Alma Mater Studiorum – Universitá di Bologna

Dottorato di ricerca in

Automatica e Ricerca Operativa

XXV Ciclo

Settore concorsuale di afferenza: 09/G1
Settore scientifico disciplinare: ING-INF/04

Titolo tesi

Coordination and Control of Autonomous Mobile Robots Swarms by using Particle Swarm Optimization Algorithm and Consensus Theory

Presentata da
Raffaele Grandi

Coordinatore di dottorato
Andrea Lodi

Relatore
Claudio Melchiorri

Esame finale anno 2013

Abstract

This thesis presents some different techniques designed to drive a swarm of robots in an a-priori unknown environment in order to move the group from a starting area to a final one avoiding obstacles. The presented techniques are based on two different theories used alone or in combination: Swarm Intelligence (SI) and Graph Theory. Both theories are based on the study of interactions between different entities (also called agents or units) in Multi-Agent Systems (MAS). The first one belongs to the Artificial Intelligence context and the second one to the Distributed Systems context. These theories, each one from its own point of view, exploit the emergent behaviour that comes from the interactive work of the entities, in order to achieve a common goal. The features of flexibility and adaptability of the swarm have been exploited with the aim to overcome and to minimize difficulties and problems that can affect one or more units of the group, having minimal impact to the whole group and to the common main target.

Another aim of this work is to show the importance of the information shared between the units of the group, such as the communication topology, because it helps to maintain the environmental information, detected by each single agent, updated among the swarm.

Swarm Intelligence has been applied to the presented technique, through the Particle Swarm Optimization algorithm (PSO), taking advantage of its features as a navigation system. The Graph Theory has been applied by exploiting Consensus and the application of the *agreement protocol* with the aim to maintain the units in a desired and controlled formation. This approach has been followed in order to conserve the power of PSO and to control part of its random behaviour with a distributed control algorithm like Consensus.

"All truths are easy to understand when they are revealed.
The hard part is finding"

(Galileo Galilei)

"Logic will get you from A to B.
Imagination will take you everywhere"

(Albert Einstein)

To my family and ...

To all the people that
in unexpected ways and often unconsciously,
have contributed to this work.

Contents

1

Aims of the Thesis

The aim of this thesis is to investigate the possibilities of integration between two different theories: Swarm Intelligence (SI) and Graph Theory. Both theories are based on the study of interactions between different entities (also called agents or units) in Multi-Agent Systems (MAS). The first one belongs to the Artificial Intelligence context and the second one to the Distributed Systems context. These theories, each one from its own point of view, exploit the emergent behaviour that comes from the interactive working of the entities, in order to achieve a common goal. The features of flexibility and adaptability of the swarm have been exploited with the aim to overcome and to minimize difficulties and problems that can affect one or more units of the group, having minimal impact to the whole group and to the common main target.
Another aim of this work is to show the importance of the information shared between the units of the group, such as the communication topology, because it helps to maintain the environmental information, detected by each single agent, updated among the swarm.
Swarm Intelligence has been applied to the presented technique, through the Particle Swarm Optimization algorithm (PSO), taking advantage of its features as navigation system. The Graph Theory has been applied by exploiting Consensus and the application of the *agreement protocol* with the aim to maintain the units in a desired and controlled formation. This approach has been followed in order to conserve the power of PSO and to control part of its random behaviour with a distributed control algorithm like Consensus. In conclusion the final goal of this thesis is to lay the basis of a distributed control algorithm for a Mobile Robotic Transportation Platform, which is

based on the interaction between many robots, that collaborating are able to transport heavy loads, which are difficult or impossible to be managed by a single robot. Exploiting the benefit of being composed by many simpler coordinated robots, the platform has a huge amount of applicative scenarios and it may be transported and reassembled everywhere. Moreover the possibility to substitute damaged units or to add one or more of them in a real time fashion is very important. This thesis shows the benefit of involving not only terrestrial robots but also underwater vehicles and maybe in the future even aerial ones.

2

Introduction

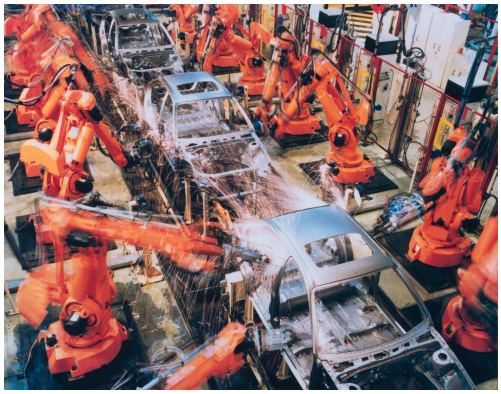
A brief introduction to the world of robotics with no claim of completeness is dutiful, in order to frame the reader in the vast field which inspired this work. After a brief excursus on the birth of robotics and the related fields of development which are currently most active, we have moved from considering robotics as something related to a single unit, to the frontier of robotics, today consisting in groups of robots that are adequately coordinated to achieve a common goal. There are many multi-robot coordination algorithms, but in this work we have considered only two of them, the Particle Swarm Optimization, that comes from the field of Swarm and Artificial Intelligence and the Theory of Consensus, which exploits the power of Graph Theory and which is applied in the field of Distributed Controls.

2.1 A Brief Overview on Robotics

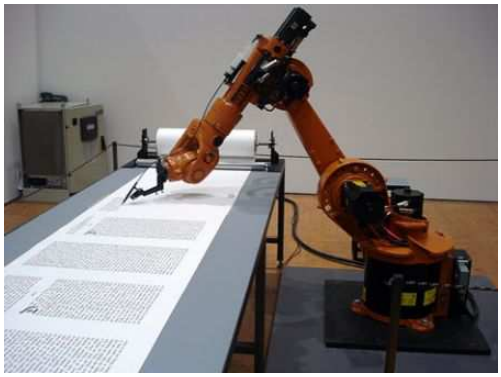
Nowadays the concept of robotics is no longer considered only as industrial robotics, where robots are used in production lines, but also as a complex system that interacts autonomously with the environment and man, on many levels and with various methodologies.

2.1.1 Evolution of Robotics

The concept of robotics was born in the very moment when man thought to give some heavy works to a man-made machine, an automaton. The word *robot* comes from the



(a) An example of assembly-line where a chain of ABB robots produces automobiles.



(b) A manlike-arm robot, produced by KUKA, is used as amanuensis.

Figure 2.1: Example of industrial robots.

Czech word *robota* that means literally **heavy work**[1]. Although we immediately tend to think about humanoid robots for performing these tasks, the first robots used with profit to make heavy works have been industrial robots, designed and created not just to emulate man's full-body, but only his arm.

Robot's science has developed a lot in that direction, mainly for two reasons: the first was certainly due to industrial and commercial interests,the second one was due to the fact that the first technological components used to create robots were not enough sophisticated and therefore it was possible to create robots only in this fashion. Modern industrial robots are largely used for moving relatively heavy loads in a restrict and controlled workspace and the 90% of them have anthropomorphic arm shape. This type of robot generally has 5-6 degrees of freedom[2] and it is used in industries to assemble parts of vehicles (see Fig. 2.1a), or for painting, welding, cleaning mechanical parts and handling loads. All these works need great precision and repeatability and modern commercial robots embed the state of art of these features: both control algorithms and mechanical structures are well known, studied and tested. One of this high precision works is represented in Fig. 2.1b, where a anthropomorphic robot is used to produce copies of The Holy Bible with a charming calligraphy.

[1] From Wikipedia "The word robot was introduced to the public by the Czech inter-war writer Karel apek in his play R.U.R. (Rossum's Universal Robots), published in 1920"

[2] In mechanics, the degrees of freedom (DOF) of a mechanical system are the number of independent parameters that uniquely define its configuration

However, in the last twenty years, other needs have grown up. The necessity of deeper and more complex human-robot interactions forced roboticists to study a new concept of robot, that could be used out of the secure and automatised working cell of an industrial environment. Fortunately, the research in electronics has developed powerful and low cost electronic components, with a very high integration and strong miniaturisation, allowing scientists to develop something different from the classical industrial robot and going back to the original target of robotics: a closer emulation of human abilities.

New knowledge and scientific approach have been developed in this new science, not only from the mechanical construction's point of view but even from the robot's behaviours one, that has been studied in Artificial Intelligence. When a robot moves out from its structured working cell environment to go to work in an unfamiliar, highly variable and crowded environment, a lot of interesting scenarios have open in a large number of situations, for example safe human-robot interaction, or object recognition algorithms used for obstacle avoidance, object manipulation, multi-robot cooperation. In the real environment, the ability to recover information from the on-board sensors is fundamental. Sensors are distributed on the whole robot's body to measure distances from objects or the position of the robots, to give the robots the image of the environment and to permit to the algorithms to make the right movements and take the right decisions. More or less as in human body. It's easy to understand that an intelligent algorithm mounted on a mobile robot is very different from the one mounted on a classical industrial robot. The first one must have characteristics of adaptability and self organisation that the second one does not require. From a conceptual point of view, it is possible to identify two macro-blocks in which the new disciplines that are revolving around *alternative* robotics are divided:

- the first one tries to extend or supplement human skills and human activities using robots to fill the gap (see Fig. 2.2a);
- the second one tries to study autonomous and self-organised behavioural algorithms, often by borrowing concepts that belong to biological life and it applies them to individual robots or group (e.g in Fig. 2.2b).

The biomedical field can be taken such as a representative scenario of the first category. Both from patient's and doctor's point of views important aspects have been

(a) A *Luke Arm* robotic prosthesis.

(b) An example of mobile robot swarm.

Figure 2.2: Two important aspects of Robotics.

developed: most advanced prosthetic limbs could help in human disability or exoskeletons and robots could help in rehabilitation. Robots can help the surgeon in making operations less invasive and more precise, eliminating the normal human hand tremors, all supported by technologies that seek to improve the force feedback and increase the presence of the operator, in short words *haptic interfaces*.

With regards to the second category, there are robots that try to solve problems of exploration in unknown and/or humans inaccessible environments, taking decisions as individual or as a group, along with other robots also performing the same operative task. Between the two categories, often, there isn't a clear division and cross-contamination has given rise to new research areas or has strengthened existing ones providing benefits. To face this type of challenge it seems to be necessary a human-like intelligence.

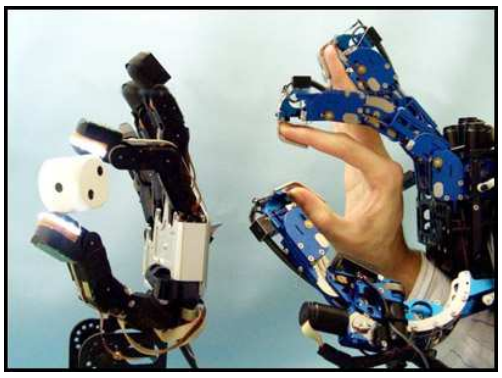
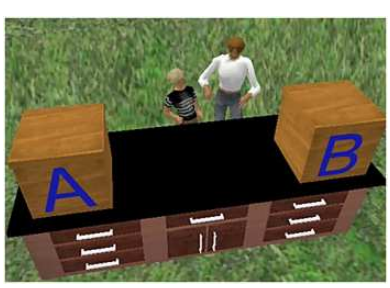


Figure 2.3: An haptic glove useful to remotely control a proper robotic hand.

Even though the objective of developing a robot with a human-like reasoning ability still remains in the science fiction imagination (see Fig. 2.4), some steps are being made.

An example could be a result in the Artificial Intelligence (AI) research field reported in 2008 from the Rensselaer Artificial Intelligence and Reasoning (RAIR) Lab-

Figure 2.4: The main character of "I robot", the film inspired by the related Isaac Asimow's book.



(a) **(b)**

Figure 2.5: The virtual child Eddie (2.5a) and the chess-playing computer Deep Blue (2.5b) are two remarkable examples of Artificial Intelligence applications.

oratory, where a software (Eddie) that seems to have a capacity of reasoning similar to a four years-old child has been developed. By interacting in the simulated environment "Second Life" with other human-driven characters, Eddie is able to provide the correct answer of some "False Belief problems", an answer that a child is able to give only from age four. Another significant example coming from Chess is "Deep Blue", a chess-playing computer developed by IBM (see Fig. 2.5b). On May 11, 1997, the machine, with human intervention in games, won the second six-game match against world champion Garry Kasparov by two wins to one with three draws. It has represented a crucial event toward the development of a real "intelligent machine". More recently, at Thomas J. Watson Research Center under the supervision of David Fer-

rucci it has been developed in IBM's DeepQA project a machine called Watson able to answer questions on the quiz show *Jeopardy!* understanding human natural language. In 2011, Watson competed on Jeopardy against former winners Brad Rutter, and Ken Jennings. Watson received the first prize of 1 million. Later Watson has been tested as medical assistant with high score results. On February 2013, IBM announced that Watson's first commercial application would be in management decisions in lung cancer treatment at Memorial SloanKettering Cancer Center in conjunction with health insurance company WellPoint. On IBM project's site it is possible to read:

> "Watson is a Question Answering (QA) computing system built by IBM that describes it as an application of advanced Natural Language Processing, Information Retrieval, Knowledge Representation and Reasoning, and Machine Learning technologies to the field of open domain question answering which is built on IBM's DeepQA technology for hypothesis generation, massive evidence gathering, analysis, and scoring."

These examples show how it is possible to develop powerful and intelligent algorithm but in any case quite close to their specific field of application and in any case they are limited to the physical constraints of the computer that calculates their program.

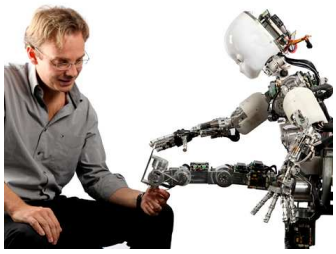
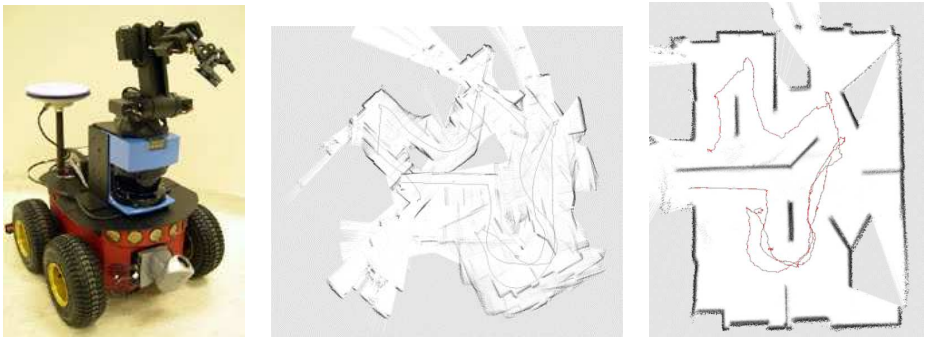


Figure 2.6: An exemplary of the humanoid platform IIT-iCub.

To ensure that the *ghost in the shell* may be connected with what exists in the outside environment and may produce an appropriate interaction, it is necessary to move towards robotics (e.g.Fig. 2.6). The challenge to human intelligence, which has always been the object of many research fronts, taking the path of evolution from the bottom to the top, which means to start from the interaction with the environment.

2.1.2 Intelligent Robots

Giving the definition of what an intelligent robot actually is, it is more difficult than you may think, especially for the reason that it has not been possible to provide a clear definition of *Intelligence* yet. Usually when we talk about intelligent robots, we talk about machines that can be able to perform a limited task with a specific



(a) (b) (c)

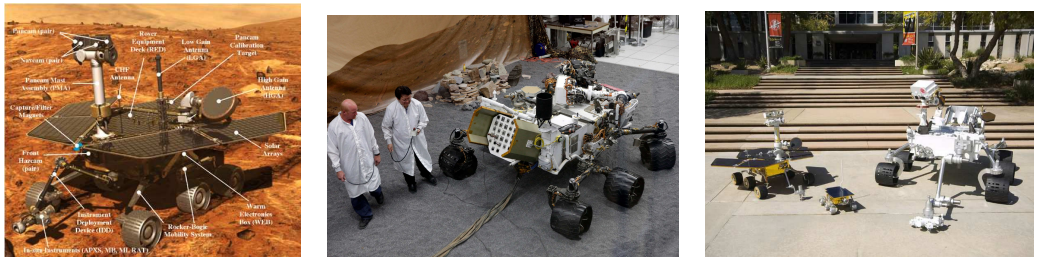
Figure 2.7: The 4 wheeled mobile platform Pioneer 3-AT, endowed with laser-scanner, GPS device, video-camera, robotic griper, ultrasonic sensors, is one of the best platforms used to study mobile robotics. By exploiting the laser scanner it is possible to build an environmental map (2.7b) which needs a post elaboration in order to be usefully exploited (2.7c).

grade of intelligence and autonomy. Mobile robots have many locomotion devices, someone can even fly but today when we speak about mobile robots we speak about a simple rudimentary machine if compared to human skill. Although, these machines are excellent platforms for developing different types of algorithms: behavioural, learning, adaptive and exploratory, we can speak about biologically inspired robotics, cognitive robotics, collective robots, autonomous mobile robots and many other disciplines that are involved in this field, in relation to their main research focus. One of the most complete working platforms that is usually adopted to study theories and algorithms, it's a mobile robot with two or four wheels (see an example in Fig. 2.7a) that can interact with the environment by using a wide range of sensors: infrared sensors, location sensors, on board cameras and laser tracking systems. The information provided on the surrounding environment, allows robot's algorithm to build any map of the route and then to plan paths in a previously unknown environment. Through a laser scanner it is possible to retrieve the characteristics of the surrounding environment and then through a post elaboration of recorded data, to reconstruct the map of the explored area.

The presence of on-board cameras helps the robot to recognize objects or other features of the environment. Sometimes, to complete the robot's equipment, it is also installed a GPS sensor, allowing path tracking with very good approximation, especially

Figure 2.8: Stanley, the Stanford University Car that won the DARPA Grand Challenge in 2005 edition. It has completed the entire path by covering 240 Km in 6:54 hours.

when the environment to be explored includes many miles of desert like DARPA Grand Challenge. It was a car race competition organized by the U.S. Defence Department in 2004 and 2005, where participants were only driver-less full autonomous vehicles with the objective to complete an off-road path of 240 km across part of the Mojave Desert within a limited time. Each vehicle was endowed by a full-set of sensors with laser scanner, hi-resolution cameras, GPS and so on, and clusters of computers stored in their luggage box. As an example the Stanford University's vehicle, that won the 2005 edition, is depicted in Fig. 2.8. The exploration of unknown environments, and the related development of an appropriate capacity to process information autonomously, have always had a big support from all the researches that in some way are involved in space exploration. The inability to directly manage the situation by man forced researchers to develop autonomous intelligent programs as agents. Even staying within mobile robotics, it is easy to move in any other adjacent area, space exploration is an interdisciplinary field where a variety of knowledge converges whit no comparison to other areas. An example of the deep usage of the previously described techniques, could be the autonomous robots called mars-rover sent on Mars to analyse the soil for life search (see Fig. 2.9). The scientists have worked on developing intelligent algorithms that permit to rovers an almost total operational autonomy in task solving. A brief radio contact with Earth to download acquired information and receive new tasks has been the only connection of the robot with man. Moving the attention from space to Earth's surface, it is impossible to ignore one of the most agile and dexterous robot on the Earth: Big Dog. Built by Boston Dynamics, under the sponsorship of DARPA, the robot is able to perfectly emulate a mule (despite of the name) with



(a) (b) (c)

Figure 2.9: Mars rover is another well know example of full autonomous mobile robot. Different types of rovers with different types of tasks and features have been sent to the Martian soil. The dimensions of robots have grown up during the different missions: from Spirit (2.9a) sent on Mars in 2006 to Curiosity (2.9b) sent in 2012, the complexity of tasks, the types and the number of environmental analysis scheduled have need more on-board instruments and more autonomy in term of task managing with respect to the earlier mission, which was performed by the the "little" Sojourner sent on Mars in 1997 (2.9c in the middle).

the capacity to carry very heavy loads. The platform is continuously updated (see Fig. 2.10a and Fig. 2.10b). The robot has both autonomous capacity and a remote radio-controller device. It has been developed to give support to military operations in the most various scenarios. The studies conducted by Boston Dynamics on the legs' motion and locomotion have produced even a biped version of legged robot named *Petman* (See Fig. 2.10c). The movement of these robot is so natural that watching it is very impressive. The applications of this kind of robot are quite obvious. Besides research centers, government agencies and more other institutions it is possible to study robotics and robot applications within the dinner room of our houses or in some little laboratories of every schools and faculties through the Lego Mindstorm Kit (see Sec. 7.3). Lego has developed a robotic kit that permits us to build from scratch a real mobile robot as the one depicted in Fig. 2.11. The application of this robotic kit (e.g. the one in the Sec. 8.4) depends only by users' imagination and it is possible to build a huge quantity of different electro-mechanic devices, from the simpler to the most sophisticated ones. The attention of researchers is not focused only on a single powerful robot and in the study of robotics some interesting questions arise, for example, why don't we use two or three robots instead of one? And why not a swarm of robots? What is the task that can be brought to accomplishment only with the coordination of

(a) (b) (c)

Figure 2.10: The evolution of the Boston Dynamics' Big Dog (from left to right) and the science-fiction realization of a robotic soldier, Petman (2.10c), show the deeply interest of researchers in the field of robotics.

Figure 2.11: A differential wheeled mobile robot built with Lego Mindstorm NXT Kit.

the whole swarm? How it is possible to coordinate so many robots all together? What kind of communication topology is needed? All these questions and many other ones suggest a new research path toward all that concerns the control and coordination of groups of robots.

2.1.3 Groups of Robots

In all the examples shown before, it has always been taken in consideration only individual robots but a lot of recent studies are directly involved in the developing of groups of robots, because a swarm of collaborative robots, each one observing the problem from a different point of view, is able to solve more difficult tasks than one robot by itself. The strength of a swarm can be found in the sharing of information. Pieces of information are sent by a robot to a nearby robot and so on, connecting in this way all

(a) (b)

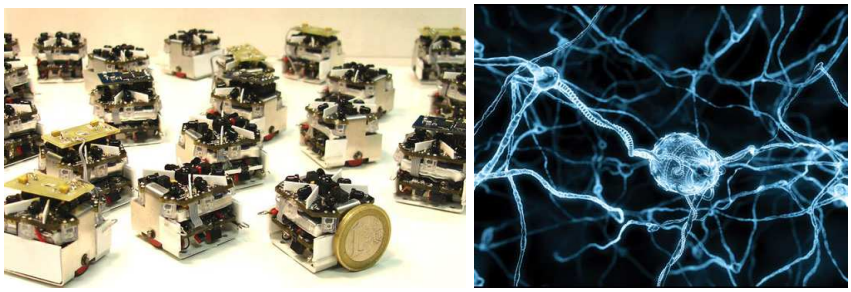
Figure 2.12: Some robots of the Swarm-Bots Project developed at Université libre de Bruxelles (ULB).

the robots of the swarm in a synergistic network. For example, each robot used for the exploration of unknown environment could explore by itself a little part of the search space and at the same time could share information with neighbours. In this fashion the exploration task is achieved after a shorter time than using a single robot. From a general point of view the decomposition of the problem in less difficult sub-problems gives the possibility to a simpler robot to positively contribute to the global solution.

In many cases a practical problem admits solutions, only when the robots work together as a group. An example could be the one depicted in Fig. 2.12b, where a robot must pass upon a rift of the terrain larger than itself. Other examples can be an obstacle to avoid or an object too heavy to be transported by a single robot. I all these problems the group makes the difference. Mobile robots could have any sort of dimensions, from the ones as large as a car (see Fig. 2.8) to the ones as little as a 2 euro coin (e.g. Fig. 2.13a) or less (Fig. 2.16b). In many research centers around the world roboticists are still studying nano-scale robots (e.g. [1, 2]) with the aim to drive a robot swarm inside human body and treat various diseases, from virus infections to cancer (see Fig. 2.13b).

2.2 Mobile Robot Systems

The main difference between a group of robots and a Mobile Robot System (MRS) is the control algorithm. In that case the group of robots operates as a single entity with



(a) (b)

Figure 2.13: Alice (2.13a), developed at EPFL, is one of the most used mini-robots in the world. A very little area is large enough for the mobile robotics techniques applications that involve this robot model, because its dimensions are 2x2x2 cm. However the frontier of the robot dimensions' reduction process is in the nano-scale robotics. A futuristic vision of the nano-bots technology as substitution of neurons 2.13b may be representative of the future.

the same target. The basic principle behind this new approach to 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. Some typical example can be found in the sea, on the ground and in the air, and more evolved animals can collaborate to perform more complex social behaviours (see Fig. 2.14)

A Mobile Robot System is generally composed by homogeneous units on which the importance of a single unit is negligible and it can be substituted without affecting the global task. This highlights the intrinsic robustness of a MRS but it is not only a feature of the swarm. More depends on the algorithm's control structure. It is possible to identify four typologies of control structure:

- **centralized**: there is a unique supervisor that receives data from all the robots connected and calculates the related motion for each one. The advantage of this approach is to have a single control point that computes and collects the information for the whole swarm. Each robot could be very simple and completely remotely controlled. This could also be a drawback. The supervisor node must be powerful enough for all the robots. Scalability is not possible and the task's complexity is directly related to the power of the super-node. Moreover if the super-node encounters a fault, all the system is compromised. (e.g. in [3, 4]);

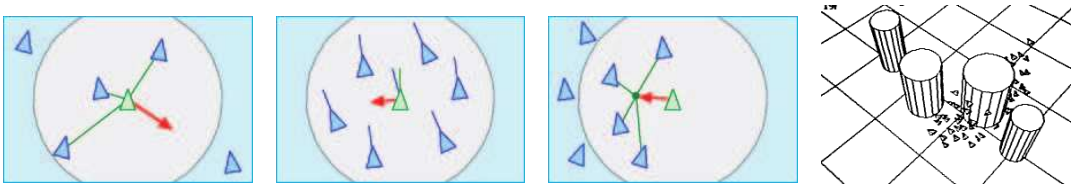
(a) (b) (c) (d)

Figure 2.14: Schools of fishes, flocks of birds, termite mounds and hunting wolves are clear examples of *complex behaviour* where the rules at the basis of the coordinated behaviours make the difference.

- **cecentralized** : each robot computes the data by itself and operates on the basis of the local information, self-generated or received by team-mates. Shared information are fundamental, units are generally homogeneous. The architecture has all the good properties coming from the swarm structure (cf. Chp. 3). However the complexity of the task is limited to the structure of the control algorithm because designing a completely decentralized task is not simple (e.g. in [5]). The *Swarm-Bots* project depicted in the figure-set 2.12 is an example;

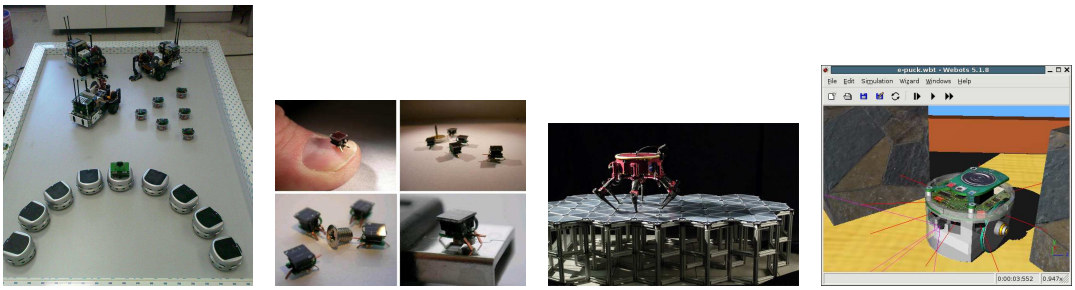
- **hierarchical**: this technique is directly inspired from military command protocols. The architecture is composed by leader and followers robots distributed on different command priority levels. Robots that are leaders in a lower level become followers in the upper level and so on. The advantage comes from the division of the global task in simpler sub-tasks. The architecture is scalable but the drawbacks are similar as in the centralized method, indeed the recovering from failures of the command leaders is difficult;

- **hybrid**: generally this kind of approach tries to combine the advantages coming from centralized and decentralized methods. Some robots cover the role of leaders by assigning tasks and swarm resources. Followers are limited to the accomplishment of the given tasks by using the given resources. This technique often embeds leaders re-election procedures in case of failures.



(a) Separation **(b)** Alignment **(c)** Cohesion **(d)** Flocking boids

Figure 2.15: Reynolds' rules representation.



(a) **(b)** **(c)** **(d)**

Figure 2.16: Different typologies of robots, from real to simulated ones, can be used to study mobile robotics in many research and application fields.

The first experiments on real robotic swarms began in the earlier '90s. Roboticists studied animals' interactions in order to find basic rules applicable to the robot's context, by following the Reynolds example [6], the first remarkable example of this kind of approach, depicted in the figure-set 2.15.

