties, and to add objects to the environment by creating or

loading them from a library, and more manipulators can be used at the same time. Moreover, the manipulator can be programmed in different ways and simulated by means of new Simulink blocks. An *AVI* file can be created for each simulation session. The RobotiCad toolbox is available on the web at:

http://www.roboticad.deis.unibo.it/

Here, several examples and a library of robots (such as SCARA, Unimation Puma 560, Stanford Arm and so on) can be found.

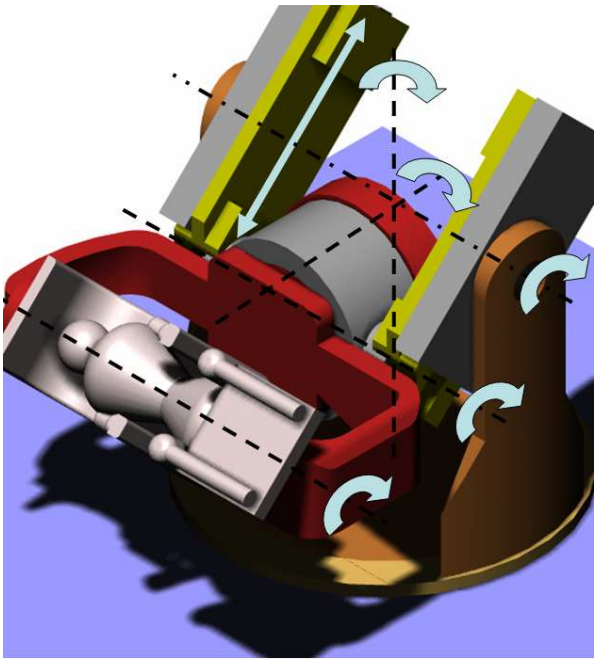
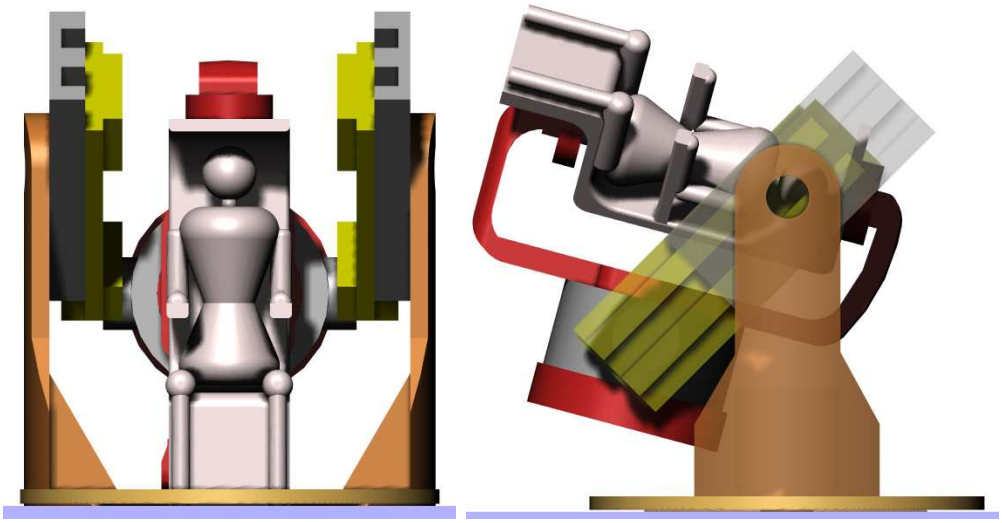


Figure A.16: Possible Mechanical design of best found solution



(a) Rest position (b) Rotation along right AC axis

Figure A.17: Manipulator performing task

# List of Figures

# Bibliography


[1] J. Carvalho E. Paiva J. Ramos A. Elfes, M. Bergerman and S. Bueno. Air-ground robotic ensembles for cooperative applications: concepts and preliminary results. In *Proceedings of the International Conference on Field and Service Robotics*, pages 75–80, 1999.

[2] A. S. Morse A. Jadbabaie, J. Lin. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 48(6):988–1001, June 2003.

[3] M. Mesbahi A. Rahmani. On the controlled agreement problem. In *American Control Conference*, pages 1376–1381, June 2006.

[4] S. Anton. Easy-Rob, 3D Robot Simulation Tool. *www.easy-bot.com*, 2005.

[5] B. Armstrong, O. Khatib, and J. Burdick. The explicit dynamic model and inertial parameters of the puma 560 arm. volume 3, pages 63–73, 1986.