Reynolds' rules are basically three: *Separation*, *Cohesion* and *Alignment*. The first one defines the minimal distance between boids in order to avoid their crowding. The second one defines the maximal distance for the opposite purpose. The third one instead defines the boids' flocking direction that is related to the average heading of local boids. These studies have grown from '90s [7, 8, 9] in order to become the modern Swarm Intelligence theory (cf. Chp. 3) and exploit groups of real robots to study the emergent behaviours [5] Different types of robots have been made to study and emulate natural swarms in all fields: terrestrial, underwater and aerial. Not only real robots have been exploited but also the simulated ones (see figure-set 2.16)

Another technique pointed out by researchers to control MRS, in addition to the Swarm Intelligence, is based on graph control theory. This fully decentralized approach, initially used on groups of massless-point agents [10] and later on real robots

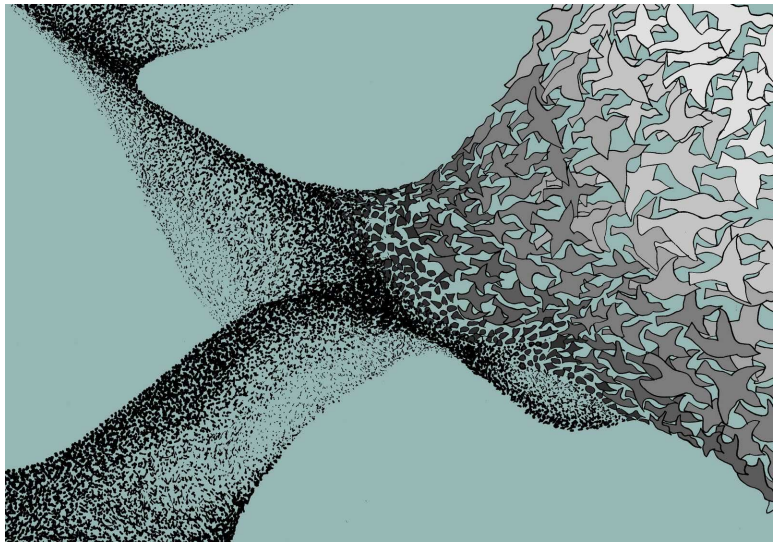
[11, 12, 13], is strictly based on the exploitation of the basic matrices that describe a graph, like Incidence matrix and Laplacian matrix. The basic idea is to use the information exchanged between edge-connected robots in order to maintain a coordination on some state variables i.e. mutual positions or motion velocities. This approach called Consensus is based on the agreement protocol. For more information see [14]

2.3 Outline

A large part of the themes just introduced has been investigated and realized during the making of this thesis. Indeed in the following chapters the control techniques designed and their applications are described, as well as both theoretical and practical tools for their realization. This thesis is organized as follows: a brief introduction of Swarm Intelligence and Particle Swarm Optimization (PSO) is done in Chp. 3, while distributed control techniques respectively designed to drive a group of ground mobile robots and a group of underwater vehicle, by exploiting algorithms based on modified versions of the PSO, are described in Chp. 4 and Chp. 5. Moreover a hybrid control technique based on both PSO and Consenus approach has been described in Chp. 6. Educational Robotics and the *Unibot Mobile Robot* have been introduced in Chp. 7 while the related *Unibot Mobile Laboratory* has been presented in Chp. 8. The simulators on which all the techniques have been tested, are briefly described in Chp. 9. Final conclusions and future work are illustrated in Chp. 10.

3

Swarm Intelligence

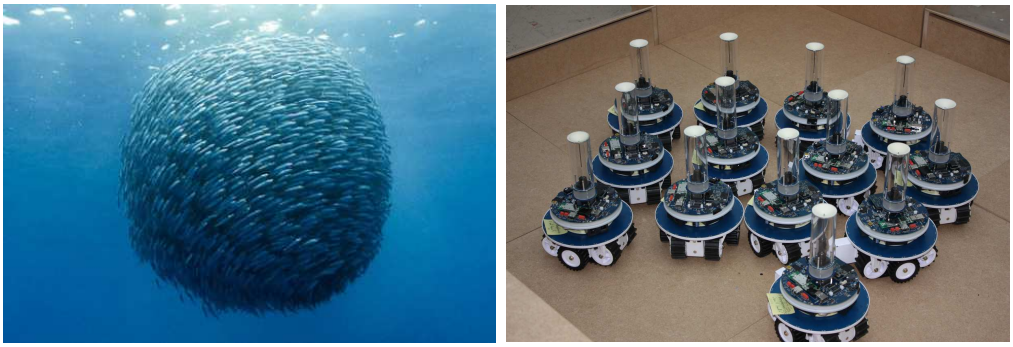


Swarm Intelligence (SI) is a research field, afferent to Artificial Intelligence, that studies the decentralized collective behaviour of entities belonging to both artificial and natural systems. The expression has been used for the first time in the context of cellular robotic systems [15] in 1989 even if studies on interaction rules inside flocks of simulated birds (boids) already existed [6]. SI takes advantage of ideas and theories strongly inspired by biological systems. The rules that lie beneath the natural interactions between individuals in a biological system are quite complex but it is possible to extrapolate the few of them that are useful to guide an artificial system, often created inside a

software simulator. The system commonly used as matter of tests is made up by a population of entities, which are called units, agents or particles in relation to the research field. Entities have the ability to interact with the surrounding environment and with other entities of the population exchanging information in some fashions. Each entity operates autonomously and in a completely decentralized fashion with the purpose to achieve the same target and following the same simple rules. The intelligent behaviour of the group emerges in a self-organized way from the behaviour of each single entity. The most biological groups studied from SI are schools of fishes, flocks of birds, swarms of bees, colonies of ants and herds of animals in general, from which scientist have created many applications in mathematics, statistics, immunology, sociology, engineering and in many other research fields included robotics e.g. multi-robot systems. Swarm Intelligence has grown, in fairly recent period, providing great contribution to both theoretical projects and applications [16, 17, 18, 19, 20].
This great interest in SI is due to the good intrinsic features of the swarm concept in engineering contexts. As previously introduced, the typical swarm intelligence system has the following properties:

- a swarm is composed by many entities;
- entities are relatively homogeneous i.e. equal or belonging to few different typologies;
- entities' interactions are equal or similar and the information exchanged belongs to the same symbolic ontologies;
- each individual action can modify the environment or the behaviour of other entities;
- behaviours and interactions are based on the same simple rules and are computed locally by each entities;
- the action expressed by the whole group results from the combination of the individual coordinated actions without any supervisor.

The artificial system design starts following the above mentioned properties. The obtained system is scalable, fault tolerant and it operates in a parallel fashion. Three desired good properties for an artificial system:



(a) School of fish **(b)** Swarm of robots

Figure 3.1: In a natural system few rules can generate a very complex behaviour that can be reproducible from artificial system with impressive similarity.

- *scalability* is obtainable in a simple way because each entity operates locally and only with its own and neighbour's information. Increasing (or decreasing) the group's entity number, system's functionality remains the same without the need to redefine anything;
- *parallel operations* can be performed because each entity has its own behaviour and capabilities. It computes the basic rules autonomously and takes the decision and the related actions by itself. The resulting system is more flexible and adaptable;
- *fault tolerance* is intrinsic in the concept of swarm. The control system is strongly decentralized because the single entity is autonomous and weakly connected to the swarm. One or more units can be lost without clearly affecting the global behaviour. Moreover, the swarm operates without any problems even in case of one or more units' addition because swarm's behaviour is self-organized.

It is evident that by changing the few basic rules (in the behaviour or in the interactions of entities) it is possible to change the behaviour of the whole swarm by maintaining the above mentioned good properties.

3.1 Applications of Swarm Intelligence

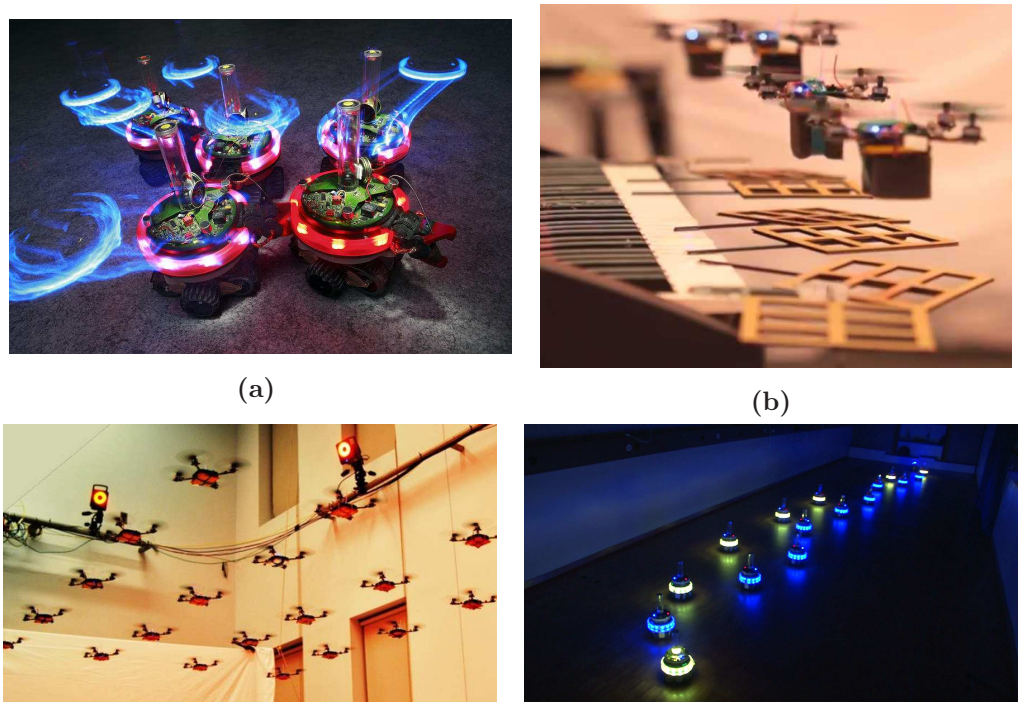
Before presenting in Sec. 3.2 the specific background on PSO, useful to better understand the next chapters, a brief introduction of swarm intelligent applications is due.

- flocks of birds and schools of fishes are the first applications of the swarm intelligence concept. From the first entities developed by Reynolds [6] with the name of boids, this kind of representation is strongly used in CGI and video games applications to simulate different natural movements of many entities;

- the Ant Colony Optimization (ACO) is a meta-heuristic population-based technique commonly used in Operations Research fields to find approximate solutions in difficult problems. Transforming the problem into a weighted graph [21] where the nodes are the "resources" of the problem and the connecting edges are the different paths to reach them, the entities of the system, which are simulated as ants, move on the graph's edges and find the shortest or longest path to link two desired different nodes. Similarly to real ants, the simulated entities release on the path a simulated volatile substance (pheromone) with a specific evaporation ratio. This environmental modification (*stigmergy*) helps the entities to build the correct solution, over the simulated time, because ants are attracted by the pheromone and tend to travel on paths with more quantities of pheromone. Examples of applications can be found in logistic activities as load transportation (such as the well known salesman problem) or network management as well as all those problems that can be easily transformed into graphs and path-finding search problems;

- the Particle Swarm Optimization [22, 23] was born as a meta-heuristic population-based technique useful to find solutions for continuous problems. Inspired by social behaviours in schools of fishes and flocks of birds, the algorithm searches for good solutions in a given optimization problem. Each entity of the system, called particle, embeds a candidate solution for the problem. The simulated swarm of particles flies into the problem's search space, attracted by better solutions. It continuously updates the best solution found among all the swarm's particles during the simulation. Better information on the complete algorithm are presented in the next section.

Many other techniques based on swarm intelligence deserve to be mentioned, especially the ones applied to the main focus of this thesis: mobile robotics.



(a) (b)

(c) (d)

Figure 3.2: The different tasks assigned to the different types of mobile robots show the versatility of this area of robotics. The achievement of each task can involve even different types of communications such as light.

3.2 Background on Particle Swarm Optimization Algorithm

In literature, Particle Swarm Optimization (PSO) is usually considered as an optimization algorithm, i.e. it is typically used to solve optimization problems defined in a $m-$dimensional space by a fitness function $f(\cdot)$, that is subject to predefined constraints and whose value has to be minimized (or maximized). Many applications have been studied [24] and an exhaustive discussion of these can be found in [25] or [26]. In the original PSO formulation [22], the optimal solution is computed by simulating a group of n particles that explore the search space of the problem in order to find the best fitness value. Each agent (or particle) moves in the solution search space with known position and velocity, and with the ability to communicate with other agents. In particular, PSO is based on a population $\mathcal{P}$ of n particles that represents a set of possible solutions of the given m-dimensional problem, i.e. $\|\mathcal{P}\| = n$ where the operator $\|\cdot\|$ computes the cardinality of a given set. Position and velocity of the i^{th} particle at the k^{th} iteration of the algorithm are identified by the m-dimensional vectors

$$\begin{aligned} \boldsymbol{p}_i(k) &= [p_{i,1}(k) \dots p_{i,m}(k)]^T \\ \boldsymbol{v}_i(k) &= [v_{i,1}(k) \dots v_{i,m}(k)]^T \end{aligned} \qquad i = 1 \dots n$$

When the algorithm is initialized, a random position $\boldsymbol{p}_i(0)$ and a starting random velocity $\boldsymbol{v}_i(0)$ are assigned to each particle. At each iteration, a set of candidate solutions is optimized by the particles which are moving through the search space toward better values of the fitness function $f(\cdot)$. Each particle knows the value of the fitness function corresponding to its current position in the m-dimensional search space and it is able to *remember* data from previous iterations. In particular, each particle is able to remember the position where it has achieved the best value of fitness function, namely $\boldsymbol{p}_i^*$ (called *local best*). This value can be exploited by neighbours to change their behaviour. In fact, assuming the possibility of a global communication between the particles, each of them can gather the positions $\boldsymbol{p}_j^*$ of other team-mates where they have detected their best fitness value. Therefore, the best value $\boldsymbol{p}_i^+$ of all the particles can be defined as *global best*. The propagation of $\boldsymbol{p}_i^+$ through the swarm depends on communication topology, thus it may happen that non-communicating particles have temporarily different values. Moreover, each agent can use the best current fitness value of its neighbours $\boldsymbol{p}_i^\times$ (called *neighbourhood best*) as another term to define its

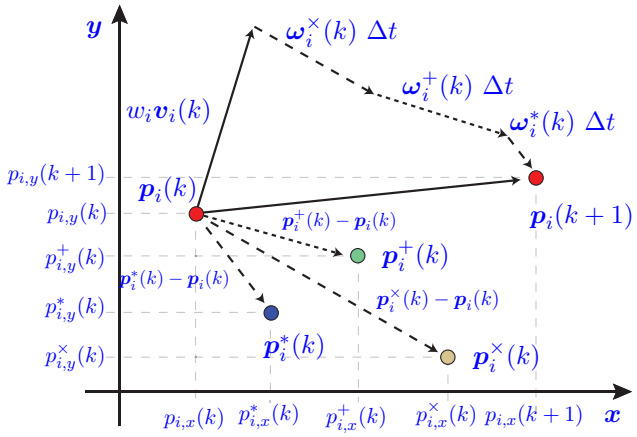


Figure 3.3: Original PSO algorithm applied to particles moving in a 2D environment.

behaviour. Fig. 3.3 clarifies this concept in a 2D space.

In conclusion, at each iteration of the algorithm, the behaviour of each particle is defined by a proper function made of three factors: $\boldsymbol{p}_i^*$ (the local best), $\boldsymbol{p}_i^+$ (the global best), and $\boldsymbol{p}_i^\times$ (the neighbourhood best). From these considerations, it follows that the communication topology chosen to route information among the particles is an important feature able to drastically change the behaviour of the whole swarm. Therefore, it has to be carefully chosen. There are many works on the importance of swarm topological configurations [27, 28]. As an example, in case of a *static* communication topology, the contribution of $\boldsymbol{p}_i^+$ generates a quite *static behaviour*. Therefore, it is not guaranteed that the search space is fully and properly explored. As a matter of fact, in this case the movements of each particle must be somehow limited in order to remain connected to the predefined neighbours. The consequence is that the search possibilities also become limited. It follows that this approach could drive the system to a local minimum and thus to a local optimal solution. Furthermore, if the chosen static communication topology has not enough connections, each agent could even have a different value for $\boldsymbol{p}_i^+$ since the data could not be updated properly. In this case, the swarm could eventually reach dispersed configurations. The ring graph depicted in Fig. 3.4a is an example of a static communication topology where each agent chooses a well defined neighbour, always keeping the same connections.

In order to avoid these problems, we have assumed a dynamic communication topology that depends on inter-particle distances. Each particle can move without constraints in

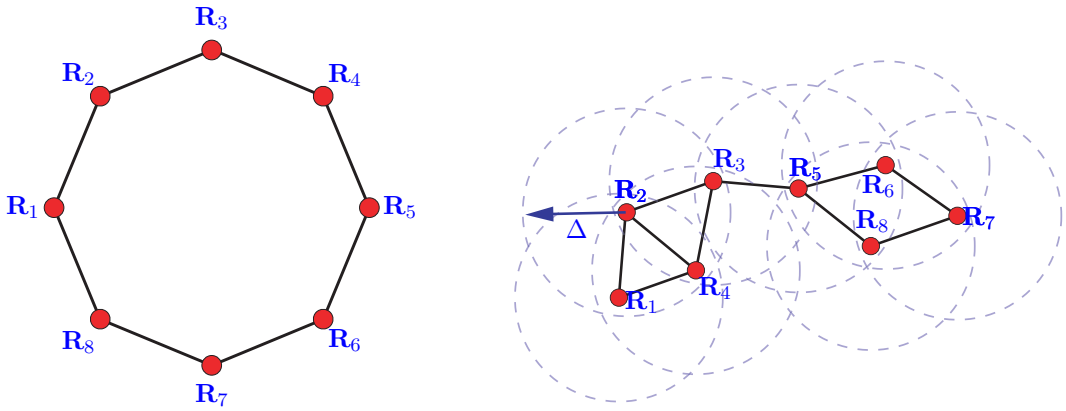


Figure 3.4: Examples of a 2D communication graph in case of $\|\mathcal{P}\| = 8$ with $\Delta_i = \Delta$, $i = 1 \dots 8$: a *ring* graph, 3.4a, and a Δ-disc graph, 3.4b.

the search space and can communicate with any other neighbour in its communication range. In this way we have obtained a better exploration of the search space and a faster update of data related to the environment. If a particle has better data on the environment, its neighbours automatically tend to aggregate around it to reach the best available position (and solution).

Communication graphs, whose topology depends on inter-agents distance, are usually addressed as Δ-disc graphs, see Fig. 3.4b. This kind of topology is typically applied to 2-D networks [29]. With abuse of notation we have addressed distance-based communication topology as Δ-disc graphs also in case of agents defined in $m > 2$ dimensions. To this purpose, we have defined the neighbour-set of the i^{th} particle as the set of all the particles whose distance from $\boldsymbol{p}_i$ is smaller than a predefined threshold Δ_i

$$\mathcal{N}_i = \left\{ j \in \mathcal{P} : ||\boldsymbol{p}_i - \boldsymbol{p}_j|| \leq \Delta_i \right\}$$

where $||\cdot||$ computes the length of a vector. At the k^{th} iteration of PSO algorithm, the state of each particle is updated as follows

$$\boldsymbol{v}_i(k+1) = \xi_0 \left(w_i \boldsymbol{v}_i(k) + \boldsymbol{\omega}_i^*(k) + \boldsymbol{\omega}_i^+(k) + \boldsymbol{\omega}_i^\times(k) \right) \tag{3.1}$$

$$\boldsymbol{p}_i(k+1) = \boldsymbol{p}_i(k) + \boldsymbol{v}_i(k+1)\ \Delta t \tag{3.2}$$

where w_i is a scalar constant that represents the *inertia* [30] of the particle, ξ_0 is a parameter called *constriction factor* [31] (introduced to avoid the dispersion of the

particles), $w_i \boldsymbol{v}_i(k)$ is addressed in literature as *persistence* and represents the tendency of a particle to preserve its motion direction; Δt is the simulation time step. The terms $\boldsymbol{\omega}_i^*(k)$, $\boldsymbol{\omega}_i^\times(k)$ and $\boldsymbol{\omega}_i^+(k)$ represent *historical* and *social* contributions to the control action and they are defined as

$$\begin{aligned} \boldsymbol{\omega}_i^*(k) &= \phi\, r_i^*(k)\ \ (\boldsymbol{p}_i^*(k) - \boldsymbol{p}_i(k))/\Delta t \\ \boldsymbol{\omega}_i^+(k) &= \phi\, r_i^+(k)\ \ (\boldsymbol{p}_i^+(k) - \boldsymbol{p}_i(k))/\Delta t \\ \boldsymbol{\omega}_i^\times(k) &= \phi\, r_i^\times(k)\ \ (\boldsymbol{p}_i^\times(k) - \boldsymbol{p}_i(k))/\Delta t \end{aligned} \tag{3.3}$$

The first term represents the contributions to the velocity given by the *best individual* value of the self particle. The second and third elements represent the *swarm* and the *neighbourhood* contribution respectively. Each term is a vector attracting each particle towards the corresponding point into the search space. The parameters $r_i^*(k)$, $r_i^+(k)$ and $r_i^\times(k)$ are usually selected as uniform random numbers in $[0, 1]$ and they are computed at each iteration of the algorithm to give a range of randomness to the particle's behaviour. The parameter ϕ is used to modulate the maximum influence of this random behaviour. Because of its importance in tuning PSO parameters, the value of ϕ has been determined in literature by using many methods, see e.g. [32], often with empirical approaches as in our cases. As a matter of fact, after many simulations and considering in particular the capability of the swarm to overcome obstacles without loosing any particles, we have set $\phi = 2$ for the algorithm described in Chp. 4 and $\phi = 3.5$ for the algorithm described in Chp. 5. The motivations for the choices on parameters' values and the related simulations are presented in Sec 4.3 and 5.3.

3.3 Application of the Particle Swarm Optimization Algorithm

Two applications of the PSO on Unmanned Ground Vehicles (UGVs) and Autonomous Underwater Vehicles (AUVs) are presented in the next two chapters in both cases to drive a robotic swarm in a-priori unknown environments. In the first application the algorithm has been designed to drive a classic differential wheeled robot swarm, while in the second one it has been applied to underwater robots, which have been modelled as under-actuated single thrust vehicles with four steering fins.

4

A Navigation Strategy for Multi-Robot Systems Based on PSO Techniques

A novel strategy with the aiming to control a group of mobile robots moving through an unknown environment is presented. The proposed control strategy is based on a modified version of the Particle Swarm Optimization (PSO) algorithm, and it has been extensively validated by means of numerical simulations considering complex maze–like environments and groups of robots with different numbers of units.

4.1 Introduction

As well known, the problem of driving a group of mobile robots through an unknown environment from a starting area to a final one while avoiding obstacles has been widely faced in literature on the basis of different classes of algorithms [33]. Among these, one of the main approaches used to solve the problem, which are adopted by many popular techniques, exploits the concept of virtual potential fields (VPFs). The comparison between our technique, based on PSO, and VPFs seems to be useful because in author's opinion among all the commonly used techniques, VPFs is the most close to ours. Indeed, as described e.g. in [34], an *attractive* potential field is associated to the target area, while *repulsive* potential fields are associated to the obstacles sensed in the environment by the robots. The result is that the environment is perceived by

each robot as a *landscape* created by the combination of both static and time-varying factors. In this landscape, *valleys* and *peaks* represent the *global* attracting and repulsive zones respectively. Despite its versatility, virtual potential fields approach has many drawbacks. First of all, the robots have to gather a huge amount of information regarding obstacles, i.e. robots have to exchange many information to coordinate and to reach the target area. Moreover, a typical problem that may arise is the presence of local minima, i.e. areas in which robots are in deadlock situations. This problem has been solved by introducing a new class of *global* potential fields, called *social potentials* [35], that can ensure the convergence of the swarm to its final destination.

Another relevant aspect in the coordination of multiple robots is the definition of the robot-to-robot interactions. As a matter of fact, as pointed out also in [36], by changing the communication topology it is possible to improve the amount of information that each robot can exploit in order to achieve a predefined goal.

The technique presented in this chapter solves these problems by using a very limited exchange of data, a dynamic fitness function, and a free communication topology (cf. Sec. 3.2). The proposed approach for the control of a swarm of robots adopts a meta-heuristic algorithm based on the Particle Swarm Optimization (PSO) (cf. Sec. 3.2). As a matter of fact, this technique has some interesting features that can be exploited for the guidance of a swarm of robots, such as its reliability, its intrinsic simplicity and the relatively small amount of information needed to create the desired emergent behaviours. The PSO approach was originally developed in 1995 [22] to study social interactions and it was initially inspired by flocks of birds. In literature, many modifications to the original PSO algorithm have been proposed in order to improve its efficiency, such as in [30] where a parameter called *inertia* has been introduced to preserve the motion direction of the particles. More recently, PSO algorithms have been applied to path planning for robots, as in [37] or [38], where a multi-objective cost function has been used. An exhaustive analysis of publications on PSO can be found e.g. in [25].

This chapter is organized as follows: the modified version of the PSO is introduced in Sec. 4.2. The results of the simulations, used to validate the presented approach, are analysed in Sec. 4.3. Conclusions and future work are reported in Sec. 4.4.

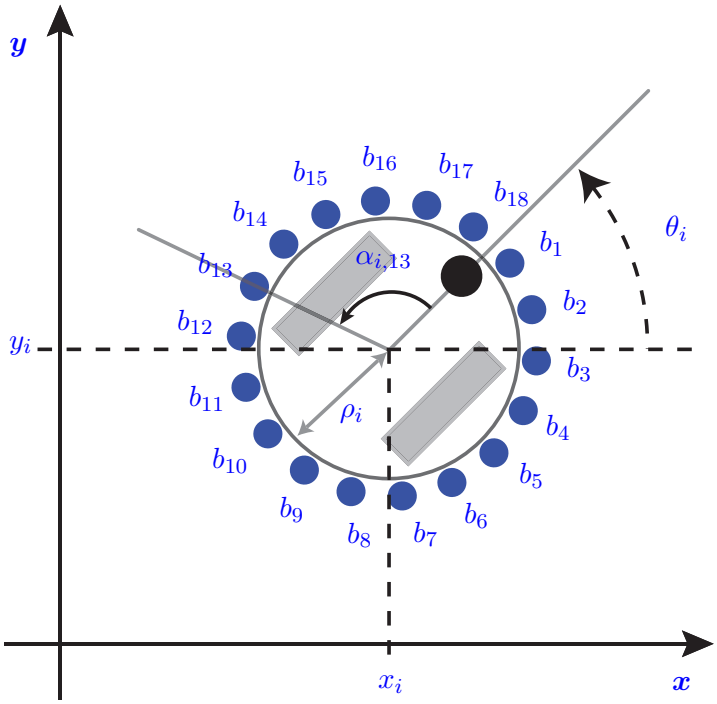


Figure 4.1: Kinematic model of the robots: the black dot identifies the robot's front, $[b_1 - b_{18}]$ are the bumpers, and $\alpha_{i,j}$ is the angle of the j-th bumper.

4.2 Improving the PSO Algorithm

The algorithm presented has been developed in order to drive a swarm of robots $R_i, i = 1, \ldots, n$ moving in unknown planar environments. To this purpose, we have considered a group of differential-wheeled robots (see Fig. 4.1) whose kinematics is given by the following equations

$$\begin{aligned} x_i(k+1) &= x_i(k) + u_i(k)\cos(\theta_i(k)) \\ y_i(k+1) &= y_i(k) + u_i(k)\sin(\theta_i(k)) \\ \theta_i(k+1) &= \theta_i(k) + \omega_i(k)\ \Delta t \end{aligned}$$

where $u_i(k), \omega_i(k)$ are respectively the forward and the steering control velocities of the robot during the time step $[k,\ k+1]$. Two problems need to be solved in order to apply the PSO algorithm to the navigation of a robotic swarm: how to match each particle to a single robot and how to match the search space to the environment surrounding the swarm.

4.2.1 Matching PSO Agents with Physical Robots

As PSO algorithm considers each agent as a single integrator in x, y coordinates (i.e. it calculates a velocity vector $\boldsymbol{v}_i(k) \in \mathbb{R}^2$ without taking into account robot's kinematics), the robot exploits the data $\boldsymbol{p}_i(k+1) = [p_{i,x}(k+1),\ p_{i,y}(k+1)]^T$ given by the PSO as a target for its movements. Since we are dealing with robots moving on a plane surface and it is $\boldsymbol{p}_i(k) = [p_{i,x}(k),\ p_{i,y}(k)]^T$ for R_i, the control input $u_i(k+1)$, $\omega_i(k+1)$ are computed as

$$\begin{aligned} u_i(k+1) &= K_u|\boldsymbol{v}_i(k+1)\ \Delta t| \\ \omega_i(k+1) &= K_\omega\ \text{atan2}({}^y\delta_{p_i}, {}^x\delta_{p_i})\ \Delta t \end{aligned}$$

where ${}^y\delta_{p_i} = p_{i,y}(k+1) - p_{i,y}(k)$, ${}^x\delta_{p_i} = p_{i,x}(k+1) - p_{i,x}(k))$ and K_u, K_ω are two proper constants. In particular, we have assumed that the velocities of the left and right motor (ν_L and ν_R respectively) are saturated at ν_{max}, thus $u_{max} = r\nu_{max}$ and $\omega_{max} = 2u_{max}/d$ where r is the radius of the wheels and d is the distance between them. We have set $K_\omega \gg K_u$ in order to favour the *turning–on–the–spot* behaviour of each robot. In practice, when the control actions are calculated, the robot starts turning on the spot and it moves slowly forward, then it accelerates only when it is almost aligned with the desired direction and it stops when it reaches the target, or detects a collision. Moreover, we have supposed that each robot knows its own position with respect to a global inertial frame.