[6] R. J. Babuska. Matlab Design Environment for Robotic Manipulators. 2005.

[7] T. C. Balch, T. & Arkin. Behavior-based formation control for multirobots teams. *IEEE Transaction on Robotics and Automation*, 14(6):926–939, 1998.

[8] Arkin B.C. *Behavior-based robotics*. The MIT Press, 1998.

[9] J. A. Benediktsson and P. H. Swain. Consensus theoretic classification methods. *IEEE Transactions on Systems, Man and Cybernetics*, 22(4):688704, July/August 1992.

[10] Vassura G. Modugno G.C. Berselli G., Falconi R. Task based kinematic design of a serial robot for the treatment of vestibular lithiasis. In *IEEE 10th International Conference on Rehabilitation Robotics (ICORR), 2007*, April 2007.

[11] D. P. Bertsekas and J. Tsitsiklis. *Parallell and Distributed Computation*. Prentice-Hall, 1989.

[12] Fan Zhang Bin Lei, Wenfeng Li. Stable flocking algorithm for multi-robot systems formation control. In *IEEE Congress on Evolutionary Computation*, pages 1544–1549, June 2008.

[13] Bradley E. Bishop. On the use of redundant manipulator techniques for control of platoons of cooperating robotic vehicles. *IEEE Transaction on System, Man, and Cybermetics*, 33:608–615, September 2003.

[14] Koren Y Borenstein J. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1179–1187, 1989.

[15] V. Borkar and P. Varaiya. Asymptotic agreement in distributed estimation. *IEEE Transactions on Automatic Control*, 27(3):650655, June 1982.

[16] Daniel J. Stilwell Bradley E. Bishop. On the application of redundant manipulator technique to the control of platoon of autonomous vehicles. In *IEEE Internation Conference on Control Applications*, pages 823–828, September 2001.

[17] V. Braitenberg. *Vehicles: Experiments in synthetic psychology*. The MIT Press, 1984.

[18] Gonzales L. Koestler A. Nguyen M. Petitt J. Braunl T., Boeing A. The autonomous underwater vehicle initiative - project mako. In *IEEE Conference on Robotics, Automation and Mechatronics*, pages 446– 451, December 2004.

[19] John B. Moore Brian D. O. Anderson. *Optimal Control: Linear Quadratic Methods*. Prentice-Hall, 1990.

[20] Hollis R. Butler Z., Rizzi A. Cooperative coverage of rectilinear environments. In *IEEE International Conference on Robotics and Automation*, pages 2722 – 2727, 2000.

[21] G. Royle C. Godsil. *Algebraic Graph Theory*. Springer, 2001.

[22] Ramstein E. Chedmail P. Robot mechanisms synthesis and genetic algorithms. In *1996 IEEE Robotics and Automation Conference*, pages 3466–3471, 1996.

[23] Burdick J. Chen M. Determining task optimal robot assembly configurations. In *IEEE International Conference on Robotics and Automation*, pages 132–137, 1995.

[24] F. R. K. Chung. *Spectral Graph Theory*. American Mathematic Society, 1997.

[25] Peter I. Corke. A robotics toolbox for Matlab©. *IEEE Robotics and Automation Magazine*, 3:24–32, 1996.

[26] Sachs Cvetkovic, Doob. *Spectra of Graphs: Theory and Applications*. Vch Verlagsgesellschaft Mbh, 1998.

[27] K. J. Kyriakopoulos D. V. Dimarogonas. On the state agreement problem for multiple unicycles with varying communication links. In *45th IEEE Conference on Decision and Control*, pages 4283–4288, December 2006.

[28] Caleb A. Sylvester Daniel J. Stilwell, Bradley E. Bishop. Redundant manipulator techniques for partially decentralized path planning and control of a platoon of autonomous vehicles. *IEEE Transaction on System, Man, and Cybermetics*, 35:842–848, August 2005.

[29] M. H. DeGroot. Reaching a consensus. *Journal of American Statistical Association*, 345(69):118121, 1974.

[30] Kostas J. Kyriakopoulos Dimos V. Dimarogonas. On the rendezvous problem for multiple nonholonomic agents. *IEEE Transactions on Automatic Control*, 52(5):916–922, 2007.

[31] Melchiorri C. Falconi R. A decentralized control algorithm for swarm behavior and obstacle avoidance in unknown environments. In *IFAC Navigation, Guidance and Control of Underwater Vehicles (NGCUV08)*, April 2008.

[32] Melchiorri C. Falconi R. Decentralized control for robot teams in unknown environments. In *40th International Symposium on Robotics*, March 2009.

[33] A. Buffa M. Ji G. Ferrari-Trecate, M. Edgerstedt. Laplacian sheep: A hybrid, stop-go policy for leader-based containment control. In *9th Hybrid Systems: Computational and Control Conference*, pages 212–226, February 2006.

[34] Stefano Chiaverini Gianluca Antonelli. Kinematic control of a platoon of autonomous vehicles. In *IEEE International Conference on Robotics and Automation*, pages 1285–1292, December 2003.

[35] Stefano Chiaverini Gianluca Antonelli. Fault tolerant kinematic control of platoons of autonomous vehicles. In *IEEE International Conference on Robotics and Automation*, pages 3313–3318, April 26-May 1 2004.

[36] Stefano Chiaverini Gianluca Antonelli. Kinematic control of platoons of autonomous vehicles. *IEEE Transactions on Robotics*, 22:12851292, December 2006.

[37] Stefano Chiaverini Gianluca Antonelli, Filippo Arrichiello. The null-space-based behavioral control for mobile robots. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 1257–1262, July 2005.

[38] Michael Bonani Francesco Mondada Marco Dorigo Stefano Nolfi Gianluca Baldassarre, Vito Trianni. Self-organized coordinated motion in groups of physically connected robots. *IEEE Transaction on System, Man and Cybernetics*, 37:224–239, February 2007.

[39] R. Gourdeau. Roboop, a Robotics Object Oriented Package in c++. *http://www.cours.polymtl.ca/roboop/*, 2006.

[40] George J. Pappas Herbert G. Tanner and Vijay Kumar. Leader-to-formation stability. *IEEE Transactions on Robotics and Automation*, 20(3):443– 455, June 2004.

[41] Frank A. Dale D. Vian J. How J.P., Bethke B. Real-time indoor autonomous vehicle test environment. *Control Systems Magazine, IEEE*, 28(2):51–64, April 2008.

[42] H. Jacobsen. A generic architecture for hybrid intelligent systems. In *IEEE World Congress on Computational Intelligence.*, pages 709–714, 1998.

[43] A. Jaramillo-Boter, A. Matta-Gomez, J. F. Correa-Caicedo, and W. Perea-Castro. Robomosp. *IEEE Robotics and Automation Magazine*, 6:63–73, 2007.

[44] Vijay Kumar Jaydev P. Desai, Jim Ostrowski. Controlling formations of multiple mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 2864–2869, 1998.

[45] Cédric Favre Riccardo Falconi Jim Pugh, Xavier Raemy and Alcherio Martinoli. A fast on-board relative positioning module for multi-robot systems. *IEEE Transactions on Mechatronics*, To be appear.

[46] R.V.; Peters T.M.; Jing Ren; McIsaac, K.A.; Patel. A potential field model using generalized sigmoid functions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 37(2):477–484, april 2007.

[47] Cao Qixin; Huang Yanwen; Zhou Jingliang;. An evolutionary artificial potential field algorithm for dynamic path planning of mobile robot. pages 3331 – 3336, 2006.

[48] L. Kelmar and P. K. Khosla. Automatic generation of kinematics for a reconfigurable modular manipulator system. 1988.

[49] Khosla P.K. Kelmar L. Automatic generation of forward and inverse kinematics for a reconfigurable modular manipulator system. *Journal of Robotic Systems*, 7(4):599–619, 1990.

[50] BEKEY G. KHOSHNEVIS B. Centralized sensing and control of multiple mobile robots. In *Computers and industrial engineering*, pages 503–506, 1998.

[51] Khosla P. Kim J.O. Design of space shuttle tile servicing robot: An application of task based kinematic design. In *IEEE International Conference on Robotics and Automation*, pages 867–874, 1993.

[52] Min Gyu Park; Jae Hyun Jeon; Min Cheol Lee;. Obstacle avoidance for mobile robots using artificial potential field approach with simulated annealing. volume 3, pages 1530 – 1535, 2001.

[53] F. L. Lewis. *Optimal Control*. John Wiley and Sons, 1986.

[54] James F. Keller R. Vijay Kumar Camillo J. Taylor Luiz Chaimowicz, Ben Grocholsky. Experiments in multirobot air-ground coordination. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, pages 4053–4058, 2004.

[55] M. Edgerstedt M. Ji. Distributed formation control while preserving connectedness. In *IEEE Conference on Decision and Control*, pages 5962–5967, December 2006.