4.2.2 Matching the Search Space to the Environment

This problem has been solved by defining a proper fitness function embedding not only the optimization problem but also the obstacle avoidance. The fitness function can be described as the distance between two points $\boldsymbol{a}_g$ and $\boldsymbol{a}_i$ belonging to a 3D space called *fitness map*. This map is defined by adding a third component z to the standard x, y coordinates of the robot in the arena. In this manner, the position $\boldsymbol{p}_i = (x_i, y_i)$ of the i^{th} robot in the arena has a corresponding point $\boldsymbol{a}_i = (x_i, y_i, z_i)$ in the map. The coordinate z_i represents, as described in the following, the *cost* of point $\boldsymbol{p}_i$ in terms of robot perception of the environment. In fact, the coordinate z_i is computed by a function $\mathcal{Z}(\boldsymbol{p}_i, k, \mathcal{C}_i^k) \geq 0$ (see eq. (4.3)) that, roughly speaking, is a combination of Gaussian functions related to the distance of the robot from known obstacles.

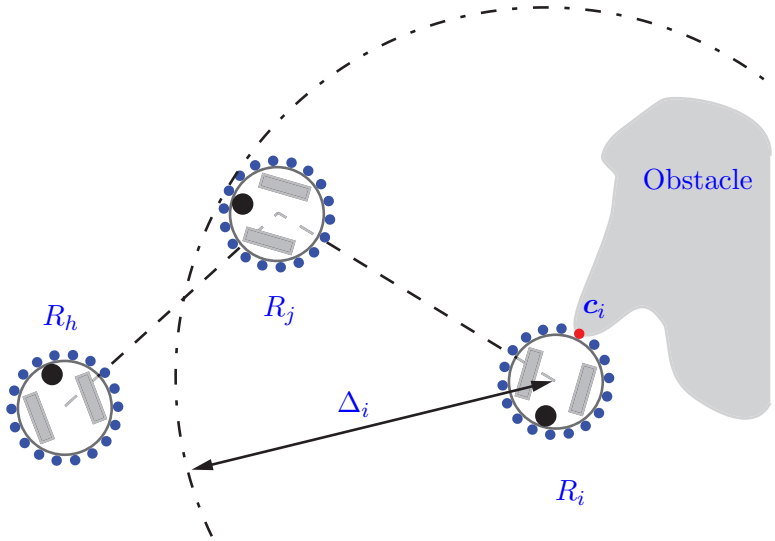


Figure 4.2: Example of broadcasted data for a collision. The dash-dotted circumference shows the communication area of robot R_i while dashed lines represent the inter-robot communication network.

In the presented PSO–based coordination algorithm, each robot of the team, flocking toward a common predefined global target (or *goal*), positioned in $\boldsymbol{p}_g = (x_g, y_g)$ and corresponding to $\boldsymbol{a}_g = (x_g, y_g, 0)$ in the fitness map, evaluates the following fitness function

$$f(\boldsymbol{p}_i, k, \mathcal{C}_i^k) = \gamma_i \; |\boldsymbol{a}_g - \boldsymbol{a}_i(k))| \tag{4.1}$$

where the parameter γ_i can be tuned to modify the relevance given by each robot to the target point. The z component of $\boldsymbol{a}_g$ is always zero, so that the target point $\boldsymbol{p}_g$ defines the minimum value of the fitness function.

Another important aspect has to be considered in order to apply the PSO approach to a robot swarm: the standard PSO algorithm does not consider agents with physical properties or constraints. Therefore, with the purpose of defining an algorithm that allows agents (i.e. robots) to navigate in an unknown environment while avoiding obstacles, passing through narrow passages and preserving connectivity, we have proposed two modifications to the PSO algorithm. These modifications concern the *obstacle avoidance* capability and a modified social interaction between neighbour particles, named *neighbours' aggregation vector*.

4.2.3 Obstacles and Local Minima Avoidance

Since PSO was originally defined as an algorithm to drive particles in a *virtual* m-dimensional environment, the case of agents with physical properties was not considered. In our case, as we are considering real robots, we must take into account that they have a not-null radius (see Fig. 5.1a) and that they can possibly collide with obstacles and teammates. In order to deal with these constraints, we have considered each robot equipped with eighteen bumpers evenly spaced on its perimeter. Moreover, in case a collision is detected, its position is broadcasted to teammates. If we suppose that the robot R_i detects a collision, as depicted in Fig. 5.3c, the position of the contact point $\boldsymbol{c}_{i,h} = [x_{c,i},\ y_{c,i}]^T$ is computed as

$$\begin{aligned} x_{c,i} &= x_i + \rho_i \cos(\alpha_{i,m} + \theta_i) \\ y_{c,i} &= y_i + \rho_i \sin(\alpha_{i,m} + \theta_i) \end{aligned}$$

where ρ_i is the radius of the i-th robot, $\alpha_{i,m}$ is the angle of the m-th bumper (see also Fig. 5.1a), and $h \geq 0$ is a label that uniquely identifies the collision point. Moreover, the time step k_h^0 at which a collision h is detected is saved. Let us remark that, as long as robots do not have any knowledge of the environment, different $\boldsymbol{c}_{i,h}$ could correspond to the same obstacle. More formally, considering for simplicity the 2D case, let us define $\mathcal{O} \in \mathbb{R}^2$ and $\mathcal{R}_i \in \mathbb{R}^2$ as the set of points of the plane that are part of an obstacle and of a robot, respectively. We can then define the set of collisions detected by R_i up to the k-th step as

$$\begin{aligned} \mathcal{C}_i^k \quad := \Big\{ \boldsymbol{c}_{i,h} \in \mathbb{R}^2 : \boldsymbol{c}_{i,h} = (\mathcal{O} \cap \mathcal{R}_i) \wedge (k - k_h^0) \leq \tilde{k} \Big\} \\ h \in [1..n_i(k)] \end{aligned} \tag{4.2}$$

where $\tilde{k}$ represents the *lifetime* of the detected collisions and $n_i(k)$ is the total number of collisions detected by the i-robot from the beginning of the simulation until the time step k. Namely, eq. (4.2) states that a point of the plane can be considered as a collision point for the i-th robot at time k if there is at least an intersection between the set of points of the robot and the set of points of the obstacles. In order to include obstacle avoidance in the fitness function, the function $\mathcal{Z}$ used to compute the fitness map is defined as

$$\mathcal{Z}(\boldsymbol{p}_i, k, \mathcal{C}_i^k) = \beta \sum_{\boldsymbol{c}_{i,h} \in \mathcal{C}_i^k} \Theta(k, k_h^0)\ g(|\boldsymbol{p}_i - \boldsymbol{c}_{i,h}|, \sigma_q) \tag{4.3}$$

where β is a proper parameter, $\Theta(k, k_0^h)$ is a *forgetting factor* for detected collisions, $g(\cdot)$ is a Gaussian function. The forgetting factor $\Theta(\cdot)$ has been introduced in order to decrease in time the effects of an obstacle. This introduction is due in consideration of the fact that, when robots move in the environment, the obstacles earlier detected are probably already far from them, consequently the related stored collision data have a minor probability to be helpful in real time obstacle avoidance at k^{th} step. For $k \geq k_h^0$, it is defined as

$$\Theta(k, k_h^0) = \begin{cases} \dfrac{\lambda}{k - k_h^0 + 1} & \text{if } (k - k_h^0) \leq \tilde{k} \\ 0 & \text{otherwise} \end{cases}$$

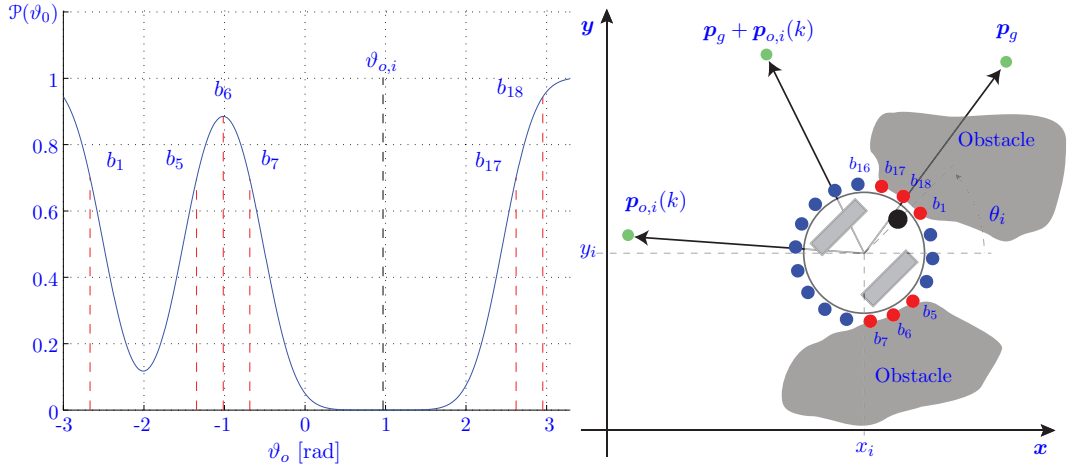
where $\tilde{k}$ is defined in eq. (4.2). In our case we set $\lambda = 1$ and $\tilde{k} = 500$. The term $g(|\boldsymbol{p}_i - \boldsymbol{q}_j|, \sigma_q)$ is a normal Gaussian distribution defined as

$$g(|\boldsymbol{p}_i - \boldsymbol{q}_j|, \sigma_q) = \frac{1}{\sigma_q \sqrt{2\pi}} \, e^{\dfrac{-|\boldsymbol{p}_i - \boldsymbol{q}_j|^2}{2\sigma_q^2}}$$

where the variance σ_q is used to shape the influence of a collision on its surrounding area. This way, the function (4.3) defines a map shaped with 'peaks' and 'valleys' generated by collisions. When two robots are within the communication range, the relative collision sets $\mathcal{C}_i^k$ are exchanged, and each robot merges its information with those of the other one. In this manner, by sharing information about detected collisions, a *collective memory* is introduced able to faster drive agents towards the target while avoiding obstacles.

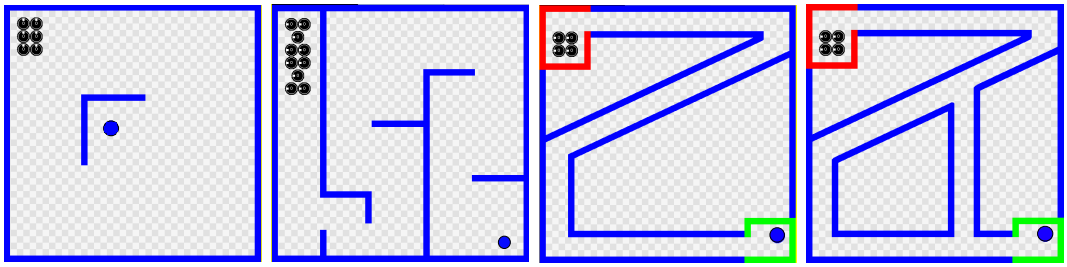
However, the obstacle avoidance technique defined by using (4.3) is not sufficiently fast for avoiding local minima in real time. Indeed, the fitness function takes into account obstacles, but the detected collisions do not directly affect the robot's motion and therefore a considerable amount of time could be necessary to move a robot away from critical zones. In order to solve this problem, a technique inspired by the dynamic window approach [39] has been applied. In particular, as depicted in Fig. 4.3a and similarly to what described in [40], we have assumed that each robot can project around itself a probability curve that depends on the obstacles detected by on board sensors. Thus, each robot computes the relative angle $\vartheta_{o,i}$ where the probability of colliding with an obstacle is minimal

$$\vartheta_{o,i} = \min(\varphi_{o,i}), \qquad \varphi_{o,i} = \sum_{h=1}^{N_b} b_h \; g(\beta, \sigma_{h,i}, \alpha_{h,i}) \tag{4.4}$$



(a) Probability curve related to (4.4). **(b)** Escape windows technique.

Figure 4.3: Example of escape window computed by considering obstacles detected by bumpers $\{b_{17},\ b_{18},\ b_1\}$ and $\{b_5,\ b_6,\ b_7\}$.



(a) Scenario 1 **(b)** Scenario 2 **(c)** Scenario 3 **(d)** Scenario 4

where $\beta \in [-\pi \dots \pi]$, $b_h = 1$ if the h^{th} bumper detects a collision and $b_h = 0$ otherwise, and $g(\cdot)$ is the normal Gaussian distribution centred on $\mu = \alpha_{h,i}$ defined by

$$g(\beta, \sigma_{h,i}, \alpha_{h,i}) = \frac{1}{\sigma_{h,i}\sqrt{2\pi}}\ e^{-\frac{(\beta - \alpha_{h,i})^2}{2\sigma_{h,i}^2}}$$

where the variance $\sigma_{h,i}$ can be used to define how each bumper affects robot's perceptions. Once the value of $\vartheta_{o,i}$ in (4.4) is determined, a *virtual target* $\boldsymbol{a}_{o,i}(k) = [x_{o,i}(k),\ y_{o,i}(k),\ 0]^T$ is temporarily defined, with

$$\begin{aligned} x_{o,i}(k) &= x_i(k) + \varrho_i \cos(\vartheta_{o,i}(k)) \\ y_{o,i}(k) &= y_i(k) + \varrho_i \sin(\vartheta_{o,i}(k)) \end{aligned}$$

where $\varrho_i = |\boldsymbol{p}_g - \boldsymbol{p}_i(k)|$ is the distance between the actual position of the robot and the

target area's center. The fitness function in (4.1) is then modified as (see Fig. 4.3b)

$$f(\boldsymbol{p}_i, k, \mathcal{C}_i^k) = \gamma_i |(\boldsymbol{a}_g + \boldsymbol{a}_{o,i}(k)) - \boldsymbol{a}_i(k)|$$

4.2.4 Neighbors Aggregation Vector

Bearing in mind the things that have been presented in Chp. 3.2, especially the neighbours contribution in (3.3), the last main modification introduced respect to the standard PSO algorithm is the *Neighbours Aggregation Vector* (NAV), indicated as $\boldsymbol{\omega}_i^{\otimes}(k)$. This term replaces $\boldsymbol{\omega}_i^{\times}(k)$ in (3.1) and it is inspired by the idea that *if the team-mates move in a certain direction, maybe that direction is the right one* [6]. Indeed, the term $\boldsymbol{\omega}_i^{\otimes}(k)$ modifies the robots' behaviour by taking into account the best solution found by their neighbours, and is defined as

$$\boldsymbol{\omega}_i^{\otimes}(k) = \phi \cdot r_i^{\otimes}(k) \sum_{\boldsymbol{p}_j \in \mathcal{N}_i} \frac{\boldsymbol{p}_j^{\times}(k) - \boldsymbol{p}_i(k)}{\Delta t} \tag{4.5}$$

where $r_i^{\otimes}(k) \in [0,\ 1]$ is a random number updated at each iteration and ϕ a proper tuning value. In a nutshell, (4.5) represents a randomly weighted barycentre of the position where the neighbours of R_i have detected their *neighbour best* at the k^{th} time step.

4.2.5 Dynamic Constriction Factor

Moreover, the parameter ξ_0 in (3.1) is replaced by a dynamic *constriction factor* [31] that depends, for each robot, on the mean value of the number of collisions memorized by the robot itself in the last 3 time steps. If we assume $\mathcal{D}_i^j \subset \mathcal{C}_i^k$ where $j \in [k-2, .., k]$ the constriction factor is defined as

$$\xi_i = \xi_0 (1 + \frac{1}{3} \sum_{j=k-2}^{k} \|\mathcal{D}_i^j\|) \tag{4.6}$$

In practice, the term ξ_i in (4.6) is a sort of *collision trading* that increases if a high number of collisions is detected, thus changing robot's behaviour and pushing it away from critical zones. It follows that in environments with few obstacles robots flock clustered, while in case of many constraints robots are forced to move sparsely.

To conclude, with the proposed modifications, the standard PSO algorithm defined in (3.1) and (3.2) is rewritten as

$$\begin{aligned} \boldsymbol{v}_i(k+1) &= \xi_i \left(w_i \boldsymbol{v}_i(k) + \boldsymbol{\omega}_i^*(k) + \boldsymbol{\omega}_i^{\otimes}(k) + \boldsymbol{\omega}_i^{+}(k) \right) \\ \boldsymbol{p}_i(k+1) &= \boldsymbol{p}_i(k) + \boldsymbol{v}_i(k+1)\ \Delta t \end{aligned}$$

4.3 Simulations and Comments

The presented PSO algorithm has been widely tested in our simulated environment J.U.S.E. [41], better described in Sec. 9.1. In particular, we have considered a 2×2 $[m]$ virtual arena where the starting and goal areas are placed in non-trivial positions, i.e. with at least an obstacle between them. Robots have been modelled as real differential-wheeled robots with radius 0.05 $[m]$, maximum speed $u_{max} = 0.35$ $[m/s]$, $w_{max} = 7$ $[rad/s]$ and communication range $\Delta = 0.4$ [m]. Obstacles have been modeled as rectangles of arbitrary length and width of 0.05 $[m]$, and a sample time of $\Delta t = 0.01[s]$ has been selected for the controller.

To enlighten the algorithm's performances, we have considered different scenarios, see figure-set 4.4a–4.4d. The starting area has been placed in the upper-left corner while the goal area is at the opposite corner, except in the case of Fig. 4.4a, where the target is in the center of the arena.

In order to gather statistical results about the performance of the proposed PSO algorithm in driving robot swarms, we have performed 20 simulations, executed in the arena depicted in Fig. 4.4c using both standard PSO and PSO–NAV. The results are shown in Fig. 4.4 where the mean square error (MSE) and standard deviation (STDEV) for the data set provided by a team of 5 robots are reported. Note that the distance error between each robot and the final goal area does not converge to zero if standard PSO is used. Vice versa, by using the NAV, the system converges to a value close to zero. Due to robots' physical dimensions, the value of the MSE cannot be null because robots aggregate around the final point. STDEV demonstrates that despite the random parameters introduced in (3.1)–(3.2) and (4.5), the algorithm is able to adapt and to drive the robots to the goal area. Moreover, let us remark that the Z–shaped maze in Fig. 4.5 would not allow to either classic potential field algorithms or original PSO algorithm to reach the target area since, before moving toward the final location, robots

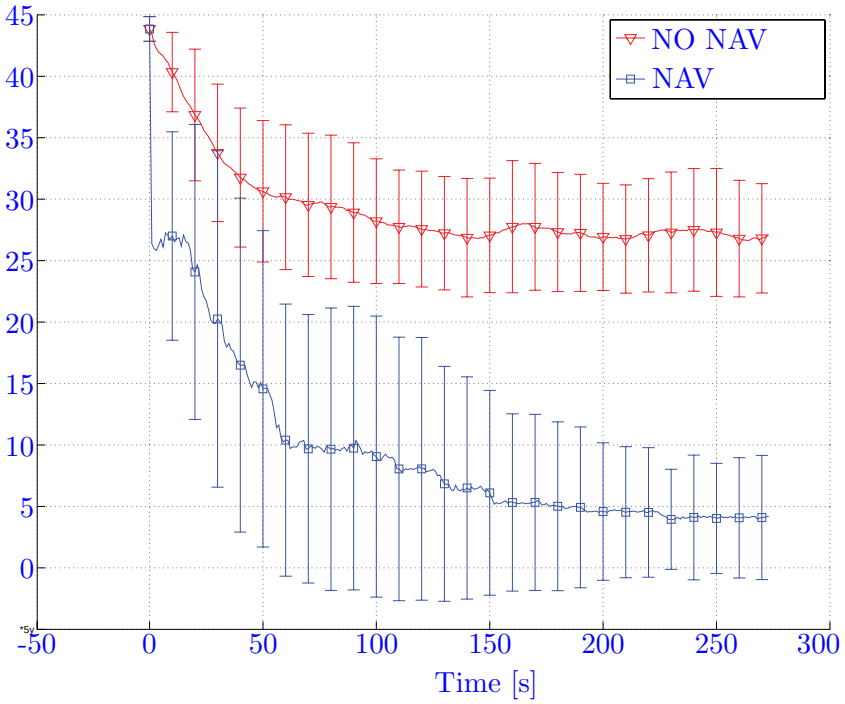
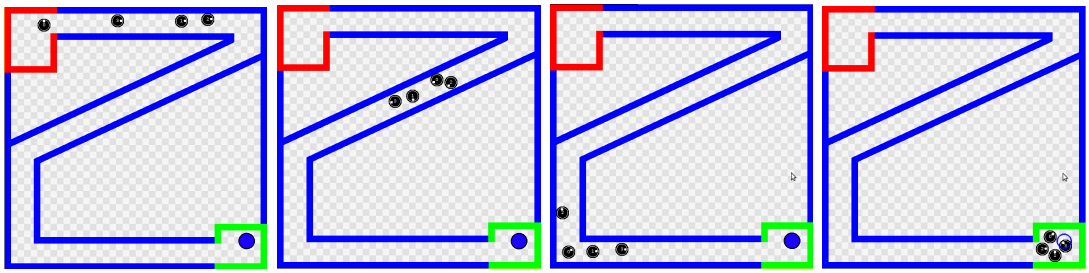


Figure 4.4: MSE and STDEV of distance error between each robot's position and the goal both with and without NAV. Data gathered over 20 simulations on Scenario 3.

have to move temporarily away from it. For this reason, PSO-NAV increases swarm's dispersion to overcome local minima as shown in Fig. 4.4 at about T=100 [s]. This is related to the choice $\phi = 2$: lower values make the swarm too compact and unable to overcome obstacles (and local minima), while higher values makes the swarm to free and thus not able to converge after their exceeded. Moreover, notice that the considered scenarios do not have the same complexity. Scenario 3 and 2 are the most difficult. In scenario 2, the obstacle located in the center of the arena decreases the NAV contribution and the algorithm's performances are close to classical PSO.

Fig. 4.5 shows a simulation involving 4 robots in Scenario 3. We have considered swarms with a number of robots ranging from 3 to 12. From the case studies, it results that swarms with less than 3 robots are usually not able to perform the given tasks in complex scenarios, while more than 10-12 robots create too many collisions and the algorithm fails to maintain the group compact. In this case, the swarm breaks up into small groups that reach the target autonomously. Probably, with larger arenas, the number of robots necessary to have a fully functional swarm would be greater but in our environment, the simulations suggest that 4-6 robots are the best trade-off. Finally,



(a) t = 5 s **(b)** t = 50 s **(c)** t = 170 s **(d)** t = 270 s

Figure 4.5: Scenes of a simulation with 4 robots in Scenario 3.

we want to remark that only a very limited set of data is exchanged by robots during their rendez-vous: the three vectors $\boldsymbol{p}_i^+$, $\boldsymbol{p}_i^\times$, $\boldsymbol{p}_i^*$, and the set $\mathcal{C}_i^k$.

4.4 Conclusions and Future Work

In this chapter, a novel version of the classic Particle Swarm Optimization algorithm has been proposed in order to drive a group of mobile robots in unknown environments from a starting point to a final one. In particular, PSO algorithm has been modified in order to consider robots' physical constraints, i.e. their dimensions and their interaction with the environment. Mobile robots have been modelled as differential-wheeled robots, able to access their own position, to broadcast data to neighbours and to sense the environment by using bumper sensors. The performances of this modified version of PSO algorithm have been validated using the data collected in several simulations, where different groups of robots have been simulated in many maze-like environments. Future work will include experiments on real robots such as the one described in [41] and in Sec. 7.1. Moreover, the application of this algorithm to more complex systems as groups of heterogeneous robots (e.g. groups of robots including both ground and aerial vehicles) and in more complex environments (e.g underwater) will be considered.

5

A Distributed Multi-Level PSO Control Algorithm for Autonomous Underwater Vehicles

This chapter presents a distributed control technique based on the Particle Swarm Optimization algorithm and able to drive a group of autonomous robots to a common target point in unknown environments. We have considered in particular the case of underwater vehicles. The algorithm is able to deal with complex scenarios, frequently found in benthic exploration e.g. in presence of obstacles, caves and tunnels. Moreover, the algorithm is able to consider the case of a mobile target. The data are asynchronously exchanged between the vehicles and dynamic communication topologies have been considered. Simulation results are provided to show the features of the proposed approach.

5.1 Introduction

In this chapter, a navigation control technique able to drive a group of autonomous underwater vehicles (AUVs) is presented. Main features of this technique are its distributed structure and intrinsic robustness with respect to the presence of unknown obstacles, tunnels and dead ends, and also with respect to the possibility of loosing

one or more units. Moreover, the case of a mobile final target has also been considered. This technique is based on the fairly recent theory of *Swarm Intelligence*, which takes into account the study of self-organizing systems [17]. This means that the action expressed by the whole group results from the combination of coordinated actions by individual entities. Initially, simple motion rules have been defined from the study of these actions [6]. More recently, more complex techniques have been introduced, such as the optimization algorithm initially considered for the development of our navigation technique. As presented in section 3.2, the Particle Swarm Optimization algorithm (PSO) [23, 24] is a meta-heuristic algorithm biologically inspired by flocks of birds. It is a non-gradient and direct-search based optimization strategy, in which a set of $\mathcal{N}$ possible solutions (namely a population of particles) is iterated in parallel. They search for the best solution in a multi-dimensional space (or domain). As a matter of fact, this technique has some interesting features that can be exploited in the guidance of a swarm of robots, such as its reliability and flexibility, its intrinsic simplicity, the robustness to failures and the relatively small amount of information needed to create desired emergent behaviours. PSO approach was originally developed in 1995 to study social interactions, [22]. Later, many modifications to the original formulation of the algorithm have been proposed in order to improve its efficiency [25] and, more recently, PSO algorithms have been applied to robot navigation [42], [43] and path planning [38]. PSO has also been used in a hybrid fashion with other bio-inspired techniques: Genetic Algorithm (GA) [44] and Artificial Neural Network (ANN) [45]. Concerning control strategies for autonomous underwater vehicles, many papers have been presented in the literature, see e.g. [46, 47]. Some applications of the PSO algorithm on AUVs can also be found e.g. in [48] and [49], although probably the most interesting modification to original PSO algorithm, useful to guide a swarm of robots, has been only recently introduced: the asynchronous and completely decentralized features described e.g. in [50], where each particle evaluates PSO rules autonomously and a dynamic communication topology is considered [27]. Recent applications of these features are presented in e.g. [43, 51, 52, 53]

A novel improvement of the algorithm is presented in this chapter, starting from its asynchronous version in order to use it in the navigation of underwater vehicles. The purpose is to make the group of robots suitable for seabed and cave exploration,

being simultaneously able to avoid collisions with obstacles and team-mates. Navigation algorithm is completely decentralized and structured in three levels. The two upper levels, which are directly based on PSO, provide navigation way-points and obstacle avoidance respectively. The third one is devoted to AUV control with standard techniques. Each agent is autonomous and it exploits not only its own knowledge of the environment, but also the data received from reachable neighbours. A particular feature of this technique is the addition of obstacle avoidance. Unknown environment barriers, perceived by vehicle sensors, are displayed by the algorithm as constraints into the search space. During the navigation, agents exchange data about best locations and encountered obstacles. The sharing of these information creates a kind of *collective memory* that is used in the selection of future way-points towards final destination. This procedure runs asynchronously and in an intermittent fashion, depending on a free communication topology which is a function of inter-agent distances.
This chapter is organized as follows: the developed PSO technique is described in Sec. 5.2 while the simulations used to validate our approach are presented and discussed in Sec. 5.3. Final conclusions and future work are reported in Sec. 5.4.

5.2 The Algorithm

The presented algorithm has been developed with the aim to drive a swarm of underwater vehicles $\mathcal{V}_i$, $i = 1, \ldots, n$ moving in unknown 3D environments. Each vehicle is autonomous: this means in particular that a local version of PSO algorithm is executed individually by each agent, and that all the decisions are made on the basis of the data locally available at the k^{th} step. In order to apply PSO algorithm to AUV 3D navigation, there are some problems to be solved: the matching of each particle to a single vehicle; the matching of the PSO search space with the environment surrounding the swarm and the manner to embed obstacle avoidance capabilities; local minima avoidance and the transformation of exchanged information, collected by each vehicle, into a unique collective map.