[56] M. Edgerstedt M. Ji. Distributed coordination control of multiagent system while preserving connectedness. *IEEE Transactions on Robotics*, 23(4):693–703, August 2007.

[57] M. Edgerstedt M. Ji, A. Muhammad. Connectedness preserving distibuted coordination control over dynamic graphs. In *American Control Conference*, pages 93–98, July 2005.

[58] M. Edgerstedt M. Ji, A. Muhammad. Leader-based multi-agent coordination: Controllability and optimal control. In *American Control Conference*, pages 1358–1363, June 2006.

[59] M. Mataric. Designing emergent behaviors: From local interactions to collective intelligence. In *International Conference on Simulation of Adaptive Behavior, Proceedings of the*, 1992.

[60] Maja J Matari'c. Issues and approaches in the design of collective autonomous agents. *Robotics and Autonomous Systems*, 16:321–331, December 1995.

[61] J. McLurkin. Stupid robot tricks: Behavior-based distributed algorithm library for programming swarms of robots. Master's thesis, Massachusetts Institute of Technology, Cambridge, 2004.

[62] C. Melchiorri. *Traiettorie per Azionamenti Elettrici*. Esculapio Ed, Bologna, 2000.

[63] Biagiotti L. Melchiorri C. *Trajectory planning for automatic machines and robots*. Springer, 2009.

[64] O. Michel. Webots: Professional mobile robot simulation. *Journal of Advanced Robotic Systems*, 1:39–42, 2004.

[65] Andrew T. Miller and Peter K. Allen. Graspit!: A Versatile Simulator for Robotic Grasping. *IEEE Robotic and Automation Magazine*, 2004.

[66] P. Milutinovic, D. Lima. Modeling and optimal centralized control of a large-size robotic population. *Robotics, IEEE Transactions on*, 22(6):1280–1285, December 2006.

[67] Francesco Mondada, Edoardo Franzi, and Paolo Ienne. Mobile robot miniaturisation: A tool for investigation in control algorithms. In *Proceedings of the 3rd International Symposium on Experimental Robotics*, 1993.

[68] Yoshikawa T. Nakamura Y., Hanafusa H. Task-priority based redundancy control of robot manipulators. *The International Journal of Robotics Research*, 6(2):3–15, 1987.

[69] Epley J.M. Nakayama M. Bppv and variants: Improved treatment results with automated, nystagmus-based repositioning. *Ornithology-Head and Neck Surgery*, 133:107–112, 2005.

[70] Newtonium©. Roboworks, a Tool for Real Time Interactive 3D Modelling and Animation with Distributed Simulation. *http://www.newtonium.com/*, 2005.

[71] K. Chang D. Ruspini R. Holmberg A. Casal A. Baader O. Khatib, K. Yokoi. Force strategies for cooperative tasks in multiple mobile manipulation systems. In *In Proceedings International Symposium of Robotics Research*, 1996.

[72] Ade B. Ogunsanya A. Graph theory in intra-urban traffic flow estimation. *GeoJournal*, 12(3):334–336, 1986.

[73] L.E. Parker. Alliance: An architecture for fault-tolerant multi-robot cooperation. In *Robotics and Automation, IEEE Transactions on*, pages 220–240, 1998.

[74] A. (2006) Pugh, J. Martinoli. Relative localization and communication module for small-scale multi-robot systems. In *2006 IEEE International Conference on Robotics and Automation*, pages 188–193, May 2006.

[75] Brooks R. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2:14 – 23, 1986.

[76] Kalantar S.& Zimmer U. R. Motion planning for small formations of autonomous vehicles navigating on gradient fields. In *International Symposium on Underwater Technology, Proceedings of the*, pages 512–519, 2007.

[77] Wei Ren. Consensus strategies for cooperative control of vehicle formations. *Control Theory and Applications*, 1:505–512, 2007.

[78] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics*, pages 25–34, 1987.

[79] Alessandro Macchelli Luigi Biagiotti Riccardo Falconi, Claudio Melchiorri. Roboticad: a matlab toolbox for robot manipulators. In *8th International IFAC Symposium on Robot Control (Syroco)*, pages 9111–9116, September 2006.

[80] Claudio Melchiorri Riccardo Falconi. Roboticad, an educational tool for robotics. In *17th IFAC World Congress*, pages 9111–9116, July 2007.

[81] Jim Pugh Alcherio Martinoli Riccardo Falconi, Swen Gowal. Graph-based distributed control for non-holonomic vehicles engaged in a reconfiguration task using local positioning information. In *Robocomm 2009*, March 2009.