5.2.1 Matching Particles with Physical Vehicles

In this section, we describe the model of the vehicles considered in the PSO algorithm (see also Sec. 3.2), the general control structure, and the matching between the mobile

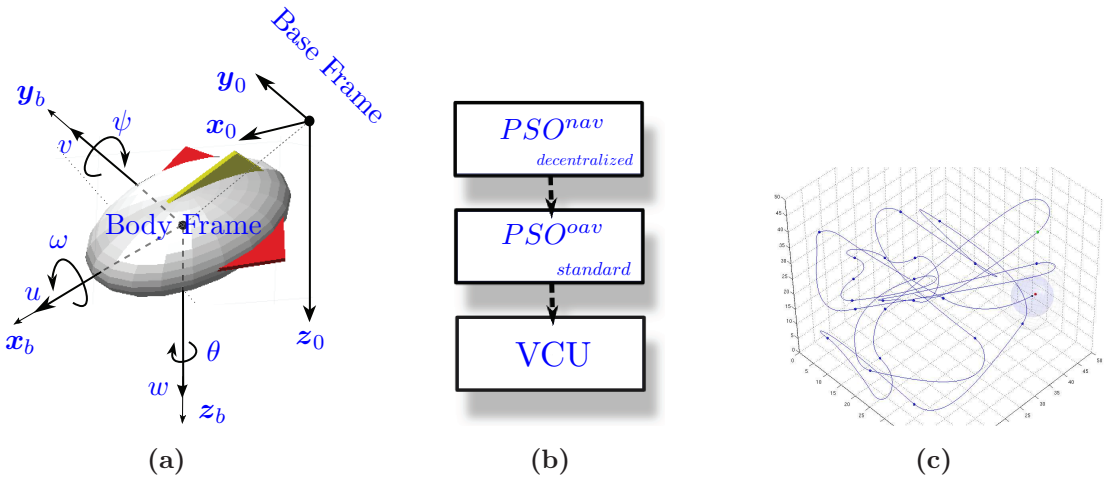


Figure 5.1: In order to define AUV's movements and to follow way-points, two different reference frames are needed, namely the Body and the Base Frame. By exploiting VCU's input variables, PSO control algorithm is able to change the AUV's attitude and speed, providing underwater navigation.

robots and the particles of the algorithm.

5.2.1.1 Model of the AUV

Since our interest is mainly focused on the distributed control algorithm, we have taken into account only the kinematic model of the vehicle, neglecting its dynamics as also done in many other works in literature. We have defined a body reference frame ${}^bF_i = \{x_{b,i}, y_{b,i}, z_{b,i}\}$ for each vehicle, and a common inertial (base) frame ${}^oF = \{x_0, y_0, z_0\}$, as shown in Fig. 5.1a (for the sake of simplicity, in the following the subscript i will be neglected). Moreover, we have assumed that each vehicle is equipped with an engine generating a linear velocity u along the x_b direction, and with four fins that can change the orientation of the vehicle itself with respect to ${}^oF$. We have used RPY angles to describe the orientation of the vehicle with respect to ${}^oF$, as shown in Fig. 5.1a. This way, the state of the vehicle can be defined by: the body-fixed linear velocity vector ${}^b\dot{\mathbf{x}} = [u, v, w]^T$, the body-fixed angular velocity vector ${}^b\boldsymbol{\omega} = [\omega, \psi, \theta]^T$ and the three RPY angles (see Tab. 5.1). The correspondence between velocity in body and inertial frame is expressed by (see also [46])

$$\dot{\mathbf{x}}_v = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \mathbf{R}_{zyx}(\Theta, \Psi, \Omega)\ {}^b\dot{\mathbf{x}} \qquad \dot{\mathbf{\Gamma}}_v = \begin{bmatrix} \dot{\Omega} \\ \dot{\Psi} \\ \dot{\Theta} \end{bmatrix} = \frac{1}{c_\Psi} \begin{bmatrix} c_\Psi & s_\Omega s_\Psi & c_\Omega c_\Psi \\ 0 & c_\Omega c_\Psi & -s_\Omega c_\Psi \\ 0 & s_\Omega & c_\Omega \end{bmatrix} {}^b\boldsymbol{\omega} \quad (5.1)$$

Table 5.1: Kinematic variables

Name	Description	Unit	Name	Description	Unit
x	Position with respect to x_0	m	u	Linear velocity along x_b	m/s
y	Position with respect to y_0	m	v	Linear velocity along y_b	m/s
z	Position with respect to z_0	m	ω	Linear velocity along z_b	m/s
Ω	Roll angle with respect to x_0	rad	w	Angular velocity about x_b	rad/s
Ψ	Pitch angle with respect to y_0	rad	ψ	Angular velocity about y_b	rad/s
Θ	Yaw angle with respect to z_0	rad	θ	Angular velocity about z_b	rad/s

where

$$\mathbf{R}_{zyx}(\Theta,\Psi,\Omega)=\mathbf{R}_z(\Theta)\mathbf{R}_y(\Psi)\mathbf{R}_x(\Omega)=\begin{bmatrix} c_\Theta c_\Psi & c_\Theta s_\Psi s_\Omega - s_\Theta c_\Omega & c_\Theta s_\Psi c_\Omega + s_\Theta s_\Omega \\ s_\Theta c_\Psi & s_\Theta s_\Psi s_\Omega + c_\Theta s_\Omega & s_\Theta s_\Psi c_\Omega + c_\Theta s_\Omega \\ -s_\Psi & c_\Psi s_\Omega & c_\Psi c_\Omega \end{bmatrix}$$

is the matrix that represents the relative orientation of the two frames, and $c_a = \cos(a), s_a = \sin(a)$. Notice that this model presents a singular configuration if $\Psi = \pm\frac{\pi}{2}$ *rad*: this is acceptable since the vehicles typically do not operate close to this configuration. However a different representation, free of this kinematic singularity, can be defined as in [54], where quaternions have been used. Notice also that, due to the non-holonomic characteristics of the vehicle, velocities v and w are always null. Finally, the inverse homogeneous transformation matrix between ${}^bF$ and ${}^oF$ is:

$${}^b\mathbf{T}_o = \begin{bmatrix} {}^o\mathbf{R}_b^T & -{}^o\mathbf{R}_b^T \boldsymbol{x}_v \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{5.2}$$

where ${}^o\mathbf{R}_b = \mathbf{R}_{zyx}(\Theta,\Psi,\Omega)$. Control inputs available to drive the vehicle are engine thrust, that in our model corresponds to the velocity u along x_b, and the position of the four fins. In particular (see Fig. 5.2), if 'fin up' ($\mathcal{F}_u$), 'fin down' ($\mathcal{F}_d$), 'fin left' ($\mathcal{F}_l$), and 'fin right' ($\mathcal{F}_r$), represent the four fins, three fin configurations can be defined:

- α: $\mathcal{F}_u$ and $\mathcal{F}_d$ both rotated of an α angle on the right of vertical axis. This produces a *Yaw* angle rotation of the vehicle;
- β: $\mathcal{F}_l$ and $\mathcal{F}_r$ both rotated of a β angle over the top of horizontal axis. This produces a *Pitch* angle rotation of the vehicle;
- γ: all the fins are rotated on the left of the relative fin's axis. This produces a *Roll* angle rotation of the vehicle.

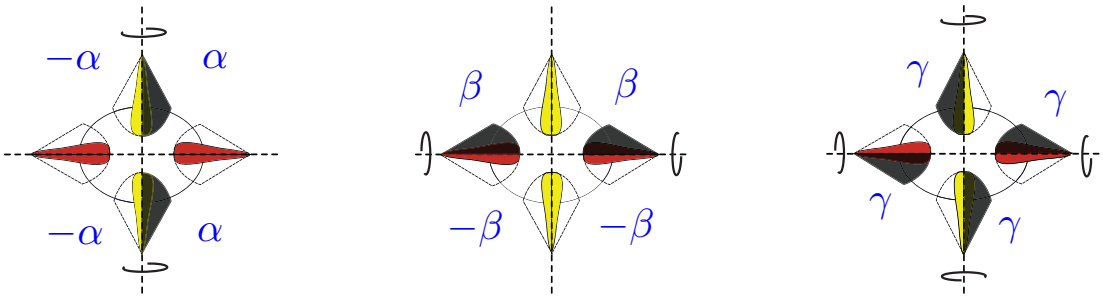


Figure 5.2: Rear view of the AUV with directional fins in α, β, γ configurations.

We have assumed that the range of motion of each fin is bounded, i.e. $\alpha,\ \beta,\ \gamma \in [-0.26,\ 0.26]\ rad$. Furthermore, when the vehicle is moving and fin positions are not null, angular velocities are not null as well. We have modelled the relation between fin positions $[\alpha,\ \beta,\ \gamma]^T$ and angular velocities $[\omega, \psi, \theta]^T$ in ${}^bF$ with a (non-linear) function

$$ {}^b\boldsymbol{\omega} = \boldsymbol{K}_\omega\ u\ \sin\left([\gamma, \beta, \alpha]^T\right) \tag{5.3} $$

where $\boldsymbol{K}_\omega$ is a proper diagonal gain matrix. In conclusion, taking into account that control input consists in the vector $\boldsymbol{\tau} = [u, \alpha, \beta, \gamma]^T$ and that velocities v and w are null, the kinematic model of each AUV is given by (5.1) and (5.3).

5.2.1.2 General Structure of the Control System

On the basis of the results presented in [46], we have designed the vehicle's low-level control (the *Vehicle Controller Unit* (VCU)) as an autopilot that provides a point-to-point navigation mode (see Fig. 5.1b). To this purpose, we have assumed that the AUV state variables (see Tab. 5.1) are available, that a desired way-point ${}^b\boldsymbol{x}_d^{VCU} = [x_d, y_d, z_d]^T$ and a desired velocity vector ${}^b\dot{\boldsymbol{x}}_d^{VCU}$ have been assigned in the body-frame (provided by PSO algorithm, see below). Then, VCU has to generate both the linear speed u and the fin angular positions α, β, γ. Linear speed is computed using the following simple control law

$$ {}^b\dot{\hat{\boldsymbol{x}}}_v = {}^b\dot{\boldsymbol{x}}_d^{VCU} + \boldsymbol{K}_p\,(\,{}^b\boldsymbol{x}_d^{VCU} - {}^b\boldsymbol{x}_v) \tag{5.4} $$

from which $u = ||{}^b\dot{\hat{\boldsymbol{x}}}_v||$. $\boldsymbol{K}_p$ is a proper constant (diagonal) matrix used to tune the proportional term. The steering variables $[\alpha, \beta, \gamma]^T$ are also computed on the basis of

the desired position ${}^b\boldsymbol{x}_d^{VCU}$. In fact, with a simple transformation from Cartesian to spherical coordinates given by

$$\begin{cases} \Psi_d &= atan2(y_d, x_d) \\ \Theta_d &= atan2(z_d, \sqrt{x_d^2 + y_d^2}) \\ \epsilon_d &= \sqrt{x_d^2 + y_d^2 + z_d^2} \end{cases} \tag{5.5}$$

azimuth (Ψ_d) and elevation (Θ_d) values are obtained and used to compute the desired rotational velocities in ${}^bF$ as

$$\psi = k_\Psi (\Psi_d - \Psi), \qquad \theta = k_\Theta (\Theta_d - \Theta) \tag{5.6}$$

from which, by exploiting (5.3) and assuming $\omega = 0$, it is finally possible to obtain fin positions α and β useful to reach the given way-point as

$$[0, \beta, \alpha]^T = \arcsin\left(\frac{1}{u}\boldsymbol{K}_\omega^{-1}\ [0, \psi, \theta]^T\right)$$

Notice that it is not necessary to reach a specific final AUV orientation, but only to reach the target. For this reason, the γ fin configuration, for the control of the Ω rotation, has not been considered. The parameter ϵ_d in (5.5) is used by the algorithm as an estimation of the remaining distance between the robot and the way-point target. A way-point navigation test is depicted in Fig. 5.1c.

As described in Sec. 3.2, PSO algorithm considers each particle/unit as a single integrator in x, y, z coordinates, i.e. it calculates a velocity vector $\boldsymbol{v}_i(k) \in \mathbb{R}^3$ without taking into account vehicle's kinematic constraints deriving from its non-holonomic characteristic. In order to solve this problem, we have developed an algorithm structured into two levels, called PSO^{nav} and PSO^{oav}. They provide navigation and obstacle avoidance respectively, see Fig. 5.1b. Each level is managed by a proper software agent (later identify as $\mathcal{SA}$).

Both levels' agents run on-line in a decentralized fashion on each AUV and compute their respective targets as detailed below. PSO^{nav} has a distributed structure and each $\mathcal{SA}^{nav}$ considers itself as a particle and the neighbor vehicles as other particles inside its work-space. By exploiting the information given by sensors and team-mates, the $\mathcal{SA}^{nav}$ computes its future position according to the PSO basic rules, see Sec. 3.2. This position is taken as a temporary way-point $\boldsymbol{x}_{wp}^{nav}$ of the global path in order to reach the final target. In the lower level, the PSO^{oav} agent adopts the way-point $\boldsymbol{x}_{wp}^{nav}$

provided by $\mathcal{SA}^{nav}$ as optimization target, and it tries to find the best way to reach this point while avoiding obstacles (see next Section on obstacle avoidance). $\mathcal{SA}^{nav}$ runs only when the previous way-point is reached by the vehicle. Instead, $\mathcal{SA}^{oav}$ runs continuously, in a real-time fashion, providing a series of better *local* positions over the time, namely $\boldsymbol{x}_{wp}^{oav}$. Notice that the unit computes both the PSO^{nav} and the PSO^{oav} in an asynchronous and completely decentralized fashion. Finally, the way-point $\boldsymbol{x}_{wp}^{oav}$ is transformed from the base frame ${}^{o}F$ to the body frame ${}^{b}F$ by using (5.2) and it is exploited by the unit's VCU as the desired ${}^{b}\boldsymbol{x}_{d}^{VCU}$ in (5.4) and (5.5). When the vehicle reaches $\boldsymbol{x}_{wp}^{nav}$, the loop starts again until final target is reached.

5.2.2 Matching the Environment with the Search Space

The solution of this problem is not easy because in real space there are physical obstacles that need to be mapped, creating therefore a *constrained* search space. We will show how the search space on which PSO algorithms work can be modified to embed the detected obstacles, and how $\mathcal{SA}^{oav}$ exploits these modifications for obstacle avoidance. As first step, it is necessary to define the fitness function used to solve optimization problems.

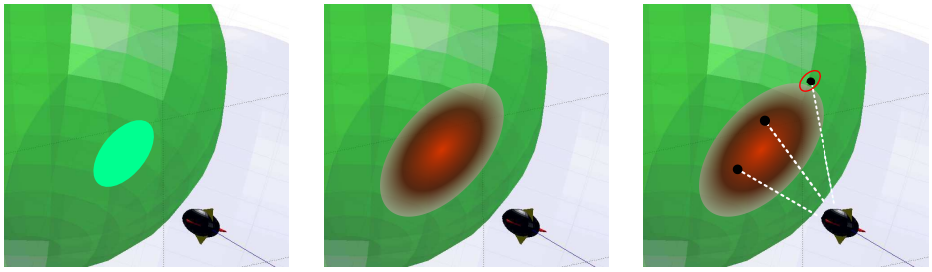
The fitness function used in both PSO levels is a simple distance function between two points in a 3D space, i.e.

$$f(\boldsymbol{p}_i^j, \boldsymbol{p}_{goal}^j, k) = \gamma_i^j \ \|\boldsymbol{p}_{goal}^j(k) - \boldsymbol{p}_i^j(k))\|, \qquad i = 1, \ldots, n_j \quad j = nav, oav$$

where the parameter γ_i can be tuned to modify the relevance given by each particle/agent to the target point. Considering PSO^{nav} algorithm, the parameter $\boldsymbol{p}_i = [x_i, y_i, z_i]^T$ is the position of the i^{th}–AUV ($\mathcal{V}_i$), while $\boldsymbol{p}_{goal}$ is the global target that the *swarm* must reach. Notice that from VCU's point of view $\boldsymbol{p}_i = \boldsymbol{x}_v$. The parameter n_{nav} is equal to i^{th}–AUV's neighbours. On the other hand, in case of PSO^{oav}, the parameter $\boldsymbol{p}_i = [x_i, y_i, z_i]^T$ identifies the position of the i^{th} *scout* particle used by the algorithm to perform obstacle avoidance, $\boldsymbol{p}_{goal}$ is the way-point given by PSO^{nav} and n_{oav} is the number of *scout* particles.

5.2.2.1 Obstacle Detection

In the simulated environment, we have used for simplicity spherical obstacles $\mathcal{O}_j$, where $j \in [1 \cdots n_o]$. Moreover we have assumed that each vehicle is equipped with a spherical



(a) (b) (c)

Figure 5.3: A circular collision area is produced when an AUV detects an obstacle (5.3a), such as the related virtual patch (5.3b) created as constraint, in the search space. Each virtual patch is defined by a 2D-Gaussian probability distribution that helps $\mathcal{SA}^{oav}$ in the computation of PSO^{oav}.

proximity sensor device that detects obstacles in surrounding space. When a vehicle $\mathcal{V}_i$ detects an obstacle $\mathcal{O}_j$, it appends that the sensor sphere $\mathcal{S}_i$ intersects it in a circular section $\mathcal{C}_{i,h} = \mathcal{S}_i \cap \mathcal{O}_j$ (Fig. 5.3a). This *section*, centred in (x_0^p, y_0^p), is used to create a *virtual patch* $\mathcal{P}_{i,h}^{\mathcal{C}}$ (Fig. 5.3b) in the search space of $\mathcal{SA}^{oav}$ with the same positions of $\mathcal{C}_{i,h}$. Indexes i, h represent respectively the vehicle which detects the collision and a time-marker that identifies each patch during the *exchanging information phase* later explained. The patch data-set stored in unit's internal database consists of several elements: the point (x_0^p, y_0^p), the set of angles $[Pitch,\ Yaw]$ of the section's orientation (respect to ${}^oF$) and the collision's probability data. Indeed, each patch is characterized by a collision probability function, described with a Gaussian distribution (see Fig. 5.3b) as follows

$$f(x^p, y^p) = A\ e^{-\left(\frac{(x^p - x_0^p)^2}{2\sigma_{x^p}^2} + \frac{(y^p - y_0^p)^2}{2\sigma_{y^p}^2}\right)}$$

where x^p, y^p are the coordinates of the points belonging to the intersection plane which incorporates $\mathcal{C}_{i,h}$. Considering a probability threshold $c_{th} = 0.5$, we have tuned Gaussian function's parameters $\sigma_{x^p}, \sigma_{y^p}$ and A, in order to adapt the function to the $\mathcal{P}_{i,h}^{\mathcal{C}}$ border, in correspondence of c_{th}. This way, it has been created a plateau of values whose collision is certain and thus we have considered the points with coordinates x^p, y^p, belonging to the patch $\mathcal{P}_{i,h}^{\mathcal{C}}$ whose value is $f(x^p, y^p) \in [0.5 \dots 1]$, as valid collision points. In this fashion $\mathcal{P}_{i,h}^{\mathcal{C}}$ becomes a *virtual wall* in the search space. The procedure is repeated for all the obstacles encountered. Roughly speaking, we can

consider this procedure like throwing a balloon full of fluorescent paint against a wall in the dark. After several launches (i.e. collision detections) a virtual wall made of fluorescent patches appears. So obstacles become visible and it is possible to plan the right path (Fig. 5.4b).

5.2.2.2 Obstacle Avoidance

As we have explained before, obstacle avoidance is directly performed by the lower PSO control level. Each patch is created inside the $\mathcal{SA}^{oav}$ search space and this modifies AUV's movements as follows. $\mathcal{SA}^{oav}$ computes PSO^{oav} algorithm by using n_{oav} particles called *scout*-particles. When it generates the next particle positions $p_i^{oav}(k+1),\ i = 1 \dots\ n_{oav}$ exploiting (3.2), a proper *validation procedure* checks if the segments that connect current with future particle positions cross an existent patch. If so, the algorithm recovers collision values of all the cross-points from the related patch data-sets. New positions are considered valid if the collision probability values of the related cross-points are below c_{th}. Only valid particles are considered by $\mathcal{SA}^{oav}$ that performs PSO^{oav} computation over $n_k = 10$ iteration steps. Among all the positions reached by the particles during this session, the one with best fitness value is selected as global "low-level best" (see Fig. 5.3c), that is useful to VCU's way-point navigation. The n_k value has been empirically chosen as best trade-off between computational speed and good results. During AUV's flocking, many different collisions can be detected, therefore the sum of all the patches $\sum_{i,h} \mathcal{P}^{\mathcal{C}}_{i,h}$ creates a *distributed wall structure* that limits particle movements in search space and AUV's movements in real space. Moreover, during the flocking, team-mates are perceived like obstacles but AUVs' positions are shared in communication's data-flows, so that a simple check on inter-vehicle distances, performed by connected neighbours, allows them to discard team-mates from static obstacles. Other units are only considered by $\mathcal{SA}^{oav}$ to perform the on-line obstacle avoidance but they are not stored in internal database. PSO^{nav} also helps to avoid obstacles: the same *validation procedure* that acts on next positions of the particles belonging to PSO^{oav} is exploited to select a valid high-level free-of-collision way-point. Finally notice that $\mathcal{SA}^{oav}$, basing its computation on scout particle's always available data, exploits a standard structure version of PSO algorithm, that is different from the asynchronous and decentralized one of PSO^{nav}.

5.2.2.3 Local Minima Avoidance

The presence of local minima is always a problem because a single AUV or all the swarm could remain stuck in the same position for a long time, loosing the possibility to achieve final target. When a robot detects a possible local minimum, suggested by the long time elapsed in the same position far from the target, a *virtual spherical patch* $\mathcal{P}^{\mathcal{V}}_{i,h}$ is produced by the stuck AUV on its position. This patch affects the search space, creating a virtual obstacle with the same benefit of the 2D patch previously described. This way, the dangerous zone is avoided by the remaining team-mates and the good result given by this position, in term of fitness value, is cancelled. So the agent possibly entrapped in a local minimum can follow its better fitness values outside the virtual patch area or it can be dragged out by the team-mates' data contribution.

5.2.3 Exchanging Information

As already mentioned, from a global point of view, $\mathcal{SA}^{nav}$ runs by using a completely decentralized version of PSO algorithm. This way each vehicle is completely autonomous. In order to permit the algorithm's correct behaviour, information exchanged between vehicles is very important. As described in Sec. 3.2, the communication topology adopted in this work allows agents to exchange data with any other close agents, meanwhile distributing information on environment. Any information exchanged have a time-stamp to identify them in temporal flow. Briefly we can sum up the set of exchanged information between two rendez-vous agents as: 1) current positions of the known AUVs; 2) individual global best value; 3) collision patch information. Taking into account for example a two team-mate rendez-vous, respective current positions recorded at each n_k time step are exchanged and stored in a proper table with time-stamp and vehicle's owner number. Moreover, the two vehicles do not only exchange their two data-sets but also past data-sets, deriving from previous exchanging with other vehicles. This way a vehicle's data can be viewed as broadcasted on vehicles' network established during every rendez-vous. As previously described, individual *global* best value is provided by PSO^{nav} algorithm of each vehicle during the flocking and it is shared in the same fashion of position's data. $\mathcal{SA}^{nav}$ that receives this value, it uses this p_i^* data as a neighbour value useful to compute PSO^{nav}. Data belonging to other vehicles are updated whenever two or more vehicles have a match. The data managing

background routine computes the same sharing procedure on collision patches' data. Each patch individually collected by each AUV is shared with all the connected team-mates. Update cycle is continuous and information run on established network. Finally, the last procedure executed by control algorithm is the merging of all the received data, in which each unit collects the most recent among the received data. To conclude, each unit updates environmental information with "best zones" and team-mate positions. Ideally, the updates of environmental data are broadcasted to the entire swarm and it is possible to consider a sort of *shared collective memory* used by all the units. Notice that a long-lived connectionless situation can produce a warp perception of the environment by unconnected units and a delay on task accomplishment. In our simulations we have detected that in order to reach the target point at least two agents must be connected.

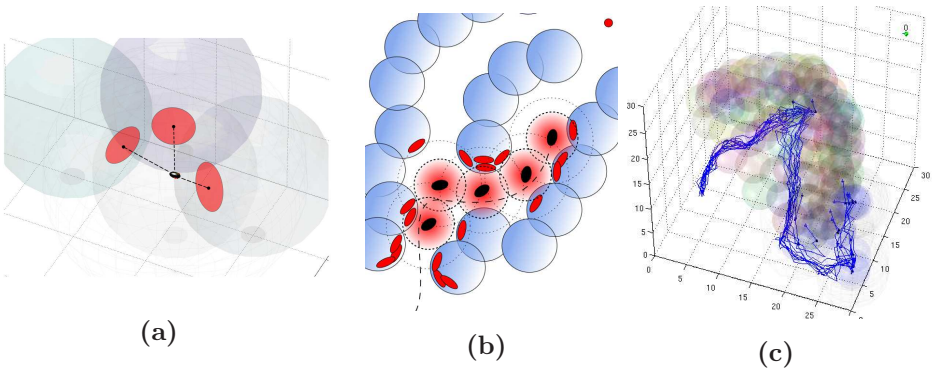
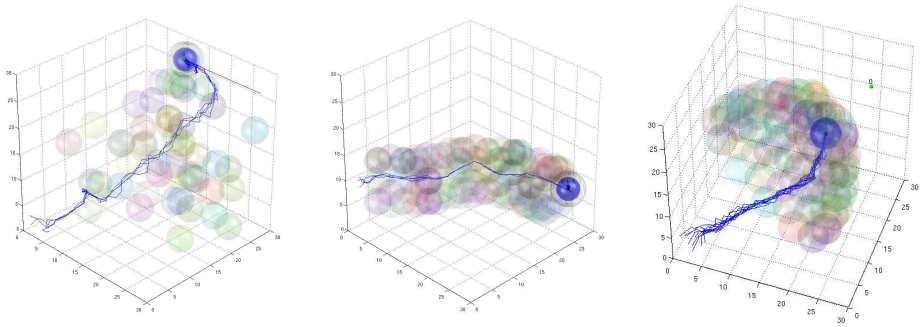


Figure 5.4: Each AUV, detecting an obstacle, creates a shared constraint that helps the other team-mates to find the best path, in order to achieve the final target.

5.3 Simulations

The presented PSO algorithm has been intensively tested in a Matlab 3D simulated environment. In particular, we have considered an environment volume whose dimensions range in $[30 \dots 300]$ $[m^3]$ and three different types of scenarios: a map with obstacles randomly disposed (Fig. 5.5a), different simulated tunnels (Fig. 5.5b) and caves (Fig. 5.5c). In order to maintain the simulated physical parameters closer to real devices [55], each unit has been modelled as an ellipsoid whose dimensions are

$[0.7 \times 0.4 \times 0.3]$ $[m]$, maximum velocity $u_{max} = 5$ $[m/s]$, four fins with equal opening speed $w_{max} = 0.07$ $[rad/s]$, max fin's opening angle $\delta_a = 0.26$ $[rad]$, sensor range $r_S \in [5 - 25]$ $[m]$ and communication range $\Delta \in [10 - 50]$ $[m]$. A sample time of $\Delta t = 0.02$ $[s]$ has been selected for the controller. Notice that the *virtual* time step Δt_v perceived by the high-level agent $\mathcal{SA}^{nav}$ is equal to 0.2 $[s]$ due to the fact that the computation session of PSO^{oav} runs over $n_k = 10$ time steps. The number of scout particles $n_{oav} = 5$. The first scenario (e.g. Fig. 5.5a) has been created by using obstacles modelled as spheres of arbitrary radius $r_{\mathcal{O}} \in [0.5 \cdots 5]$ $[m]$ randomly deployed in the middle of the testing volume. On the contrary, in the second and in the third ones the sphere's radius is fixed at $r_{\mathcal{O}} = 2.5$ $[m]$ but the dimension of the built structures has been changed. In the second scenario (e.g. Fig. 5.5b) a simulated tunnel has been created composing a different number of spherical obstacles disposed along a spline. The aperture radius of the tunnel r_a^t in the different simulations varies in $[4 \dots 8]$ $[m]$. The last type of scenario (Fig. 5.5c) tests the algorithm in a simulated cave whose dimensions are WxHxD = $[14,\ 10,\ 15]$ $[m]$.

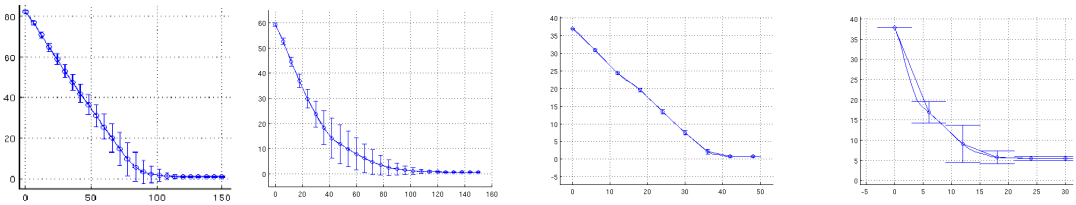


(a) **(b)** **(c)**

Figure 5.5: The presented technique is suitable to drive the AUV swarm in different scenarios, from the one with only dispersed obstacles to the heavily constrained one. The target can also be mobile (5.5a.).

5.3.1 Statistical Results and Parameter Selection

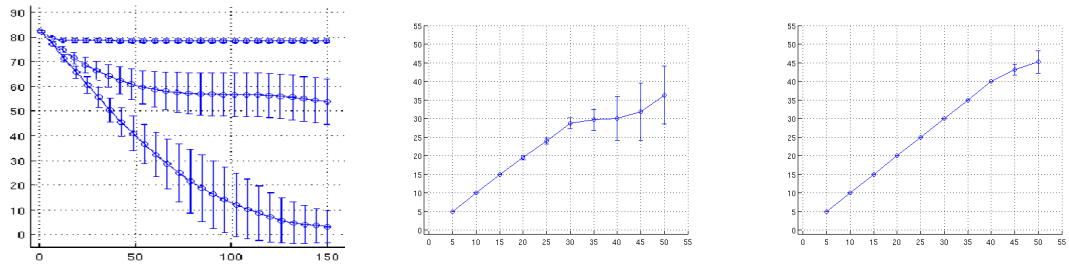
In order to gather statistical results about the performance of the proposed PSO algorithm, we have performed hundreds of simulations for each scenario. In each simulation of the first scenario obstacles' positions and dimensions have been changed. Despite of



(a) (b) (c) (d)

Figure 5.6: All the figures represent the *mean* of the units' distances between the respective start and goal areas. Statistical results for scenario 1 are depicted with respect to a fixed and a mobile target (5.6a, 5.6b) while data related to scenarios 2 and 3 are presented in 5.6c and 5.6d respectively.

this, it is possible to notice in Fig. 5.6a that AUVs are able to overcome obstacles and achieve the target even in case the target is moving, as reported in Fig. 5.6b mobile target velocity $\dot{\boldsymbol{p}}_{goal}^{nav} = [-4, 0, 0]\ [m/s]$. The enlargement of the standard deviation in the middle of the statistic graphs, describes the various path followed by the vehicles to overcome obstacles. Fig. 5.6a shows the data related to $\varphi = 3.5$ in scenario 1, that produces the best algorithm's performances. This choice has been made on the basis of the simulations made by changing $\varphi = 1, 1.5, 2$, depicted in Fig. 5.7a. From top to bottom it is possible to notice how performances increase. When $\varphi > 3.5$ randomness is too high, standard deviation increases and swarm's behaviour is not controllable any more, so units do not converge. This φ value has been adopted for all three scenarios. We have considered swarms with a number of units ranging from 2 to 50. From



(a) (b) (c)

Figure 5.7: Swarm's behaviour is closely related to parameter values and to environmental conditions. As depicted in Fig. 5.7a the number of robots that achieved the target in scenario 1 changes with the different values assumed by the φ parameter. From top to bottom $\varphi = 1, 1.5, 2$. The same test has been made on scenario 2 changing tunnel's radius from $r_a^t = 4$ (5.7b) to $r_a^t = 8$ (5.7c).

these case-studies, it appears that in case there is only one vehicle, the target is not reached even in simple scenarios, the information on the environment is too poor and the random part of the particle's velocity provides too much noise, making the accomplishment of the task difficult. We have not reached good results with a single vehicle, even tuning the random parameters in different ways. Moreover simulations show that there are some problems with robots' number in scenario 2, indeed using more than 25-30 robots creates too many collisions and the algorithm fails to maintain the group compact. In this case, the swarm breaks up into smaller groups that reach the target autonomously. By enlarging the tunnel's aperture this problem tends to disappear but it reoccurs increasing the number of units.

5.4 Conclusions and Future Work

In this chapter, a novel version of the Particle Swarm Optimization algorithm has been proposed in order to drive a group of AUVs in unknown underwater environments from a starting point to a final one, that can be possibly mobile. In particular, we have used a double-level and a completely asynchronous, decentralized version of PSO to manage swarm's coordination. Moreover, PSO algorithm has been modified in order to consider robots' physical constraints, i.e. their dimensions and their interaction with the environment. Vehicles know their own position, and they are able to broadcast data to neighbours and to sense the surrounding environment by using spherical sensors. During the navigation, units collect data on the environment not only to reach the target but also to map the environment. This technique could be useful for example in benthic exploration or search and rescue. The performances of the algorithm have been validated using the data collected in several simulations, where different groups of AUVs have been simulated in many complex environments. Future work will include deeper analysis of the relation between environmental features, parameters' optimization and performances. We will also study the possibility of controlling team-mates' distances according to environmental conditions. Moreover, the possibility of using *dynamic patches* to map both moving and evanescent obstacles, exploiting proper predictive algorithms will be considered. Finally, applications of this algorithm to more complex systems, like groups of heterogeneous robots (e.g. groups of robots including both naval and aerial vehicles) and to more complex environments will be considered.

5. A DISTRIBUTED MULTI-LEVEL PSO CONTROL ALGORITHM FOR AUTONOMOUS UNDERWATER VEHICLES

6

A Hybrid Technique for Controlling Platoons of Mobile Robots with PSO and Consensus

A hybrid technique used to control swarms of robots, which is based on both Particle Swarm Optimization (previously introduced in Sec. 3.2) and Consensus (whose details can be found in [14]), is presented in this chapter. The purpose of this hybrid technique is to combine PSO's adaptation skills in unknown environments' exploration and the ability of the Consensus to maintain a group of robots in a desired formation as explained in [14, 56, 57]. As presented in the related sections, both techniques are designed in a distributed form. This fact is particularly suitable for the control of a swarm of robots where one or more robots can experience temporary or permanent disconnections from the group due to various reasons (e.g. malfunctions).

6.1 Introduction

Formation control is a very well studied problem, and many different approaches can be found in the literature. Different approaches can be divided into those that use a centralized technique from those that apply a decentralized one. In a real Mobile Robot System contest, thinking to manage all the robots from a unique source that elaborates the huge information flow coming from all units and that coordinates the system, is not reliable. Using an algorithm like Consensus, that works in a decentralized

fashion [58], exploiting agents' interconnections given by the communication topology trough graph theory [10, 59, 60], even in presence of communication delays or switching [61, 62], it seems to be the right way, as described in [13, 63]. Consensus has been applied in many different fashion and environment, free [64] or constrained [65]. It has been applied to many kinds of robot systems, not only to Unmanned Ground Vehicles (UGVs) that are normally used to test this kinds of techniques, but also to Autonomous Underwater Vehicles (AUVs) as presented in [47], Unmanned Air Vehicles (UAVs) and Satellites [66]. Consensus concerns the idea of agreement and synchronization [67] between agents in different fashions. When the purpose is to achieve formation control, it is possible to a-priori fix the inter-agent distances or, as reported in [56, 57], it is possible to dynamically change and control them in order to obtain soft or rigid formations on-demand. This allows to regulate the platoon's geometry, with the aim to overcome obstacles or narrow passages without loosing the connections between agents. In presence of obstacles in the environment it is possible to exploit virtual potential fields [63] or to project virtual nodes and to create virtual edges [68] in correspondence of the detected obstacles' positions. This kind of approach permits to embed obstacles inside the graph's structure and to take properly advantage of the decentralized control's behaviour of Consensus avoiding them in an auto-regulated fashion. In the technique presented in this chapter, knowledge and methods from Chp. 4, Chp. 5 and [14, 56, 57] have been exploited in order to take advantage of PSO's features as a swarm navigation system and, on the other side, to maintain the units in a desired and controlled formation through the application of the agreement protocol performed by Consensus. This approach has been followed with the aim to conserve the power of PSO and to control part of its random behaviour with a distributed control algorithm. The chapter is structured as follows: the structure of the algorithm, the description of the simulated devices that provides sensors data and communication, the obstacle avoidance technique and the interconnection between PSO and Consensus are described in Sec. 6.2. Moreover the simulations are presented in Sec. 6.3. Final conclusions and future work are reported in Sec. 6.4.

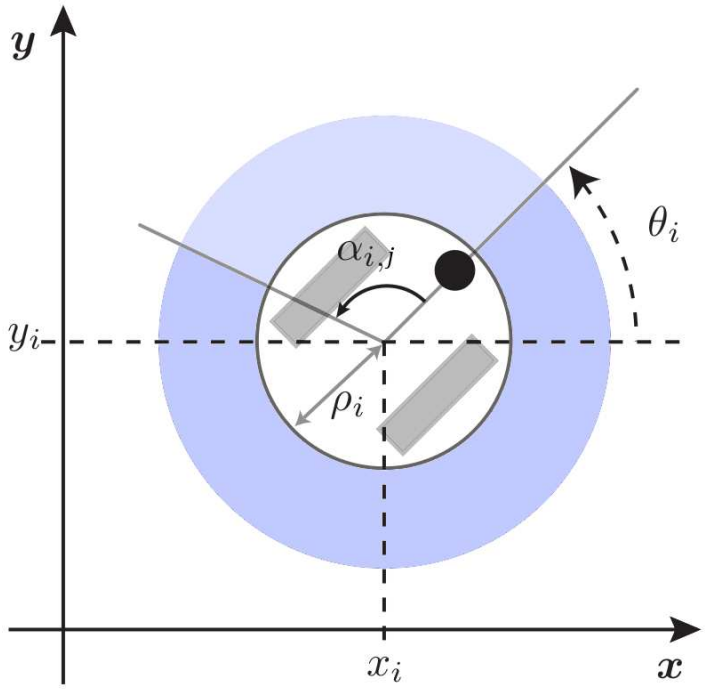


Figure 6.1: Kinematic model of the robots with the state variables. The blue ring represents the sensor's area defined by the 36 proximity sensors equally spaced around the robot, whose sensible distance and quantization can be freely configured. The parameter $\alpha_{i,j}$ defines the angle of the j^{th} sensor.

6.2 The Algorithm

The algorithm presented has been developed with the aim to drive a swarm of differential wheeled mobile robots $\mathcal{R}_i$, $i = 1, \ldots, n$ moving in unknown planar 3D environments. Each vehicle is autonomous: this means in particular that all the computation is executed individually by each software agent ($\mathcal{SA}_i$) that controls its own robot, and that all the decisions are made on the basis of the data locally available at the k^{th} step. The exchanged information, coming from the on-board devices (sensors, communications and positioning system) are elaborated by each agent, with the purpose to reach a final point, both maintaining the connection with the other team-mates in a deformable formation and avoiding obstacles on the robot's path.

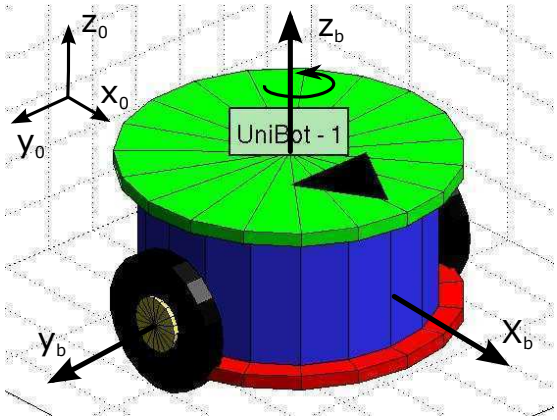


Figure 6.2: The base ${}^{o}F$ and the body ${}^{b}F_i$ frame.

6.2.1 Model of the Robot and Mathematical Tools

In order to obtain the robot's model used in this technique we have defined a body reference frame ${}^{b}F_i = \{x_{b,i}, y_{b,i}, z_{b,i}\}$ for each vehicle, and a common inertial (base) frame ${}^{o}F = \{x_0, y_0, z_0\}$ useful to describe the robot's position with respect to the simulation environment. The kinematic model of the robot has already been presented in Chp. 4. For convenience it is reported below, remembering that it is defined with respect to ${}^{o}F$

$$\begin{aligned} x_i(k+1) &= x_i(k) + u_i(k)\cos(\theta_i(k)) \\ y_i(k+1) &= y_i(k) + u_i(k)\sin(\theta_i(k)) \\ \theta_i(k+1) &= \theta_i(k) + \omega_i(k)\Delta t \end{aligned}$$

where $u_i(k), \omega_i(k)$ are the forward and the steering control velocities of the robot during the time step $[k,\ k+1]$ respectively and $\theta_i(k)$ is the Yaw angle with respect to z_0. Notice that $z_i(\cdot) = 0$ for any k-step.

Also remembering the things described in Sec. 5.2.1.1 we report the rotation matrix

$$\mathbf{R}_{zyx}(\theta,\psi,\alpha) = \mathbf{R}_z(\theta)\mathbf{R}_y(\psi)\mathbf{R}_x(\alpha) = \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi s_\alpha - s_\theta c_\alpha & c_\theta s_\psi c_\alpha + s_\theta s_\alpha \\ s_\theta c_\psi & s_\theta s_\psi s_\alpha + c_\theta s_\alpha & s_\theta s_\psi c_\alpha + c_\theta s_\alpha \\ -s_\psi & c_\psi s_\alpha & c_\psi c_\alpha \end{bmatrix}$$

that represents the relative orientation between mobile and fixed frame, where $c_a = \cos(a), s_a = \sin(a)$. The matrix is useful to define the homogeneous transformation between ${}^{o}F$ and ${}^{b}F$ as

$${}^{o}\mathbf{T}_b = \begin{bmatrix} {}^{o}\mathbf{R}_b & \boldsymbol{x}_b \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{6.1}$$

and the related inverse homogeneous transformation matrix

$$ {}^{b}\mathbf{T}_{o} = \begin{bmatrix} {}^{o}\mathbf{R}_{b}^{T} & -{}^{o}\mathbf{R}_{b}^{T}\boldsymbol{x}_{b} \\ \mathbf{0}^{T} & 1 \end{bmatrix} \tag{6.2} $$

where $\boldsymbol{x}_b$ is the position vector of ${}^{b}F$ with respect to ${}^{o}F$ and ${}^{o}\mathbf{R}_{b} = \mathbf{R}_{zyx}(\theta, \psi, \alpha)$. The parameters θ, ψ, α are the Yaw, Pitch and Roll angles respectively, with respect to the base frame ${}^{o}F$. Being the simulation computed in a planar 3D environment and having z_0 the same orientation of $z_{b,i}$ we have assumed $\psi = 0, \alpha = 0$. Finally we define a simple transformation from Cartesian to spherical coordinates given by

$$ \begin{cases} \phi_d &= atan2(y_d, x_d) \\ \xi_d &= atan2(z_d, \sqrt{x_d^2 + y_d^2}) \\ \rho_d &= \sqrt{x_d^2 + y_d^2 + z_d^2} \end{cases} \tag{6.3} $$

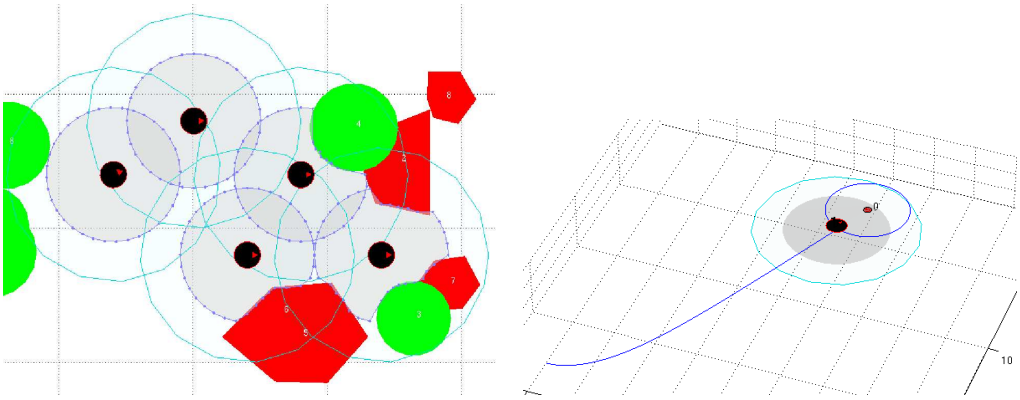
where ϕ_d (desired azimuth) and ρ_d (desired distance) are values used to compute the correct *viewing angle* of a desired point $\boldsymbol{x}_d = [x_d, y_d, z_d]$. Being a planar simulation the value ξ_d is always zero. The unusual representation of polar coordinates has been used with the purpose to easily define the right and the left side from the robot's point of view, in relation with the azimuth value. The inverse transformation is given by

$$ \begin{cases} x_d &= \rho_d \, cos(\xi_d) \, cos(\phi_d) \\ y_d &= \rho_d \, cos(\xi_d) \, sin(\phi_d) \\ z_d &= \rho_d \, sin(\xi_d) \end{cases} \tag{6.4} $$

More details are provided in the following sections.

6.2.2 Simulation Environment

As for the technique presented in Chp. 5, simulation environment is provided by M.U.S.E. As described in Sec. 9.2, M.U.S.E. is a 3D simulator designed with the purpose to provide support for the control algorithm developed for Unibot (see Sec. 7.1). As depicted in Fig. 6.3, it has the possibility to work with 2D or 3D simulated elements. M.U.S.E. provides not only the graphical view of the simulated robots and obstacles but also collisions detection in the robot-to-robot and robot-to-obstacle interactions. Each robot is equipped with Proximity Sensors and a Communication System later described in the related sections.



(a) (b)

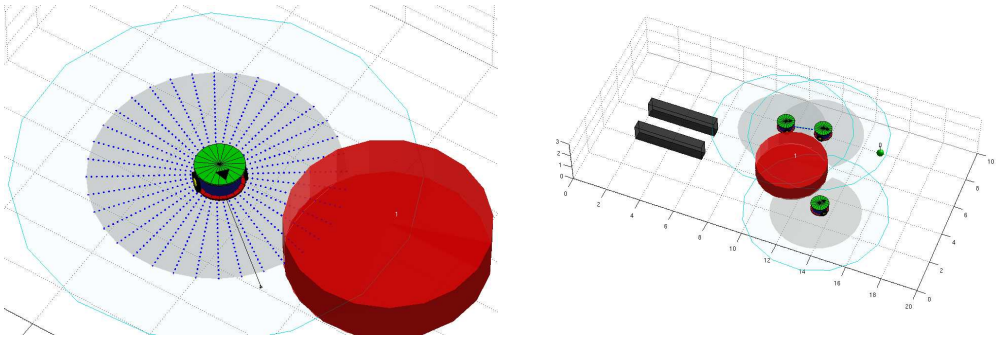
Figure 6.3: Obstacle detection is performed by sensors' device, providing the agent the possibility to perform obstacle avoidance while the agent is still computing the right control action to reach the final target. Notice that the sensor's area is clearly deformed by the presence of obstacles (6.3a).

6.2.3 Sensor Device

Each robot is equipped with a Sensor Device able to detect obstacles in the surrounding environment and able to provide the related distance to $\mathcal{SA}$. Each device is equipped with 36 proximity sensors equally spaced around each robot. The sensible distance is freely configurable as well as the quantization of the sensors' area. In Fig. 6.3a, the sensors' area of each robot deformed by collisions with obstacles is depicted, while in Fig. 6.4a the sensors' cloud during a simulation is shown. Notice that the area is linearly quantized but it is possible to chose any other function to define sensor distances. The sensor cloud is visible only in debug mode as well as the collision ring visible in Fig. 6.3a.

6.2.3.1 Obstacle Avoidance

The Sensor Device exploits data coming form sensors' cloud to project a circular Gaussian Field around the robot (see figure-set 6.5). On the basis of each sensor's value a summation of Gaussian function, where the expected value μ is calculated in correspondence of the sensor's angle value, is shaped around the robot, similarly to what described in Chp. 4. Each robot computes the relative angle $\vartheta_{o,i}$ where the probability



(a) (b)

Figure 6.4: The sensible points' cloud projected around the robot by the sensor device, is exploited to detect the distance from obstacle (6.4a). Normally it is only visible the sensible area while the point cloud is hidden. In the debugging mode it is possible to show the cloud. Moreover, when the robots move around, they have the possibility to communicate each other, indeed an obstacle avoidance sequence is shown in Fig.6.4b where communication's device action is visible. The two upper robots are connected by a blue line while the bottom one is not connected because its position is outside the communication range (i.e. the blue ring surrounding each robot).

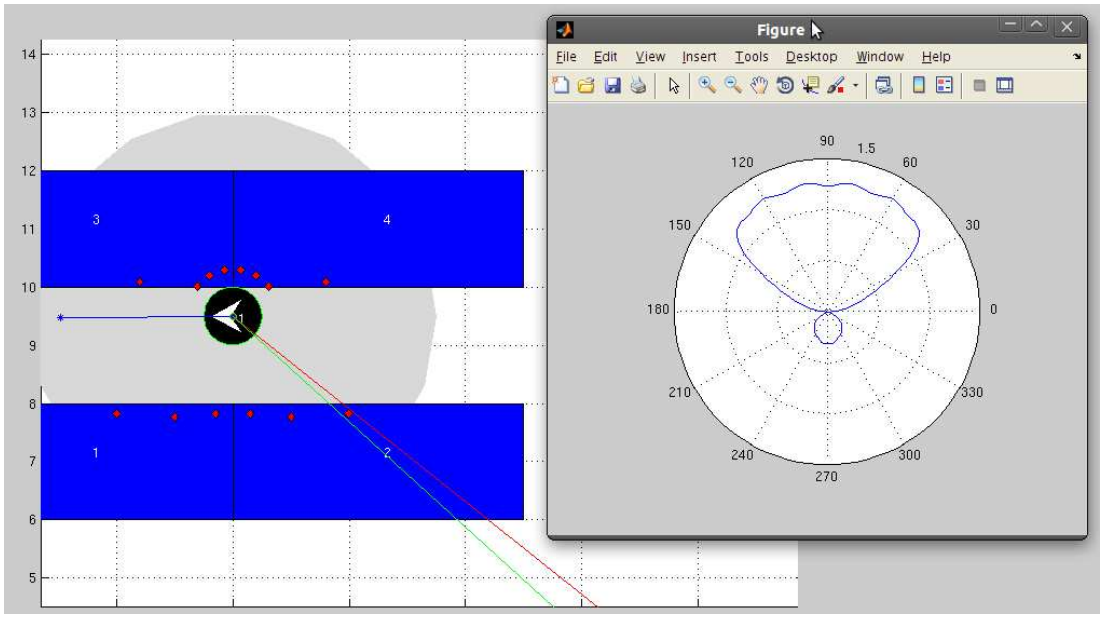
of colliding with an obstacle is minimal

$$\vartheta_{o,i} = \min(\varphi_{o,i}), \qquad \varphi_{o,i} = \sum_{h=1}^{N_b} b_h \; g(\beta, \sigma_{h,i}, \alpha_{h,i}) \tag{6.5}$$

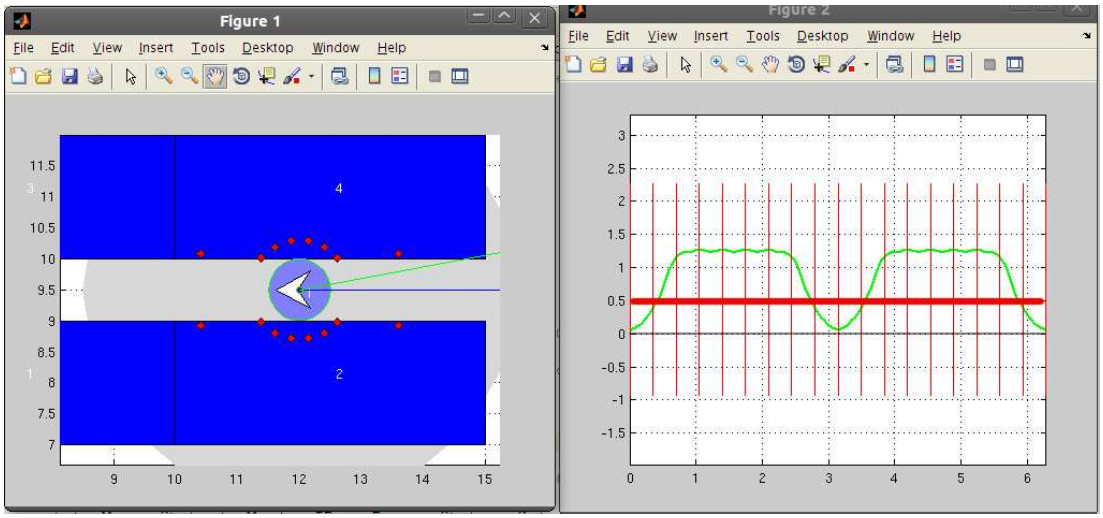
where $\beta \in [0 \dots 2\pi]$, b_h depends on the inverse of the h^{th} sensor's value, which is normalized with respect to the quantization function chosen, and $g(\cdot)$ is the normal Gaussian distribution centred on $\mu = \alpha_{h,i}$ defined by

$$g(\beta, \sigma_{h,i}, \alpha_{h,i}) = \frac{1}{\sigma_{h,i}\sqrt{2\pi}} \; e^{-\frac{(\beta - \alpha_{h,i})^2}{2\sigma_{h,i}^2}}$$

where the variance $\sigma_{h,i}$ can be used to define how each sensor affects the robot's perceptions. In this technique, in a different fashion from what has been done previously, the minimum value of the Gaussian Field is calculated exploiting the Particle Swarm Optimization, as described in more detail later, and the related value ϑ_{best} is provided to the Sensor Device for the Dynamic Windows computation. In case more than one value is detected by PSO, the one close to the robot's front is selected.



(a)



(b)

Figure 6.5: The Gaussian Field is used to detect obstacles and to perform obstacle avoidance. It is based on a Gaussian mask applied to the activated sensors when the robot encounters obstacles. The field can be represented as a continuous distance function on which the minimum values are calculated. The red line (visible in 6.5b), defined as the *valid threshold*, is the limit under which it is possible to consider the angle's aperture as free of collisions.

6.2.3.2 Dynamic Window approach

Similarly to Chp. 4 a Dynamic Window approach has been used to determinate the correct escape (virtual) angle a_v during the obstacle avoidance. In correspondence of the minimal value detected by PSO exploiting (6.5), namely ϑ_{best}, a dynamic window with a variable aperture is created, whose value, experimentally defined and related to the robot's body dimensions, is $\gamma \subseteq [10 \dots 120]$ [deg] range. The window's aperture γ is correlated by the *Gaussian Field* valley's aperture in correspondence of ϑ_{best} and is defined as the maximum angles-range available, free of collisions so that $\vartheta_{best} \in \gamma$. The virtual angle is simply defined as $a_v = \gamma/2$. In Fig. 6.5b $a_v = 0$.

6.2.4 Communication Device

As previously mentioned, each robot is equipped with a communication device, whose activity is shown in Fig. 6.4b during an obstacle avoidance sequence. The two upper robots are connected by a blue line while the bottom one is not connected because its position is outside the communication range (i.e. the blue ring surrounding each robot). Moreover, the communication device stores in its internal database the positions of the robots encountered in the simulation, during the last m-time-steps, where m is a parameter that defines the device's memory depth. In this works it has been chosen $m = 3$. This helps $\mathcal{SA}_i$ to better understand the formation's evolution during the simulation. Moreover, the communication device, on the basis of the connected robots, computes the local Laplacian matrix useful to calculate the Consensus agreement protocol.