[82] Kuka Industrial Robots. Six-dimensional fun the world first passenger-carrying robot. IAAPA Orlando.

[83] Wyman C. Gini M. Rybski P., Stoeter S. A cooperative multi-robot approach to the mapping and exploration of mars. In *AAAI-97 Proceedings.*, pages 798–799, 1997.

[84] A. Mallet S. Lacroix, I.-K. Jung and R. Chatila. Towards cooperative air/ground robotics: issues related to environment modeling. In *10th International Conference on Advanced Robotics*, 2001.

[85] Wildermuth D. Schneider F.E. A potential field based approach to multi robot formation navigation. volume 1, pages 680–685, 2003.

[86] L. Sciavicco and B. Siciliano. *Modelling and Control of Robot Manipulators*. McGraw-Hill, New York, 1996.

[87] Bruno Siciliano. Kinematic control of redundant robot manipulators: A tutorial. *Journal of Intelligent Robotic Systems*, 3:201212, 1990.

[88] Reid Simmons, Sanjiv Singh, David Hershberger, Josue Ramos, and Trey Smith. First results in the coordination of heterogeneous robots for large-scale assembly. In *In Proceedings of the International Symposium on Experimental Robotics (ISER)*, 2000.

[89] Magnus Edgerstedt Staffan Bjorkenstam, Meng Ji and Clyde Martin. Leader-basedmulti-agent coordination through hybrid optimal control. In *Forty-Fourth Annual Allerton Conference*, pages 1352–1357, September 2006.

[90] Gaurav S. Sukhatme, James F. Montgomery, and Richard T. Vaughan. Experiments with cooperative aerial-ground robots. In *Robot Teams: From Diversity to Polymorphism. AK Peters*, pages 345–367, 2001.

[91] L. Matthey S. Indra T. Lochmatter, X.Raemy and A. Martinoli. A comparison of casting and spiraling algorithms for odor source localization in laminar flow. In *2008 IEEE International Conference on Robotics and Automation*, pages 1138–1143, May 2008.

[92] H. G. Tanner. On the controllability of nearest neighbor inteconnections. In *43rd IEEE Conference on Decision and Control*, pages 2467–2472, December 2004.

[93] Hugli H. Tieche F., Facchinetti C. Multi-layered hybrid architecture to solve complex tasks of autonomous mobile robots. *International Journal on Artificial Intelligence Tools*, pages 6–8, 1998.

[94] Bruckstein A. Wagner I., Lindenbaum M. Mac vs pc: Determinism and randomness as complementary approaches to robotic exploration of continuous unknown domains. *The International Journal of Robotics Research*, 19(1):12–31, 2000.

[95] Bourgeous W. Sorensen N. Yang Quan Chen Wei Ren, Haiyang Chao. Experimental implementation and validation of consensus algorithms on a mobile actuator and sensor network platform. In *ISIC. IEEE International Conference on Systems, Man and Cybernetics, 2007*, pages 171–176, October 2007.

[96] Bourgeous W. Sorensen N. Yang Quan Chen Wei Ren, Haiyang Chao. Experimental validation of consensus algorithms for multivehicle cooperative control. *IEEE Transaction on Control System Technology*, 16(4):745–752, 2008.

[97] Chen I.M. Yang G. Task-based optimization of modular robot configurations mdof approach. *Mech Mach Theory*, 35(4):517–540, 2000.

# Thanks

The author kindly thanks his colleagues that helped him during these 3 years: Gianni Borghesan, that provided a coach where he can sleep and that will be soon defeated in Guitar Hero (Let's Rock!), Alessandro Macchelli, that has always a ready-to-cook "fiorentina" for him, and is always able to find out the worst side of everything, Gianluca Palli, that never talk, and Luigi Biagiotti, that talk too much, Raffaella Carloni, that after two years is still not able to summarize any concept in less than hundreds words, Andrea Paoli, Matteo Sartini, Luca Gentili (the three CASY musketeers) and the other CASY fellows, Roberto Naldi, that soon or later will find a way to make his pot flying, and prof Lorenzo Marconi for finding the time to help me. Besides, the author thanks all the people on the other side of the Alps that helped him during his working period in Lausanne: prof. Alcherio Martinoli, Sven Gowal, Jim Pugh, Amanda Prorok, and all the DISAL fellows. It was a honor to work with you, and he hopes to see you soon (for a Guinness, of course!) The author also thanks his tutor, Prof. Claudio Melchiorri. The last thanks go to Claudia, and to the author's family, for their unconditioned support.