6.2.5 The Control Algorithm

The aim of the algorithm is to maintain a desired robots' formation during the navigation from a starting area to a final one avoiding obstacles. In order to achieve this target, at the beginning of the simulation, the algorithm calculates the final destination for each robot and the connections' matrix useful to create the connection graph and to compute the Consensus protocol.
When the final destination is given to each $\mathcal{SA}_i$ and the connected team-mates are chosen, each robot agent starts to compute the algorithm depicted in Fig. 6.6. The main control procedure begins with the computation, by the sub-procedure *CalculateRealTarget*, of a surrogate target point that takes into consideration the possible presence

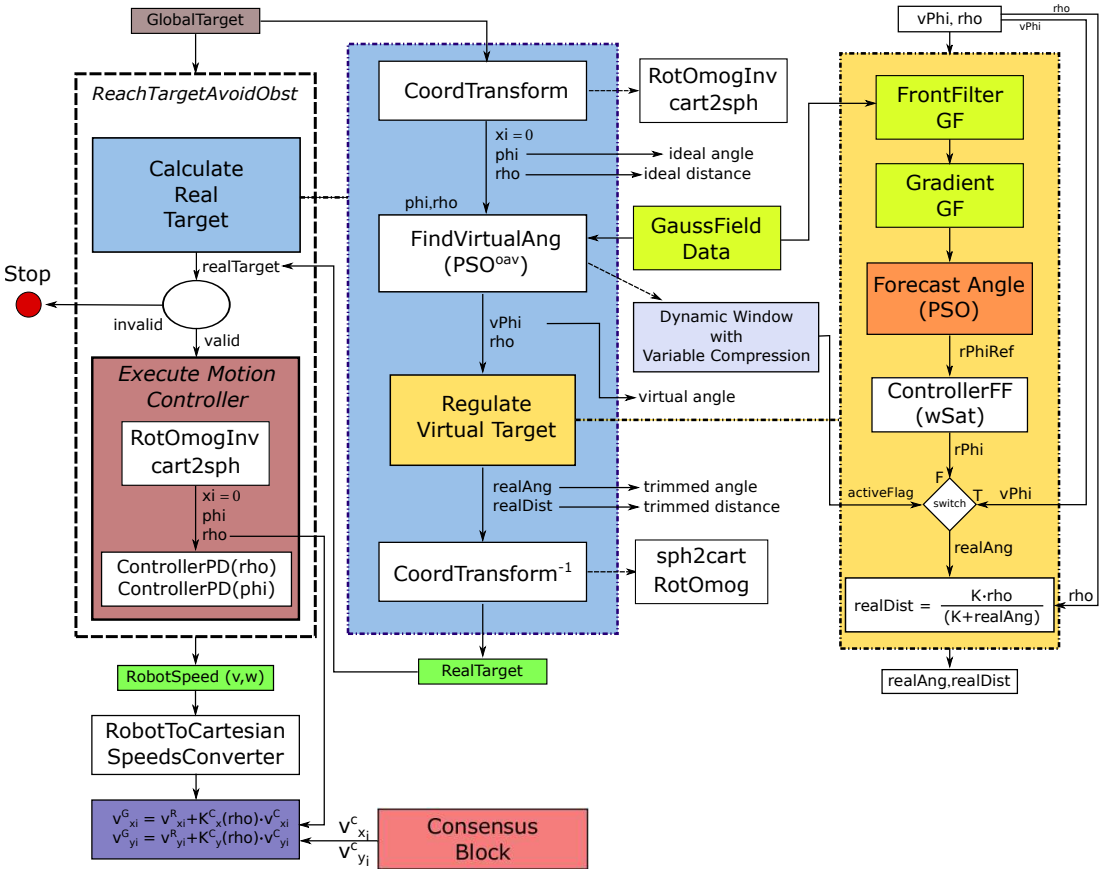


Figure 6.6: The global control scheme of the algorithm, from the robot's (agent's) point of view, is useful to better understand all the operations computed by the control system. Moreover it is possible to understand the interconnections between "classic" and modern control techniques as PSO and Consensus.

of obstacles. The sub-procedure starts with the transformation of the *global target* coordinates ${}^{o}\boldsymbol{x}_i^g = (x_i^g, y_i^g, 0)$ with respect to the base frame ${}^{o}F$, acquired by the agent, into polar coordinates, which are measured with respect to the mobile frame ${}^{b}F_i$. By using (6.1) and (6.3) the global target coordinates are transformed as follows

$$\begin{aligned} {}^{b}\boldsymbol{x}_i^g &= {}^{b}\mathbf{T}_o \; {}^{o}\boldsymbol{x}_i^{gT} \\ \phi_d &= atan2({}^{b}y_i^g, {}^{b}x_i^g) \\ \rho_d &= \sqrt{{}^{b}x_i^{g2} + {}^{b}y_i^{g2} + {}^{b}z_i^{g2}} \end{aligned} \tag{6.6}$$

where the obtained values (ϕ_d, ρ_d) represent the desired *ideal* distance and angle of the global target in a *robocentric* view. The data are filtered by the *FindVirtualAngle*

block, which operates exploiting Gaussian Field Data in order to find the right virtual angle as described in Sec. 6.2.3.2. The *virtual target*, identified by the couple $({}^{v}\phi_d,\ {}^{v}\rho_d)$, where ${}^{v}\phi_d = a_v$, is not ready to be used as a reference for the *MotionController* block yet, that moves the robot toward the final destination. The virtual target needs a regulation procedure, which is operated by the *RegulateVirtualTarget* block (identifiable by the light-orange color block in Fig. 6.6).
Defying the front of the robot as the angle-set belonging to the range $[-\pi/4 \dots \pi/4]$ centred on 0 just in front of the robot, the regulation block provides a distance and angle adjustment: by using the Gaussian Field Data related to the robot front, it calculates the gradient of the Gaussian Field on the front and finds the minimum value filtering the result trough a second PSO block. The result of this procedures' chain is the data ${}^{r}\phi_{ref}$ called *forecasting angle*. To clarifying the concept, when a robot moves and encounters an obstacle, the gradient of the Gaussian Field suggests the direction to follow in order to anticipate the Dynamic Windows procedure. The forecasting angle is used as reference angle step in a linear feed-forward controller

$$ {}^{r}\phi = \boldsymbol{K}_p\, {}^{r}\phi_{ref} \Delta t $$

The value ${}^{r}\phi$ is saturated at 10 degs with the purpose to maintain the forecasting angle controlled, independently by the obstacle's geometry. In case the compensation, given by the forecasting angle, is not enough in order to avoid obstacle and the Dynamic Window is activated because the robot moved too close to the obstacle, the *Switch* block deactivates the feed-forward controller and the resulting value is substituted by the unregulated previous value ${}^{v}\phi_d$. The final obtained value, at the output of the *Switch* block, namely *real angle* or ϕ_{real}, is used to regulate the virtual distance ρ_d simply as follows

$$ \rho_{real} = \frac{K\,\rho_d}{K + \phi_{real}} $$

where K is a proper coupling constant that specifies how much the angle value affects the distance value. The couple $(\rho_{real}, \phi_{real})$ is used to determinate the linear velocity and the steering velocity in the subsequent procedures. It is clear that when the

steering angle (ϕ_{real}) grows the real distance (ρ_{real}) decreases, following what normally happens in car driving: in order to maintain the control during a strict curve the speed is decreased.
By using (6.2) and (6.4) the absolute real target ${}^{o}\boldsymbol{x}_i^{real} = [{}^{o}x_i^{real}, {}^{o}y_i^{real}, 0]$ is obtainable from the relative one ${}^{b}\boldsymbol{x}_i^{real} = [{}^{b}x_i^{real}, {}^{b}y_i^{real}, 0]$ as follows

$$\begin{array}{rcl} {}^{b}x_i^{real} & = & \rho_{real}\, cos(\xi)\, cos(\phi_{real}) \\ {}^{b}y_i^{real} & = & \rho_{real}\, cos(\xi)\, sin(\phi_{real}) \\ {}^{o}\boldsymbol{x}_i^{real} & = & {}^{o}\mathbf{T}_b\ {}^{b}\boldsymbol{x}_i^{real\,T} \end{array}$$

This way the inverse coordinates' transformation of (6.6) is used to produce a valid target for the *MotionController* block.
As shown in the global scheme depicted in Fig. 6.6, this block uses $(\rho_{real}, \phi_{real})$ in two different linear PD-controllers after the (6.6) transformation as described below

$$\begin{aligned} u_i(k+1) = &\ K_u^D \frac{\rho_{real}(k)}{\Delta t} + K_u^P\, \rho_{real}(k) \\ \omega_i(k+1) = &\ K_\omega^D \frac{\phi_{real}(k)}{\Delta t} + K_\omega^P\, \phi_{real}(k) \end{aligned}$$

in order to compute linear ($u_i(k+1)$) and angular ($\omega_i(k+1)$) robot's velocities. By using the well known odometric methods [69] as the Exact Solution methods or the 2^{nd} order Runge-Kutta method (in case $\omega_i(k) = 0$), it is possible to estimate with good precision the robot's position and to calculate its Cartesian speeds ${}^{o}\dot{\boldsymbol{x}}_i^{R} = [v_{x_i}^{R}, v_{y_i}^{R}, 0]$. The Consensus contribution is evident at this point of the control algorithm. Indeed the speed's contribution, related to $\mathcal{R}_i$, coming from the Consensus agreement protocol and defined by ${}^{o}\dot{\boldsymbol{x}}_i^{C} = [v_{x_i}^{C}, v_{y_i}^{C}, 0]$ is added to the robot's Cartesian speeds just calculated as follows

$$\begin{aligned} v_{x_i}^{G} = v_{x_i}^{R} + &\ K_{Cx}(\rho_{real}(k))\, v_{x_i}^{C} \\ v_{y_i}^{G} = v_{y_i}^{R} + &\ K_{Cy}(\rho_{real}(k))\, v_{y_i}^{C} \end{aligned}$$

obtaining the global speed contribution ${}^{o}\dot{\boldsymbol{x}}_i^{G} = [v_{x_i}^{G}, v_{y_i}^{G}]$ from which it is possible to recalculate the new inputs u_i, ω_i useful to control the robot maintaining the connection with the team-mates. The parameters K_{Cx} and K_{Cy} are proper coupling constants useful to tune the influence of the Consensus contribution. Notice that each constant depends on $\rho_{real}(k)$ value. When the ρ_{real} value is high, it means that there are no

obstacles and the robot can perform formation control. Otherwise, when ρ_{real} is low, it means that there are obstacles and the robot needs to pay more attention to obstacle avoidance and not to the formation control, or it means that the robot is arrived close to the final destination and the formation control is not so important as during the motion.

6.2.5.1 PSO Blocks

In the presented control algorithm two different PSO control blocks have been used as optimizers, with the purpose to find the minimum value of the function associated to Gaussian Field and its gradient. This meta-heuristic technique has been adopted in order to overcome the difficulties coming from the unknown obstacle's geometric profile and its related Gaussian Field deformation. The approach used for PSO algorithm's structure is the same adopted in Chp. 5 for the PSO^{oav}. In both PSO blocks the tested number of *scout-particles* ranging from 5 to 10. Instead of using a random initial position for each particle, in this technique a well defined initial set-up has been chosen, where the positions are equally spaced on the related valid angular range. This helps to quicker find the optimal solution.

6.2.5.2 Consensus Block

As described in [14, 56, 57], the Consensus algorithm is based on Graph Theory and it implements the agreement protocol. A graph is a mathematical structure suitable to describe relations in a collection of resources. Generally in graph's terminology the relations are called edges while the resources are called nodes or vertices. Associating the robots with the nodes of the graph and the communication links between the robots with the edges of the graph, it is possible to take advantage of the mutual information shared between the connected robots, in order to create a desired formation.
At each edge could be associated a fixed or a variable weight in order to modify formation's behaviour by changing the weights. In this case the Laplacian matrix, that is the matrix that describes the graph's connections, is called *weighted Laplacian*. By exploiting the weighted Laplacian matrix defined as follows

$$\mathcal{L}_w = \mathcal{I}\,\mathcal{W}\,\mathcal{I}^T$$

where $\mathcal{I}$ represents the *incidence matrix* of the graph and $\mathcal{W}$ the edge-weights matrix, it is possible to configure Consensus in order to provide the robot's speed contribution, represented in Cartesian coordinates, as follows

$$\dot{x}(t) = \sum_{j=1}^{n} a_{ij} \left(x_i(t) - x_j(t) \right) \qquad or \qquad \dot{x}(t) = -\mathcal{L}_w \, x(t) \tag{6.7}$$

where n is the cardinality of the $\mathcal{R}_i$ neighbours set and a_{ij} represents the weight of the edge that connects $\mathcal{R}_i$ with its j-neighbours. The variables x_i and x_j are the related state variables on which Consensus operates. Defining the desired position that each robot must reach as $\boldsymbol{x}_i^f$ and as $\boldsymbol{x}_i^c$ its current position, it is possible to write the *position's error* of the i^{th} robot as $\boldsymbol{e}_i = \boldsymbol{x}_i^f - \boldsymbol{x}_i^c$. By changing the state variable in (6.7) with the position's error as

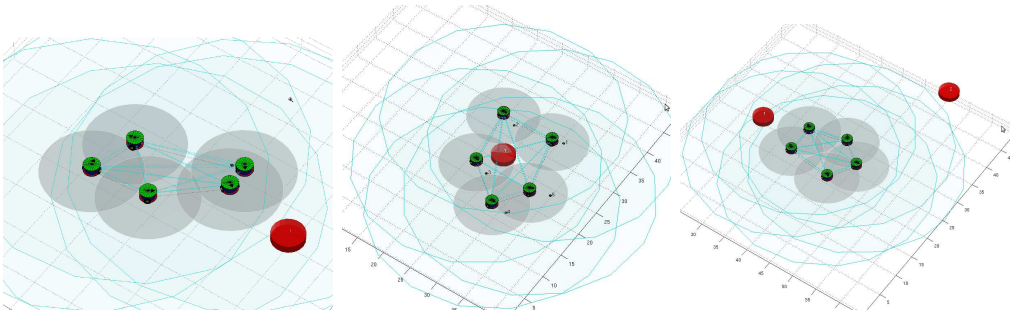
$$\dot{\boldsymbol{e}}(t) = -\mathcal{L}_w \, \boldsymbol{e}(t) \tag{6.8}$$

from which, selecting the i^{th} element of the resulting vector $\dot{\boldsymbol{e}}_i(t) = [\dot{e}_{x_i}(t), \dot{e}_{y_i}(t)]$, it is possible to define the Consensus speed contribution to $\mathcal{R}_i$ as $[v_{x_i}^C, v_{y_i}^C] = [\dot{e}_{x_i}(t), \dot{e}_{y_i}(t)]$. This way it is possible to obtain a basic control algorithm that provides formation's control.

With the purpose to have the geometric shape of the formation modifiable on-demand, in order to achieve elastic or rigid behaviour of the robot group, it is possible to change the weight value a_{ij} in relation to a specified potential function like the one used in [56]. When the edge-weight is high the influence that a robot has on it's graph's neighbours is high and it produces perturbations on their motion. Otherwise when the weight is low the robot does not affect neighbours. In order to maintain the formation as elastic as possible and stable, in this technique we have assumed $a_{ij} = 0.8$ when two robots are connected and zero otherwise. We want to remark that Consensus is a decentralized algorithm and it is computed by each $\mathcal{SA}_i$ on the basis of data provided by the Communication Device.

6.3 Simulations

Many simulations have been performed in order to validate the presented technique but due to its greenness, the number of simulations is not high enough to provide statistical



(a) (b) (c)

Figure 6.7: The soft formation control is performed by robots in order to avoid obstacles and remain connected.

results, able to demonstrate the good performances of the algorithm. However three different instants of a formation's control simulation are depicted in the figure-set 6.7. The robot group, after reaching the desired formation geometry, moves remaining in contact, opening and closing the formation in order to overcome a detected obstacle.

6.4 Conclusions and Future Work

In this chapter a hybrid technique useful to drive a group of mobile robots in an a-priori unknown environment has been presented. This technique not only drives the robots from a starting area toward a final one but it is also able to maintain the group compact with the purpose to control the related robot positions and to create a desired formation. The power of this technique comes from the simultaneous use of PSO and Consensus with the aim to exploit the better features of both. Unfortunately the greenness of this technique does not allow to permanently state the good behaviour presented during the simulations done. Future work will include many tests in order to give statistical results and the enhancing of the algorithm in the context of a completely rigid formation with the aim to achieve the final target of the thesis (see Chp. 1) Moreover one of the future works could be the application of this technique to the AUVs described in Chp. 5 with the purpose to create reconfigurable underwater MANETs.

6. A HYBRID TECHNIQUE FOR CONTROLLING PLATOONS OF MOBILE ROBOTS WITH PSO AND CONSENSUS

7

Robotics in Education: Platform & Infrastructures

The works developed in the field of educational robotics are presented in this chapter. The "Unibot Mobile Robot Project" and the "Unibot Remote Laboratory" are platforms and infrastructures that permit the study and the real application of the coordination algorithms presented in the previous chapter.

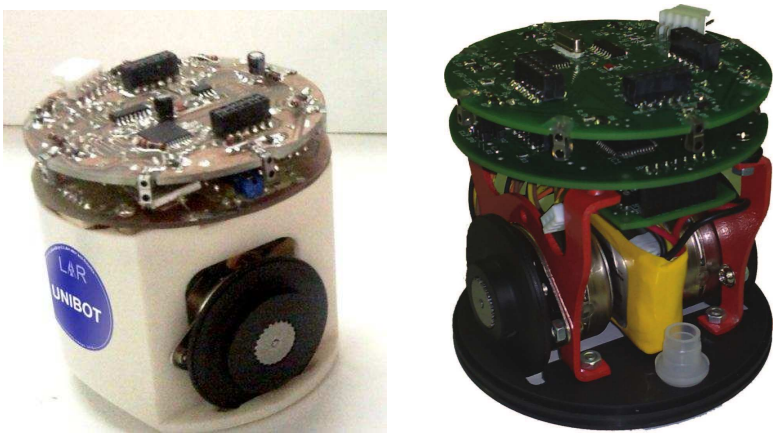
7.1 Unibot

Unibot is a prototype of a differential wheeled mobile robot. It was born with the aim to develop an experimental robot platform with a low economic impact for testing and validating the distributed control algorithm that involved groups of robot. The aim also includes the subsequently possibility to implement these algorithms on higher scale robotic platforms.

The basic idea of Unibot is to have a robot as much modular as possible, expandable and easy to use, following the idea previously developed at Ecole Polytechnique Federale de Lausanne (EPFL) with the mobile robot E-puck (see Fig. 7.5a). The basic version is equipped with a very small set of sensors but in any case useful enough to give good performances. In the table 7.1 a short summary of robot's features, deeply described in the Chp. 8. The design and realization of Unibot have been involved both Master and PhD students. Their collaboration has produced the prototype version of the robot in 2008 (Fig. 7.1a) and subsequently an optimized version in 2009 (Fig. 7.1b).

Name	Description
Platform	Custom, based on MicroChip PIC16F54 micro controllers
Internal Bus	Custom 10 bits bus: 1 byte of data, 2 bits of control
Motors	Two motor stepper 56 step 3 A. Torque unknown
Sensors	8 infra-red sensors - working range 10 cm
Communications	Bluetooth device
Chassis	Painted preformed metal chassis and plastic

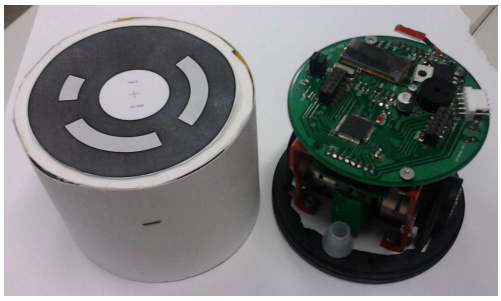
Table 7.1: Unibot V_1 short list of specifics.



(a) **(b)**

Figure 7.1: The prototype version (left) and the first full-functional version of Unibot V_1 (right) have been the inspiration for designing the later version.

The presence of this first version of the mobile robot has given to us, during the years 2009-2010, the opportunity to study, design and implement the information control infrastructure described in Chp. 8. Using this infrastructure it is possible to drive a group of Unibots inside an arena (see Fig.7.2), to test e.g. a trajectory following algorithm (depicted in Fig.7.2b). The framework is completely developed with Java technologies, exploiting the software agent programming paradigm. It is based on a completely modular version of the *Client-Server Architectural Pattern* that allows a strong decentralized information managing. In 2011 the framework has been expanded [41] to include a web-interface responsible to laboratory exercises and experiments. The interface allows the managing of the related hardware and software resources through



(a) The Unibot markers exploited by the system to recover the robot's positions inside the arena.

(b) A trajectory following test inside the arena.

Figure 7.2: Unibot tests arena.

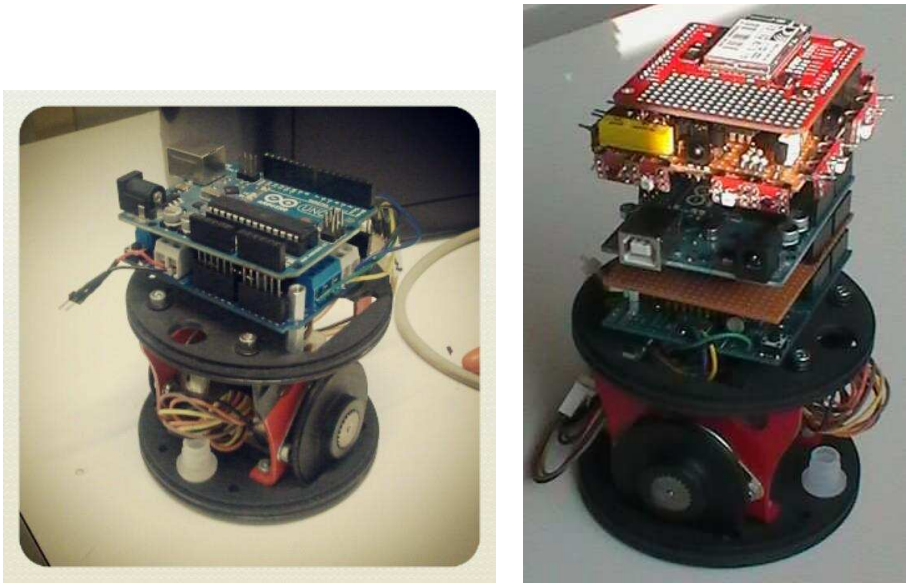
a simple internet browser.

This expansion has included the J.U.S.E. simulator (see Sec. 9.1) as part of the infrastructure. It has become the *safety* test field for Unibot control algorithms before the implementation on real robots. This way the experimental platform "UniBot Remote Laboratory", which is the topic of Chp. 8 was formed.

Through an experimental set-up consisting of a little arena, a webcam, a middle-range workstation, two Unibots V_1 and a video tracking program[1], it has been possible to test some distributed control algorithms during the laboratory activity carried out in 2011 (see Fig.7.2b).

The tests carried out have pointed out some hardware limitations of this Unibot version. In order to overcome these limitations we have studied a new version of Unibot, more functional, easier to use and to program, nearer to students and less expensive in term of time for design and prototyping. The choice for the realization of a new version of the robot has fallen on Arduino platform. The huge development encountered by this platform in recent years, the large support community, the easiness of use and the continuous effort to create new hardware and software are the points of strength of this solution. The platform provides the possibility to quickly develop robot prototypes of good quality and cheap, maintaining the modular and expandable philosophy that characterized Unibot project. At the beginning of 2012 the first A-Unibot prototype has been realized exploiting some parts of the previous version (see Figs 7.3a and 7.3b).

[1]The video tracking program is *SwisTrack* developed at EPFL.

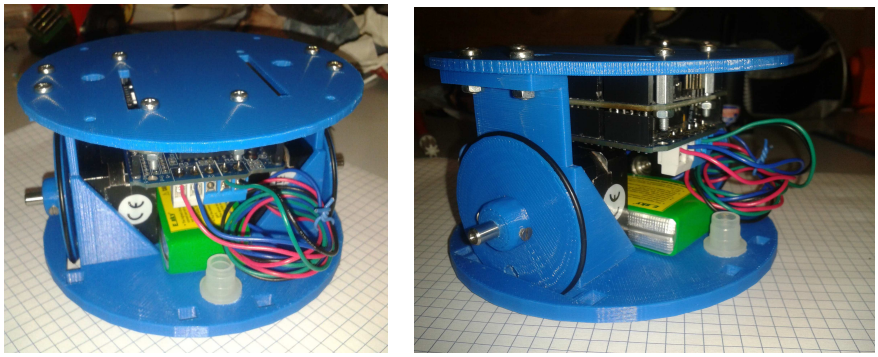


(a) (b)

Figure 7.3: The A-Unibot prototype chassis with the main board (left) and the complete prototype equipped with a custom DIY square sensor board and the WiFy device.

In figure-set 7.4 the chassis of the new version of Unibot is depicted. Its building started on july 2012. The chassis' mechanical structure was realized by using the 3D printer available at Laboratory of Automation and Robotics (L.A.R.) of the University of Bologna. Part of the equipment of which the robot is endowed (motors and motor-drivers) is visible in figure-set 7.4. A short list of A-Unibot components is presented in table 7.2.

In Fig. 7.5a a comparison between the different versions of Unibot and the well-known E-puck is presented. The bigger dimensions and the rigid plastic structure allow applications that would have been impossible to be tested with other kinds of robots. The powerful motors provide a high torque, useful feature for the final aim of this work. The next step, under development, is to equip the robot with a recharge board so that it can interface with a charging base and autonomously manage the status of its battery pack. On the recharge board infra-red ground sensors have been also inserted (Fig. 7.5b) useful to detect void areas (e.g. stairs) or different color areas.



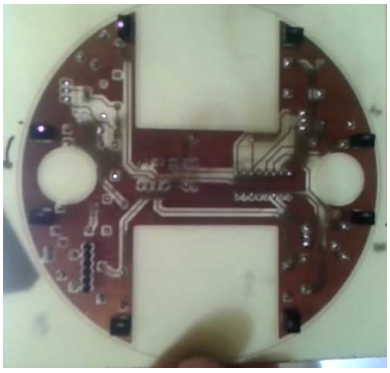
(a) (b)

Figure 7.4: Frontal and lateral view of the A-Unibot chassis.

Name	Description
Platform	Arduino, based on Atmel Atmega 328 16 bit micro controllers
Internal Bus	Arduino I^2C Bus
Motors	Two motor stepper 200 step 0.7 A. Torque 650 g/cm
Sensors	8 infra-red sensors - working range 15 cm
Communications	Wireless device
Chassis	3D printed plastic

Table 7.2: Unibot V_2 short list of specifics.

(a) A mobile robot comparison between Unibot V_1, V_2 and E-puck.



(b) The future ground sensors' board.

7.2 Unibot Remote Laboratory

This section briefly presents the Unibot Remote Laboratory (whose core infrastructure has been explained in the Chp. 8) that was born with the aim to provide an educational platform for mobile robotics in a remote mode also. Indeed UniBot Remote Laboratory (URL) is a prototype of tele-laboratory developed at University of Bologna mainly designed with the purpose to interface with Unibot V_1. The laboratory has been implemented to provide remote access via TCP connection, to assign different time-slots to students for their experiences, and to reduce the financial effort required by real set-ups. Moreover, the entire framework has been developed with high modularity both from hardware's and software's point of view and, even if the basic set-up has been conceived for mobile robotics, different kind of robots or automatic machines can be easily added and be available for experimental activities.

7.3 LEGO MindStorm

This section is not so closely related to the field that is normally considered "research", anyway it is deeply related to the concept of "Educational Robotics", focus of this chapter. During the PhD period we have spent almost ten months full-time as laboratory tutor for the course of "Foundations of Industrial Robotics" at University of Bologna. The main activity developed in this experience has been the teaching of techniques, methods and methodologies belonging to Computer Science Engineering context, in order to apply them to Mobile Robotics using LEGO Mindstorms NXT. In thirty hours of lesson as theoretic support for the laboratory activities, the students who have attended the course have learned the basic skills of Java programming, useful to write programs for the LEGO Kit, using an alternative open-source firmware based on Java technology, which name is LeJOS (presented in Sec. 7.3.2). A brief introduction of the LEGO Kit and its applications in many research fields, especially in robotics, has been presented in section 7.3.3. Moreover the task developed during the tutoring activity has described in section 7.3.4. Finally, conclusions are described in section 7.3.5.

7.3.1 LEGO Mindstorms Kit

The LEGO Mindstorms series of kits contain software and hardware to create small, customizable and programmable robots. They include a programmable brick computer that controls the system, a set of modular sensors and motors, and LEGO parts, coming from the *Technics* line in order to create the mechanical systems. The hardware and software on which the *Mindstorms Robotics Invention System* kit is based, take the basis from the programmable brick created at the *MIT Media Lab.* This brick was programmed using Brick Logo and an adaptation of the more famous programming language Logo. The software innovation was the possibility to programming the brick in a visual fashion. The first visual programming environment was called *LEGOsheets* since it was created by the *University of Colorado* in 1994 and based on *AgentSheets* an educational program for the Cyberlearning.
Virtually all kinds of existing integrated electromechanical systems (such as trucks or industrial robots) can be modelled with LEGO Mindstorms. The brick is programmed by uploading a program, written in one of the several programming languages available

Name	Description
CPU	8-bit Renesas H8/300 micro-controller
RAM	32K that stores both the firmware and user programs
Ports	3 motor ports and 3 sensor ports
Motors	3 servomotors each one equipped with an integrated rotation sensor for a precision feed-back controlling
Sensors	Light and Touch sensors
Communications	USB 1.0/serial port or special infra-red communication's interface. The second one also for team-mates communication
Display	1 row 10 character LCD display
Pieces	519 pieces of LEGO Technic
Standard Programming Interface	RCX Code or ROBOLAB (based on LabVIEW)
Power supply	Cable or batteries

Table 7.3: LEGO Mindstorms RCX specifics.

(see Sec. 7.3.2) Different versions of LEGO Kit have been released over the time, from the initial versions to the final ones, following the growth of electronics, the features of each versions have grown permitting the development of more sophisticated robots (and programs). In the following sections the basic features of the LEGO Mindstorms Kits have been presented. For more details it is possible to consult the related websites.

7.3.1.1 LEGO Mindstorms RCX

The first generation of LEGO Mindstorms was built around a brick known as RCX (Robotic Command eXplorers). Also nowadays, after many electronic innovations in the field of SOCs (single onboard computer), RCX specifics are not so bad, especially for an educational purpose (see table 7.3). An example of application with RCX brick is shown in Fig. 7.5d.

These LEGO version was used in the laboratory exercises carried out some years ago by using BrickOS as C/C++ programming interface.

Name	Description
CPU	32 bit Atmel AT91SAM7S256 at 48 MHz (ARM7) and a coprocessor 8 bit Atmel ATmega48 at 8 MHz (RISC)
RAM	256k flash and 64k RAM for the CPU, 4k flash e 512 byte RAM for the coprocessor
Ports	3 motor ports and 4 sensor ports
Motors	3 servomotors each one equipped with an integrated rotation sensor for a precision feed-back controlling. More powerful with respect to the previous one in terms of torque and speed
Sensors	Color sensor, 2 touch sensors, audio sensor and an ultrasonic sensor with approximatively 60 cm of useful range
Communications	USB 2.0 or Bluetooth. The second one also for activate a communication with at most 4 team-mates
Display	LCD with a 60x100 pixel of resolution
Pieces	619 pieces of LEGO Technic
Standard Programming Interface	NXT-G a graphical programming environment that enables the creation and downloading of programs to the NXT developed by National Instruments as branch of LabVIEW
Power supply	Cable or batteries, also rechargeable

Table 7.4: LEGO Mindstorms NXT specifics.

7.3.1.2 LEGO Mindstorms NXT

The second version of LEGO Mindstorms (called NXT from the word *NEXT*) was released by LEGO in July 2006, replacing the first LEGO Mindstorms kit generation. It is much more powerful in term of hardware and also the standard programming interface is changed following the philosophy of visual programming and using visual structured block connectible with a logic line. This Kit has been intensively tested in our laboratory experiment as described in 7.3.4. An example of application with NXT brick is shown in Fig. 7.5f.

7.3.1.3 LEGO Mindstorms NXT Kit EV_3

The strong worldwide demand of LEGO Mindstorms Kits has forced LEGO Company to build a renewed Mindstorms Kit called LEGO MINDSTORMS NXT EV_3, the most

Name	Description
CPU	ARM9 300 MHz with Linux-based operative system
RAM	16 MB of Flash and 64 MB of RAM and a Mini-SDHC card reader for 32 GB of expansion memory
Ports	4 motor ports and 4 sensor ports
Motors	3 servomotors (2 large 1 medium) with the same features of the NXT ones
Sensors	The same NXT sensors and moreover an improved 6-color detection sensor
Communications	USB 2.0, Bluetooth v2.1 and WiFy connection. Also a remote controller is available. Up to four intelligent brick can be connected together
Display	LCD with a 178x128 pixel of resolution
Pieces	594 pieces of LEGO Technic
Standard Programming Interface	The same of NXT
Power supply	Cable or batteries, also rechargeable

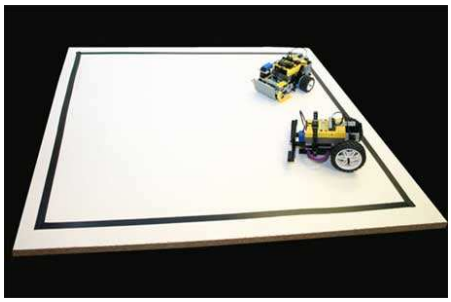
Table 7.5: LEGO Mindstorms NXT EV_3 specifics.

international robotics platform the company has ever developed. Presented in January 2013 at Consumer Electronics Show (CES), its release is planned for July 2013 and the specifics have been improved yet, in order to follow electronics evolution. In table 7.5 a brief list of specifics. An example of application with the new version of NXT is depicted in Fig.7.5h.

7.3.2 Programming Language

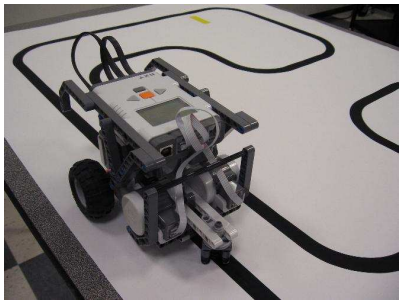
The standard interface for LEGO Mindstorms programming is provided by National Instruments and has been developed with the *Visual Programming* philosophy, using the same concept applied in the well known Matlab-Simulink and LabVIEW. The name of this graphical programming environment that comes in bundle with the NXT kit is NXT-G (see Fig. 7.6). The possibility to easy program the LEGO brick with this kind of interface is most user friendly and powerful but not useful for our purposes. Many other programming software can be used as an alternative to NXT-G, some based on C/C++ other on Java, Visual Basic, Logo and so on. In some cases it is necessary to replace the original firmware in order to command the brick. The aim of

(c) RCX Brick with some sensors and two motors.



(d) A Sumo ring where the robots are controlled by a RCX brick.

(e) The LEGO Mindstorms NXT brick.



(f) A line-follower application developed with a NXT differential wheeled robot.

(g) An inverse pendulum disguised as a biped robot built with NXT-EV_3.

(h) A LEGO Mindstorms NXT-EV_3 scorpion.

Figure 7.5: Different application of the LEGO Mindstorms from the first one RCX to the last one NXT EV_3 which release is programmed for July 2013.

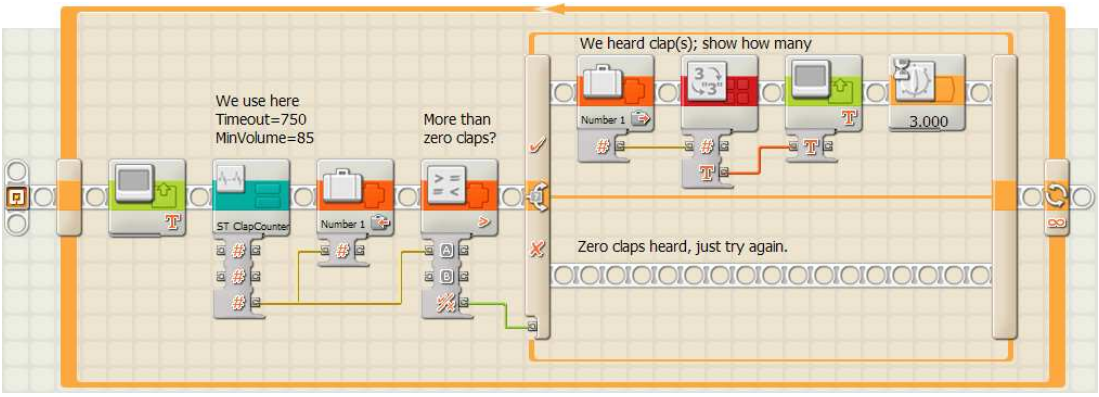


Figure 7.6: Screenshot from LEGO Mindstorms NXT-G programming interface.

the laboratory's experiences has been to learn the basic skills of programming a real automatic machine and the multi-thread concurrent programming. The most complete software available to achieve these targets was LeJOS.

7.3.2.1 LeJOS

LeJOS is a firmware replacement for LEGO Mindstorms programmable bricks. It currently supports the LEGO RCX brick and the NXT brick. It includes a Java virtual machine, which allows LEGO Mindstorms robots to be programmed in Java programming language. Some LeJOS important features are the following:

- object oriented language (Java);
- pre-emptive threads (determined context switching);
- arrays, including the multi-dimensional ones;
- recursion;
- synchronization;
- exceptions;
- types of variable including float, long, and String;
- most of the standard Java classes are available (as java.lang, java.io, java.util);
- a well-documented Robotics API;
- a wide community of support.

A remarkable example of LeJOS application was the leJOS-based robot *Jitter* that flew around on the International Space Station in December 2001.

7.3.3 Applications

LEGO Mindstorms is deeply used worldwide not only as a children's toy but also as a low-cost but powerful tool for mobile robotics. Many paper are based on LEGO Kit as tests platform, papers that involve the construction of new sensors or papers that develop a new software architecture in order to provide a better firmware. LEGO Company provides also accelerometer or gyroscope sensors and many other specific sensors not included in the basic kit, useful for example to build an inverse pendulum like the one depicted in Fig. 7.5g or the one described in [70] where a PD controller exploits accelerometer's data to maintain the robot vertically stable. On the same robot configuration many other algorithms can be tested, such as the Extended Kalman Filter described in [71]. LEGO Kit has also been used in the experimenting of remote controlling as the one presented in [72] or the one executed to test the *Unibot Infrastructure* presented in Chp. 8, where both Unibot Mobile robot and a LEGO Mindstorn NXT robot have been involved, which have been properly configured to receive remote commands from our infrastructure in order to perform a leader-follower test. Moreover LEGO has been exploited in the experiment that involved Multi-Agent Systems like the one described in [73]. In the next section our laboratory activities with some brief descriptions of the tasks performed are presented.

7.3.4 Laboratory Activities

This section briefly describes the applications developed with LEGO Mindstorms NXT during the laboratory activities:

- **car parking**: the main task was to build a self parking robot-car. The scenario used as test for this task was a model of a real street, built using pieces of cartoon, old books and boxes found in the laboratory. The street was built with different parking areas, each one with different dimensions, distributed on both sides of the street. Moreover the street was built with only one entrance (as a classic blind street). This way the cars had the possibility to recognize the end of the street and execute the manoeuvre to come back in order to check the other side of the street for finding a better parking area. The final target was achieved only when the car found the correct parking or it went out from the entrance. Some figures of this task are depicted in 7.7a and 7.7b;
- **sumo robot**: this task, inspired by the ancient art of Sumo combat, took place on a self-made wooden squared arena on which a black ring was painted. Different matches were played with different opponent configurations e.g. one-to-one and many-to-many matches with always the same target: to push the other opponent/s out of the ring, remaining inside it. Different strategies of searching, following and pushing have been studied in order to win the match, exploiting all the LEGO resources. Two images of this task are presented in Figs 7.7c and 7.7d;
- **mobile gripper**: the main task was to build a Mobile Gripper that involved two different LEGO Mindstorms physically connected to each other. This task consisted in recognizing,

collecting or discharging different coloured objects in relation to a specified desired color. In order to perform the task, the student's group needed to build a gripper with 4 degrees of freedom, mounted on a mobile robot with a basket. The mobile robot and the gripper must communicate with each other to recover the object and to identify if the object's color was the desired one. In affirmative case the robot must collect the object inside the basket, positioned on the robot's back, otherwise it had to discharge it and to continue the search for another object. The objects were placed on a black line, painted on the floor as a track, which was followed by the robot. The task was accomplished when all objects were correctly processed and the right coloured objects deployed on the path were collected in the basket. Two moments of the performance are shown in Figs 7.7e and 7.7f;

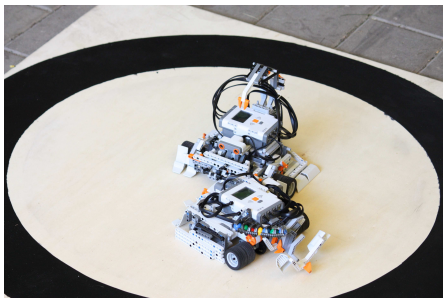
- **box transportation**: this task was based, such as the first ones, on a robots' competition. The challenge of this task was to transport a box, properly configured in order to be managed by the LEGO robots, from a starting area to a final one of an unknown environment full of obstacles. In this task the robots identified the starting area and the final one exploiting some proper lines on the floor. With the sonar sensors, the robots were able not only to detect the position of the box but also to avoid the obstacles. Some pictures that represent this task are shown in Figs 7.7g and 7.7h.

7.3.5 Conclusions

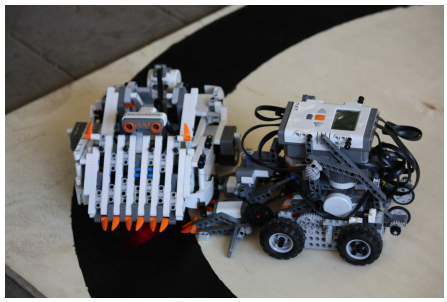
The teaching method based on coding programs for a real machine and giving to students the target of "winning a competition" has been the success of this experience and it has been very appreciated from the students. The tasks developed have been hard and challenging but the method used to create teams of development in order to better face the given tasks has proven successful. Moreover the learned knowledges have been useful also for computer science courses.

(a)

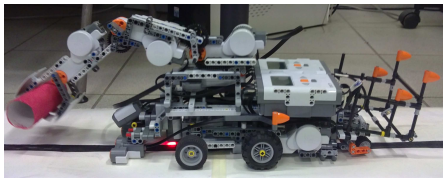
(b)



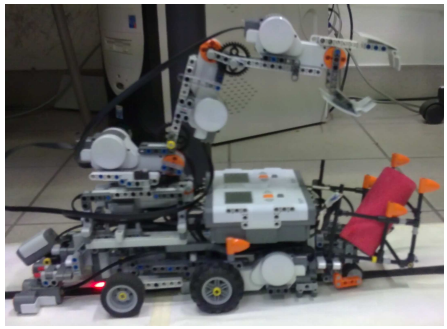
(c)



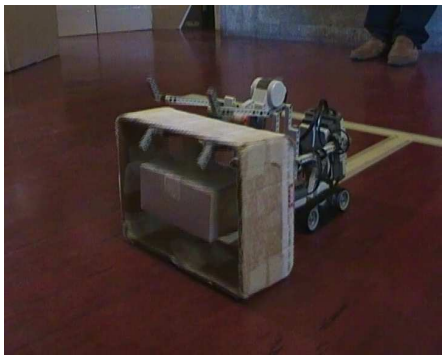
(d)



(e)



(f)



(g)

(h)

Figure 7.7: Some pictures show the different activities developed during the laboratory of robotics.

8

UniBot Remote Laboratory

This chapter presents the UniBot Remote Laboratory, a scalable web-based set-Up for education and experimental activities in robotics. It is a prototype of tele-laboratory developed at University of Bologna. The laboratory has been implemented to provide remote access via TCP connection, to assign different time-slots to students for their experiences, and to reduce the financial effort required by real set-ups. Moreover, the entire framework has been developed with high modularity both from hardware's and software's point of view and, even if the basic set-up has been conceived for mobile robotics, different kind of robots or automatic machines can be easily added and be available for experimental activities.

8.1 Introduction

There are no doubts about the importance of laboratory's experiences, for education, in several engineering fields [74]. This is a consequence of the fact that, while students learn new concepts during classes, they need to test their abilities on real systems, in order to fix and prove their knowledge. Although the ideal situation should be having an experimental set-up for each student, and possibly for each class of control problems, this is not possible, due to several practical reasons:

- each set-up is typically sold in bundle with its own teaching software, thus it may be difficult to modify it in order to implement different set-ups than the ones suggested by the dealer;
- each student, before using every experimental set-up, should understand the entire architecture of the system, and sometimes this might require more efforts than solving the problem for which the set-up has been made available;
- the financial effort for the maintenance of a large number of experimental set-ups is not indifferent.

8. UNIBOT REMOTE LABORATORY

Different experimental set-ups are provided by different companies, thus generally it may also result quite complex to interface or connect them together. It follows that it could be difficult to create a complex student lab without a great effort in order to re-engineering all these systems.

In recent years, starting from this simple considerations, many papers, focusing on low cost experimental set-ups, have been presented in literature, spacing in a wide range of different applications, from automation control [75] to robotics [76]. In fact, due to the increased computational power of personal computers, it has become possible to create small low-cost set-ups, built with off-the-shelf components, whose teaching potentialities can be compared with more expensive and performing systems.

In parallel, the recent diffusion of high bandwidth internet connections and the rapid development of web-based technologies, have led to a new concept of *laboratory experience*. In fact, current generation of laboratory facilities has been implemented in order to allow the remote access to a real experimental set-up [77, 78]. This means that students can experience the effects of their control programs on real machines, without the need to be physically present, e.g. by using a web-cam to observe the behaviour of the real system and collecting data to be analysed later on their own computers. Most of these labs use a web-based interface where the students, after authentication, can choose the experiment to be performed, and reserve a *time-slot*, for their own purposes. Some of them provide the possibility to create a Matlab/Simulink control scheme to be uploaded and executed. Nevertheless, all these *virtual labs* are strictly limited by many factors, in principle the fact that no local supervisor is usually implemented in the set-up, and the system must be safe and open-loop stable to prevent faults, due to an inefficient uploaded controller. Focusing the discussion on robotics labs, to our knowledge, some of the first remote robotic laboratories have been implemented by [79] and by [80], where the user can not strictly control the manipulator, but can only define a sequence of movements that are then computed by the local robot controller. Even if these results could be considered milestones for remote robotics labs, in order to provide students with a more involving experience, more efforts should be devoted to the implementation of flexible and user-friendly environment. As we will describe in the following sections, we have created a remote robotics lab that allows students to define a complete controller for a mobile robot, such that it can interact both with the environment and with other robots present in the arena (eventually driven by different users). Before working on real robots, users can test their algorithms in a virtual environment, where experimental set-ups are reproduced.

This chapter is organized as follows: in Section 8.2, the *UniBot* (UNIversity of Bologna mobile roBOT) differential-wheeled robot, the first robot used to test our framework, is presented, focusing on its mechanical and electronic design. In Section 8.3, an overview of the *UniBot Remote Lab* architecture is given, focusing on the web-based design of the system. An applicative example of our virtual lab is presented in Sec. 8.4, while in Section 8.5 some considerations about the presented work and future developments are reported.

Robot Name	Producer	Price (USD)
E-Puck	EPFL	≈ 850
Roomba	iRobot	≥ 450
Khepera III	K-Team Corporation	≈ 3000
K-Junior	K-Team Corporation	≈ 600
AmigoBot	ActivMedia Robotics	≈ 3500
Robotino	Festo Didactic	≈ 4000
UniBot	University of Bologna	≈ 250

Table 8.1: Comparison of different robots suitable for research and teaching purposes.

8.2 The UniBot Differential-Wheeled Mobile Robot

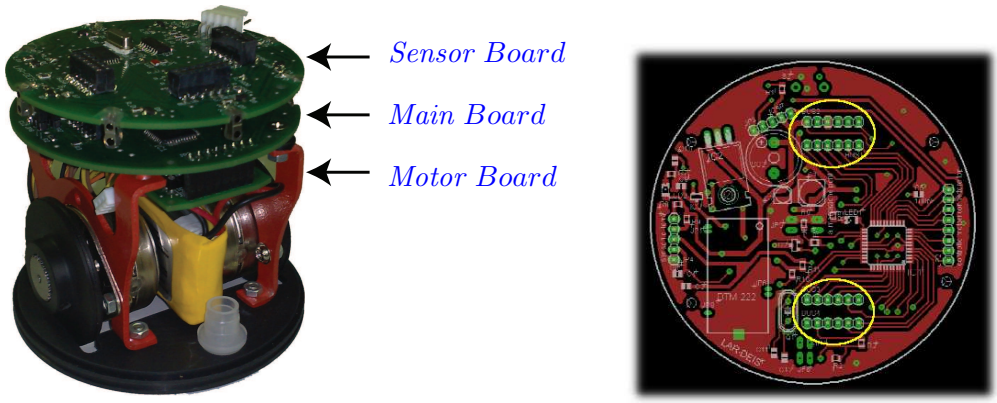
Since part of the research interests in our lab focuses on mobile robotics, and in particular on the control of platoons of differential-wheel robots, it was taken the decision to design a small mobile robot that could be used both for teaching and research activities. Even if many small robots can be found [81, 82], we start analysing the solutions offered by the market, typically too expensive to be used by many students. Roughly speaking, to define the basic specifications of our robots, we have previously defined the basic properties requested to a research and teaching instrument, ensuring at the same time the possibility of upgrading each vehicle with additional boards/functionalities to improve its performances. In particular, we have analysed many commercial robots suitable for our purposes considering at the same time both the price and the hardware/software configuration (see Table 8.1). Form the hardware's point of view, the K-Junior by K-Team is the commercial robot more similar to UniBot.

One of the main issues related to the analysed models is that, typically, they are sold with many hardware features that are rarely used or are redundant (such as the small camera embedded in the E-Puck robots or the sonar and IR sensors in the Khepera III robots), and thus the price is not always justified for some predefined tasks. Moreover, many of the commercial robots, whose price is low enough to be used as part of the teaching activity, are usually sold as monolithic machines whose performances cannot be upgraded by adding new hardware.

Starting from these considerations, we designed a small differential wheeled robot as part of our research and teaching framework, with particular attention to the hardware modularity, that is considered a key point in order to create a versatile robot that can be customized for a set of experiments as large as possible.

8.2.1 UniBot Hardware

A prototype of our robot, named *UniBot*, can be seen in Fig. 8.1. *UniBot* has been designed in order to be as flexible (and cheap) as possible: the chassis is composed by only two modelled aluminium shapes, where the motors are bolted, and a plastic platform (like a *safety ring*) is



(a) (b)

Figure 8.1: Prototype of the UniBot differential-wheeled mobile robot. In evidence the vertical bus on the main board's schema.

used to prevent collisions between the robots and other robots or obstacles in the environment. Moreover, the reduced dimensions of the robot ($8 \times 8 \times 8$ cm) make it suitable for experimental set-up, even in small labs. Following the same philosophy, the on-board electronics has been designed to be easily assembled, thus ensuring that broken boards can be easily replaced. It follows that the easy-to-mount design allows the users to add or remove expansion boards in order to improve or reduce the robot's performances, depending on the experiments performed. From these considerations it follows that UniBot differential-wheeled robot is structured in order to preserve high modularity. In particular, in its basic version, it is composed by three different boards able to ensure basic navigation skills. Namely, these three boards are the *Motor* board, the *Main* board and the *Proximity sensors* board.

8.2.1.1 Motor Board

The board is endowed with two H-bridges in order to control the motors. For the actuation, we have chosen two cheap step motors ($\approx$ 0.07 Euro per motor) with 56 steps. Robot's wheels are endowed with plastic o-rings in order to reduce the slippage. Moreover, wheels of different diameters are provided with the robot, and the motors can be fixed at different heights in order to change the odometry precision. With the default wheels, the odometry precision is around 3 millimetres. Due to the chosen motors, it has been possible to codify a total of 11 speed levels, 5 levels forward, 5 backward and the null velocity.

8.2.1.2 Main Board

The board, equipped with a Microchip 20Mhz PIC16F877A, is the core of the UniBot robot. In fact, the role of the main board is to execute the code programmed, exploiting information provided by expansion boards, mounted on the UniBot. The only sensors directly connected

to the main board are two chromatic sensors, placed under the base platform, in the front part of the robot. These sensors allow students to work e.g. on the solution of the follow-the-line task, a typical problem involving low-level control actions. Moreover, a bluetooth module has been mounted on the main board to let different robots exchange data among them or with an external computer.

8.2.1.3 Proximity Sensors Board

As an example of expansion board that can be mounted on the robot, in order to improve its performances, a board equipped with 8 IR sensors is provided with the basic robot version. The sensors are equally distributed around the board perimeter to detect obstacles or other robots in the environment. This board is equipped with the same microprocessor mounted on the main board so that it has enough on-board computational power to calculate a control action to avoid the detected obstacles.

Each expansion board, mounted on UniBot robot, is able not only to gather information from the environment, but also to calculate the best control action related to the collected data. For example, let us consider the case where a robot equipped with an IR sensor board and a sonar board is moving toward a predefined target. In this case, an obstacle higher than the robot can be detected by the sonar board before being detected by the IR sensors: it follows that, even if the IR board does not detect any object, the sonar board provides to the main board a control action in order to avoid the obstacle. On the other side, if the obstacle is not high enough, it cannot be detected by the sonars but only by the IR sensors, and thus the IR sensor board will calculate the obstacle avoidance control action to be transmitted to the main board. For example, a Braitenberg algorithm [83] can be programmed on these two sensor boards. Receiving these information, the main board has to merge all the control actions in order to reach the target while avoiding the obstacle.

From the user's point of view, the peculiarity of the decentralized paradigm chosen to design UniBot is that it allows many persons to work on different boards with the idea that each of them can be used to define a particular behaviour of the robot depending on the sensors mounted on each board. Once different behaviours have been implemented, they can be merged by the main board that operates like a supervisor. This control paradigm, that is well known in literature as *behavioural control* [8], allows to implement on the main board the *competitive* version and the *cooperative* version of it, implementing controls such as the *motor scheme* [9] or the *null space based* control scheme [84, 85]. Another advantage coming from the decentralized paradigm applied to UniBot hardware is that the computational power of each board can be exploited, thus creating a sort of parallel computing structure on each robot.

8.2.2 UniBot firmware

Even if each board mounted on UniBot can be programmed by users with an of-the shelf PIC programmer, the basic version of the robot comes with a preloaded firmware that, basically, allows to control the robots by simply sending command bytes via Bluetooth. Thus, the preloaded

Received Byte	Instruction	Description
000-00000	HALT	HALT command for main board
001-00000	HALT	HALT command for IR sensors board
xxx-00001	LST_ON	start listen procedure on board xxx
xxx-00010	LST_OFF	stop listen procedure on board xxx
⋮	⋮	⋮
000-0yyyy	SRS	Set Right motor Speed to yyyy
000-1yyyy	SLS	Set Left motor Speed to yyyy
001-00zzz	IRS	Read IR sensor zzzz on IR sensor board

Table 8.2: Example of instructions coded in the on board firmware useful to drive the UniBot robot using a remote controller.

firmware on each board is able to parse the received bytes and to transform them in instructions executable by the board. As the only board equipped with a Bluetooth module is the main board, each byte must code the address of the location where the instruction must be executed. More in particular, each byte received by a UniBot is structured as in figure 8.2. As it can be clearly seen, the first three bits are used to index the board which the remaining 5 bits are referred to. This depends on hardware project that provides only 3 address bits. It follows that with the current implementation a maximum of 7 expansion boards can be mounted on top of the main board and, consequently, a maximum of $2^5 = 32$ instructions can be parsed by each board. To ensure the possibility of creating complex expansion boards, two instructions on each board are reserved to the LISTEN_ON and the LISTEN_OFF commands. When a board receives the LISTEN_ON command, it starts listening and buffering all the 5 bit instructions received till the LISTEN_OFF command stops it. Then, all the 5-bits groups received are concatenated to create instructions as complex as desired. Examples of instructions coded in UniBot firmware are reported in Table 8.2.

The advantage of using an ad-hoc firmware coding all the possible instruction that can be executed by a UniBot robot is that in this way it is possible to close the control loop of the robot not only exploiting its on board hardware, but also using an external computer connected to each robot via Bluetooth. It is worth to notice that in this way, as the calculus of the control

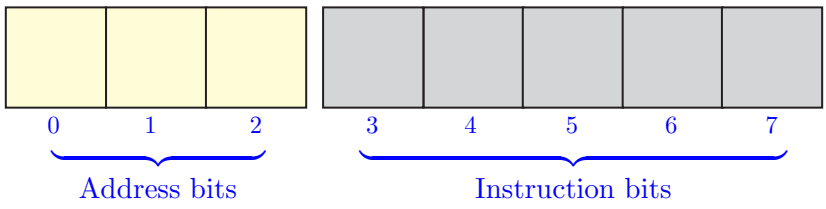


Figure 8.2: Byte command structure.

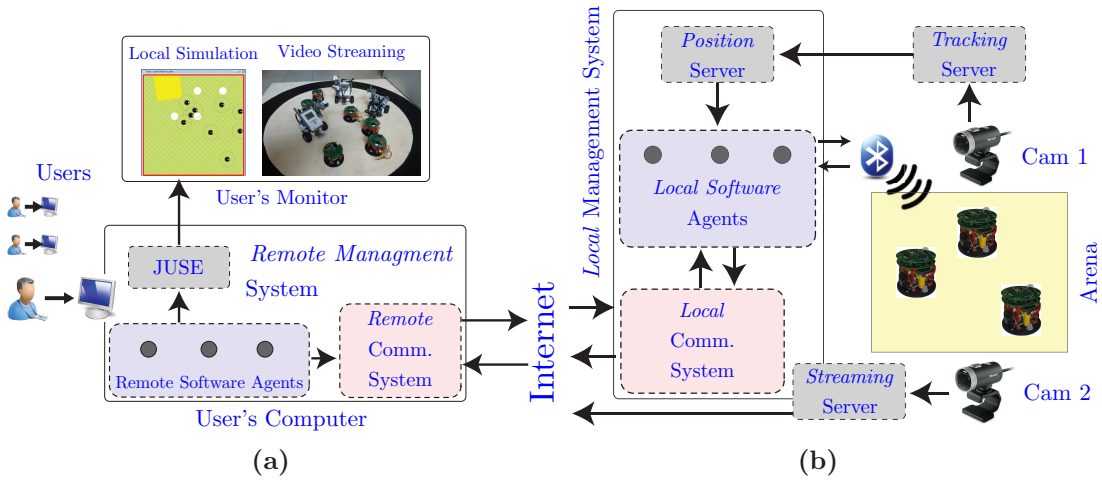


Figure 8.3: Scheme of the UniBot Remote Laboratory: Remote Management 8.3a and Local Management System 8.3b.

actions of each robot is demanded to more powerful machine, more complex behaviours can be performed, and thus more interesting and complex experiments can be tested.

8.3 Remote Laboratory architecture's overview

Beside the creation of a low-cost teaching-oriented robot, our effort focused on the creation of a software infrastructure (see Fig. 8.3) in order to allow the students to perform experiments without being physically present in the laboratory. In fact, this software has its strengths in the fully web-based architecture that can be considered as divided into three main parts: the *Remote Management System*, the *Local Management System* and the *Java UniBot Simulation Environment* (J.U.S.E.).

8.3.1 Local Management System

The Local Management System is the software used to let all the web-based laboratory software structures to communicates each other. As depicted in Fig. 8.3, the Local Management System is divided into three main interconnected parts: the *Tracking Server*, the *Software Agents Environment* and the *Local Communication Server*.

8.3.1.1 Tracking Server

The tracking system is mainly based on the SwisTrack software [81]. Having we attached special markers on the top of each point to be tracked (usually mobile robots, but more in general the markers can be used to acquire the position of other objects in the scene, like obstacles), a high resolution USB camera mounted on the top of the arena acquires the video of

the experimental environment that can be used both for streaming video and to get the position and the orientation of each marked point. As the data recovered by Swistrack are more complex and rich of information with respect to what our needs, they have to be preprocessed before they can be used to close a control loop. In particular, as described in Section 8.3.3, a string containing, for each robot, an ID number, the position and the orientation in the arena is extrapolated so that it can be sent in streaming to a remotely connected user or, alternatively, it can be saved to be analysed later.

8.3.1.2 Local Software Agents Environment

For each robot in the arena, a software agent is automatically created. The main task of each of agent is to manage the control code loaded by the user, to parse the instructions and to transform them in the right sequence of bytes that must be sent to the robots. After each instruction is coded, the corresponding byte (or bytes in case of complex instructions) is transmitted via Bluetooth to the robot assigned to the current software agents. The main advantage of using a Software Agent layer as an intermediary between the user and the robot is that not only UniBot robots but also any other robot equipped with a Bluetooth connection can be introduced in the system, as long as it has a protocol similar to the one described in Section 8.2.2.

8.3.1.3 Local Communication Server

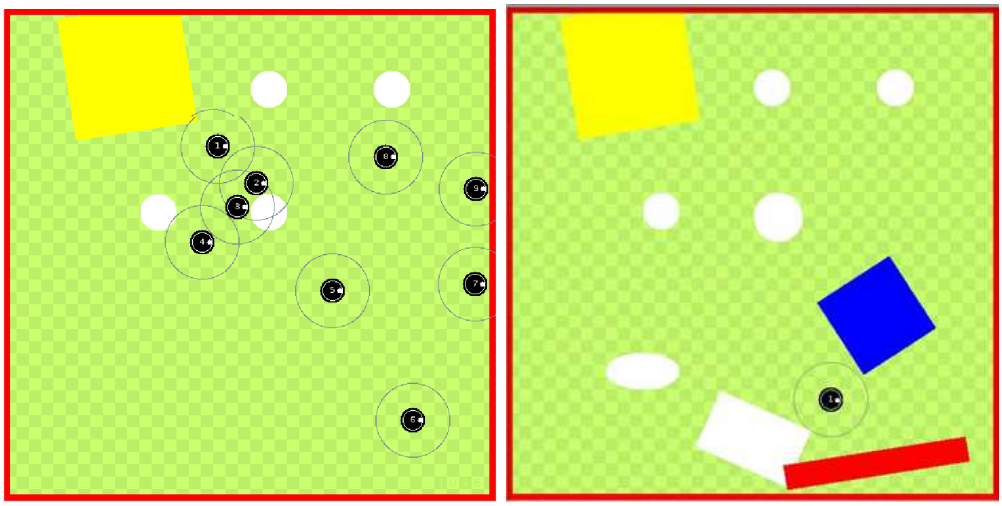
The Local Communication Server is a network interface created to control the robots in the arena providing the connection between agents and remote user. In that case, each *remote* software agent running on the computer of the remote user (see Sec. 8.3.2) sends strings to the corresponding local software agent containing the unique index of the commanded robot and the command to be executed. Then, each local software agent parses the received instructions and send it via Bluetooth to the right robot in the arena.

8.3.2 Remote Management System

The Remote Management System, that runs on the computer of the remotely connected user, is the counterpart of the Local Management System with the exception of the tracking server, that of course is not installed on any other computer than the one directly connected with the USB camera over the arena. The Remote Management System consists of two intercommunicating subsystems: the *Software Agent Environment* and the *Remote Communication Server*.

8.3.2.1 Remote Software Agent Environment

For each robot in the arena controlled by the user, a software agent is created on the remote computer. The task of each *remote* agent is to handle the peer-to-peer connection with its counterpart actually running on the computer connected via Bluetooth with the real robots. Let us note that software agents have a key role as all the messages exchanged between remotely connected user and the local system (i.e. the real robots in the arena) are filtered and parsed



(a) (b)

Figure 8.4: Snapshots of the Java UniBot Simulation Environment (J.U.S.E.).

by these agents. Roughly speaking, it is possible to imagine a two-way communication channel that connects each user to the real robot group assigned to him by the system (depending on the experiment).

8.3.2.2 Remote Communication Server

The Remote Communication Server mediates the data exchange between the Remote Management System (i.e. user's computer) and the Local Management System (i.e. the computer connected via Bluetooth with the robots). Its task is to handle all the messages exchanged between each couple of *remote* and *local* software agents, so that from the user's point of view it is not important if the controller is applied to a remotely controlled robot or to a robot simulated in J.U.S.E. It follows that both the software agents (and thus the commands mapped in the ad-hoc firmware) and the Remote Communication Server can be modified or updated independently.

8.3.3 The Java UniBot Simulation Environment (J.U.S.E.)

As part of the Remote Management System, a Java technology based simulator has been developed (see Fig. 8.4). J.U.S.E. has been thought as a stand-alone simulator where users can test algorithms on their computers before testing them on the real *remote* experimental set-up. To this purpose, each module created for the UniBot mobile robot is simulated, such as the IR sensors module. Moreover, by defining loading textures, it is possible to simulate different properties for the arena.

Another point that makes J.U.S.E. an important feature of the Remote Laboratory is

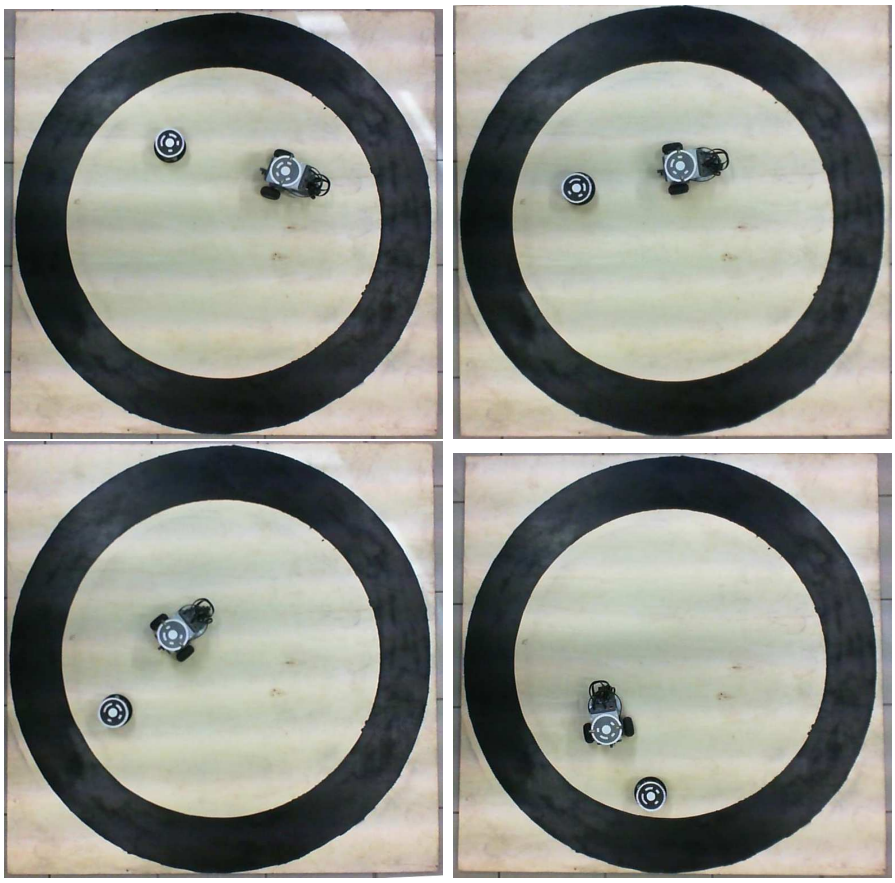


Figure 8.5: Snapshots of real experiments performed by students. The two robots, a UniBot and a Lego Mindstorm NXT, are engaged in a predator-prey algorithm. The black wide line around the arena, detected by the UniBot robot using its chromatic sensors,is used to delimit the escape area.

that it is possible to use it in order to overcome all the problems related to low bandwidth communications. In fact, in case a connected user has a low bandwidth communication and thus cannot exploit the streaming server, it is possible to configure J.U.S.E. in two different ways such that:

- the behaviour of the markers in the arena (i.e. the robots) can be shown by plotting the streamed position data into the simulator;

- a file collecting all the data recorded during an experiment can be loaded by the simulator in order to show the behaviour of the marked robots off-line.

8.4 An Application Example

In this section, an application of our teaching/research framework is reported. In particular, the task assigned to two groups of students was a problem of coordination between two different robots: a *UniBot Robot* and a mobile robot created using Lego Mindstrom NXT 2.0. The task assigned to the students was to coordinate the group of robots in a classic predator-prey task: the UniBot robot described in Section 8.2 was used as the prey and had to move inside a 50 cm radius arena while avoiding the other robot, while the Lego Mindstorm robot was the predator and had to reach the UniBot. In Fig. 8.5 snapshots of the experiments are reported: it can be clearly seen that a marker was placed on each robot as described in Sec. 8.3.1, in order to get the robots' positions. In fact, to force students to exploit all the features of our framework, the Lego robot (*follower*) was sensor-less, and thus it had to rely on the broadcasted data by the tracking server to know UniBot (*leader*) position. On the other side, the basic version of the UniBot robot used as leader was controlled by a remote computer by exploiting the firmware introduced in Section 8.2.2.

8.5 Conclusions and Future Work

In this work a framework for teaching and research purposes has been presented. This new framework has been thought as composed by two different parts. The first one is the UniBot differential-wheeled robot, a flexible (but cheap) mobile robot expressly designed for this activity. The main advantages of UniBot w.r.t. commercial robots is its cost with respect to other devices with comparable features and, at the same time, it allows users to add expansion boards in order to power up the robots depending on the experiment. The second part of the UniBot Remote Laboratory is the software developed in order to let registered users to perform experiments remotely.

Even if we have developed a new Unibot robot based on the Arduino's architecture (see Chp. 7) it is possible think to improve Unibot V_1 described in this chapter. This means that the future work regarding the UniBot Remote Laboratory will focus both on hardware upgrade for the UniBot robot and on new student oriented software features. From an hardware point of view, many improvements could be developed in order to provide the UniBot robot with more features. In particular, the attention could be focused on the project of a new set of boards, such as:

- a *sonar board* to improve the sensing ability of the robots;
- an *IR communication board* inspired by the range and bearing board [86] created for the Khepera III robot that will allow local communication between robots (this feature has been developed for A-Unibot);
- a *computational board* equipped with a 200 MHz ARM processor in order to implement on board more complex behaviours such that each robot can become a independent unit from external devices;

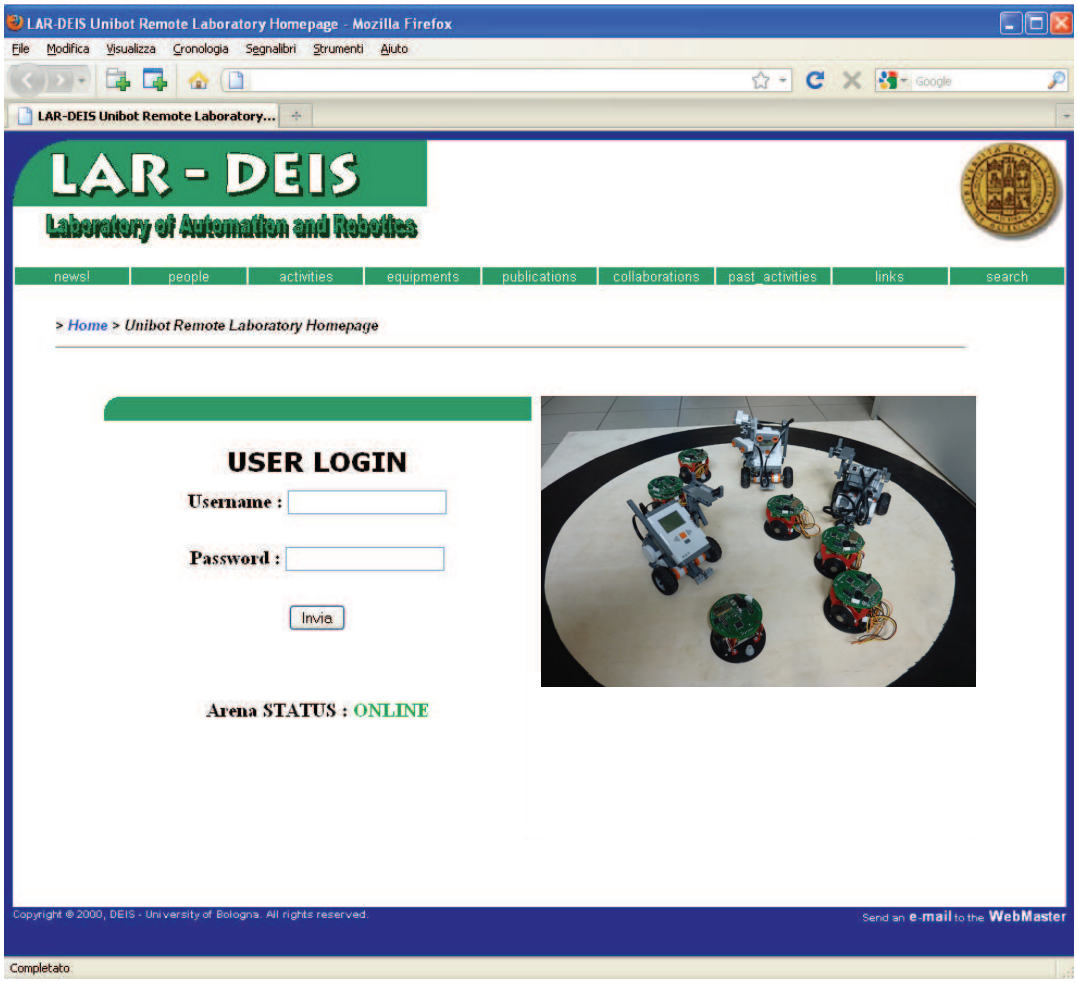


Figure 8.6: Preview of the under construction website to allow students to register and perform experiments via web.

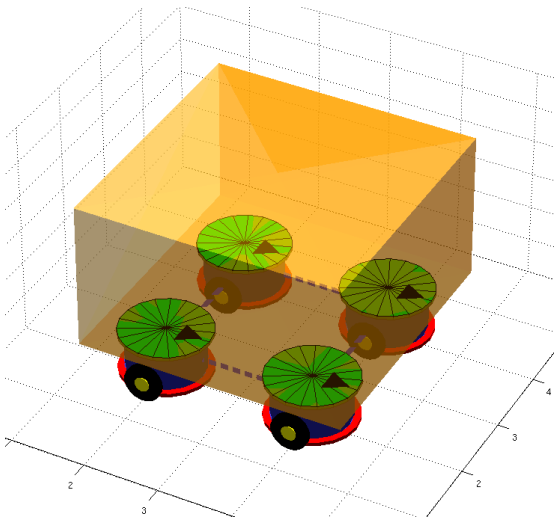
- an *arm equipped board* to be mounted on top of the robot so that experiments where an active interaction with the environment is required can be performed.

Moreover, with the prospect of mounting many expansion boards on a single robot, it is possible to build a special board that allows to include an extra battery in the system.

From a software point of view, we are currently developing a website where registered users (e.g. students) can load and perform experiments whose performances are automatically evaluated by the software. The skill of each user can be tested with with increasingly difficult experiments and the results are logged in a database. On the teacher's (researcher's) side, we are creating a Graphical User Interface (GUI) to allow the definition of experimental set-ups that can be shared on the web. A snapshot of the website is shown in Fig. 8.6.

9

Simulators



In this chapter two different types of simulators are presented, both developed with the main aim to simulate Unibot (see Sec. 7.1) and to provide a support for algorithm's testing and validation. The first one, J.U.S.E., has been inherited from a previous project [87] and later modified and extended as described in Sec. 9.1 with the purpose to embed it in the web interface of the web-lab project (described in Chp. 8). The acronym means Java Unibot Simulation Environment because the technology used as support is Java. The second simulator presented is M.U.S.E. which name means Matlab Unibot Simulation Environment. It has been built from scratch using Matlab as technology support, trying to overcome the lacks demonstrated by J.U.S.E. in terms of control algorithm development. Despite of Matlab was not born to create graphical objects and to operate with a code represented with Class and Objects models, in the development of M.U.S.E. this kind of approach has been used with very good results. The

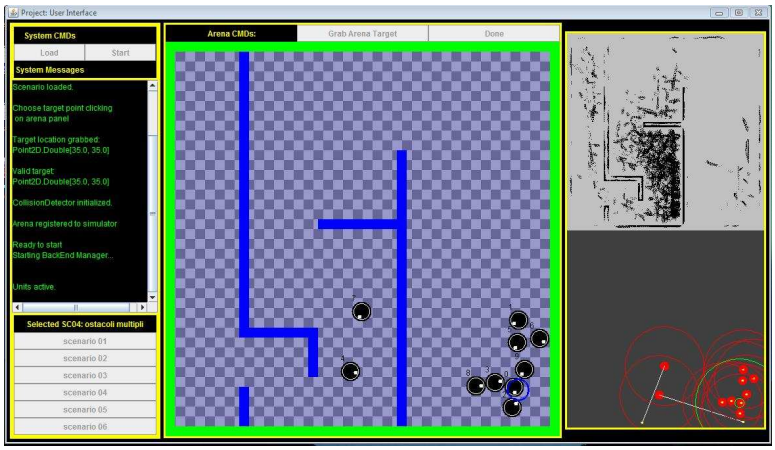


Figure 9.1: A screenshot of JUSE during a simulation in one of the maze-like scenario.

possibility to embed Java code inside Matlab code and interfacing M.U.S.E. with Simulink, provides a very powerful testing environment. Moreover M.U.S.E. can manage both 2D and 3D simulations providing good support to the three-dimensional algorithm on which is based part of this thesis (i.e. the one presented in Chp. 5). In Sec. 9.2 M.U.S.E. features are described in more detail.

9.1 J.U.S.E.

JUSE is a differential wheeled robot simulator written entirely with Java Software technologies with the aim to provide a simulation environment to university's custom-robot Unibot (see Sec. 7.1). Indeed it's name is the acronym of Java Unibot Simulation Environment.

The simulator was born in 2008 as an integrated environment with the initial aim to give only a visual support to our MSc-thesis project [87]. In Fig. 9.1 a simulation where it runs the PSO algorithm studied in the master thesis is depicted. The figure shows the robot swarm that moving in a maze-like scenario.

Afterwards, the need to have a simple simulation environment, modular, reusable and completely written in Java, has transformed J.U.S.E. in a stand-alone program, available both for research and educational activities.

The software architecture of JUSE follows the architectural pattern *Model-View-Controller* (see Fig. 9.2). This way it is possible to embed JUSE in different frameworks as we have done in Sec. 7.2 where the simulator was the main support for the *Remote Laboratory Project*. More details about this project are available in Chp. 8.

The robot model is based on a differential wheeled robot with eighteen proximity sensors (depicted in Fig. 9.3 as a little circle around each robot), a bidirectional communication device and a GPS module that provides each robot its exact location inside the simulated environment (aka arena).

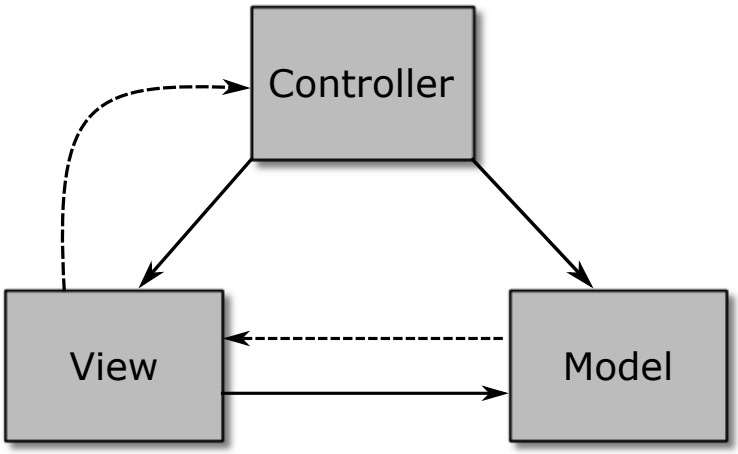


Figure 9.2: The Model-View-Controller Pattern Diagram.

On each simulated robot a software agent runs computing the intelligent algorithm and managing the virtual devices in order to detect environmental information and to share data between the robot's neighbours. As better explained in the related Sec. 7.2, JUSE is not only a viewer but a real field of experimentation where i.e. collisions between robot and obstacle or between robot and robot are computed.

9.1.1 Simulation Environment Components

The simulator is made by a large number of software components. In this section only a very short but significant description of the structure is done. For more detail see [87]. Starting the description from the bottom level to the top level of the simulator, the first component is the **Unit**. The Unit component has been designed as a real mobile robot with different operative levels. Two macro logic levels can be identified as

- *Robot*
  The level provides the graphical object to be project on the arena's surface. Moreover the level implements robot's motion, by computing the kinematic equations that define its position during the simulation. Moreover Robot level embeds the Communication Device and the Sensors. While the communication device is only a couple of message queue (in and out) where the messages from or for other units are stored, sensors are structured like a cloud of sensible points around each robot. When almost one of these points is activated, the robot register a collision. Finally two different typologies of inputs are accepted by robot procedures: linear and angular speeds or direct motor speeds. In both cases the robot converts the data into Cartesian speeds in order to calculate the correct robot's movement;

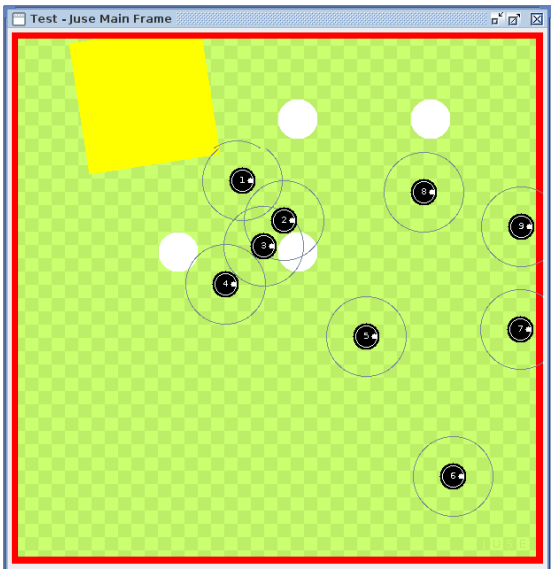


Figure 9.3: A top view of the simulated arena where a swarm of robots is performs a task. The depicted white circles are environmental obstacles, the yellow square a coloured area only. The obstacle detection performed by J.U.S.E. is based on the colours detected below the sensible area. This way it is quite simple to define if a coloured area is an obstacle or not.

- *Software Agent*
  The software agent is the mind of the robot. It computes the coordination algorithm and uses robot's resources as sensors and communication device with the purpose to achieve information about surrounding environment and team-mates. Inside this level historical data about environment are stored like a sort of memory. The algorithm executed by the agent controls the robot through the appropriate inputs. Inside the unit the agent is the actual operative component.

At the same bottom level it is possible to insert also the obstacles. Like the Robot, they have a graphical patch but no other structure inside to be managed apart their structural data and identification number. The second important component is the **Communication Manager** (CM). It implements a sort of *postal service*: when two units move in the environment, CM constantly monitors their positions and the related communication range. When a unit enters in the communication range of another unit the CM sends a message to each unit with the ID number of the other one and provides a communication link, if one of them needs to exchange information, transporting any message.
The third component is the **Environmental Manager** (EM). This one provides the collision detection. As CM, also EM constantly monitors unit's positions and when a robot's sensor area collides with an obstacle the EM component provides the robot the collision point and its distance from the robot's center by activating the sensible points of the sensor's area.

Many other components are included in the simulator's structure but the ones just described can be considered the core of the simulator.

9.1.2 Multi-threading Computation

Java is a multi-thread software technology and it has been heavily exploited in this fashion in order to create the simulator. Indeed each software agent is a thread with it's own time of computation. This has been done to realize a simulation as close as possible to what happens in the real world. CM is also a thread, but not EM, because it is necessary to calculate collisions at each time-step of the simulation under a simulated real-time mode.

9.1.3 Scenario Loader

Simulation's environment provides a simple way to create and load new scenarios useful to test the algorithm in different situations and to validate it with more statistical results. From a Computer Science point of view, each scenario is an object inherited from an abstract class which defines the basic elements of the scenario like the number, the type of obstacles and their positions. In the same fashion we have defined the same data for the units. Creating an abstract class recognizable by the simulator as a sort of operative interface, the user can program a new scenario and activate the automatic loading by the system that, through the abstract interface, knows what elements need to be loaded. Other features are explained in [87]

9.1.4 Future Work

J.U.S.E. project is still active for educational activities but the simulator has no longer evolved. Only update patches related to any bugs found inside the code by students or researchers are still active.

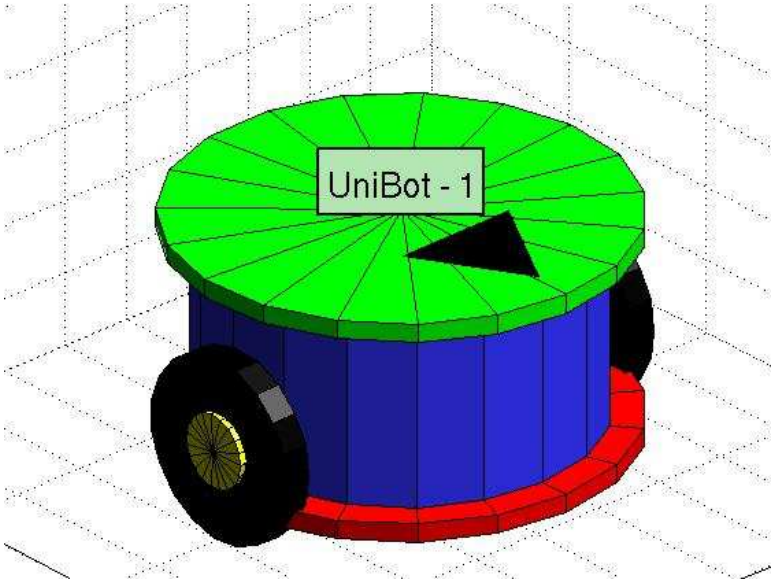


Figure 9.4: The simulated Unibot, a fundamental component for testing the techniques presented in this thesis.

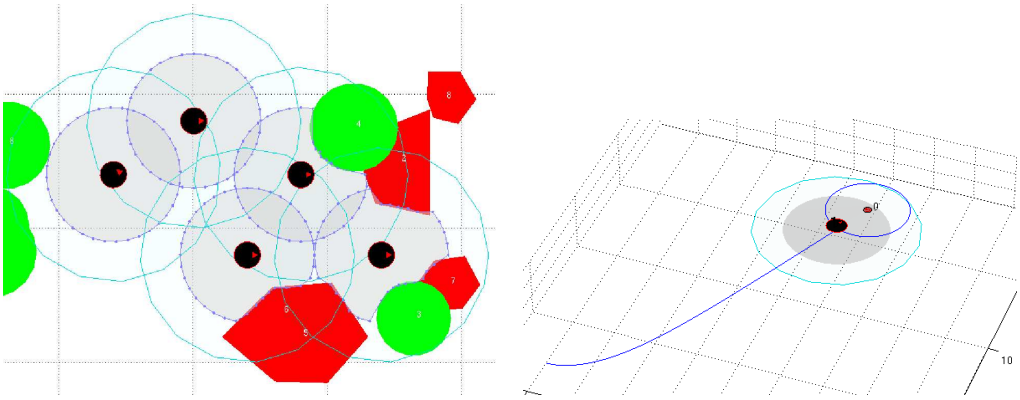
9.2 M.U.S.E.

As anticipated, M.U.S.E. (Matlab Unibot Simulation Environment) is a software simulator designed with the main purpose to give support to the Unibot control algorithms but it has been extended in order to become a valid support not only for differential wheeled robots. During the design of the simulator, the main idea was to follow the same structure of J.U.S.E., in terms of main components, as the Environmental Manager, that performs specific actions and that interacts with each other component to produce the desired result. The decision to write a new simulator has been chosen with the aim to overcome the limitation of JUSE due to the impossibility to use any other type of unit inside the arena, different from the simulated Unibot. Moreover the monolithic structure of the simulated robot and the inability to change its only single specific part in order to solve specific needs, have suggested a new representation of the object inside the environment. We have decided to build the simulated objects as pieces of Lego, modular and mountable at will. For instance the simulated 3D Unibot depicted in Fig. 9.4 is composed by seven different 3D objects: two wheels with two wheel hubs and the chassis composed by the body (the blue cylinder) and two different covers that delimit the body of the robot. Each object has been mounted with respect to the rotational axis of the object on which it is mounted, in a fixed position decided by the designer. Using roto-translation matrices, the simulator changes the positions of single or compound object. The idea was to give the object the possibility to calculate its own attitude by itself, only commanding to the object the entity of the new movement in term of Roll, Pitch, Yaw angle and X,Y,Z position with respect to the absolute reference frame. Indeed, each object is characterized not only

by its identification number, its position in the space, its vertices, its barycentre position and the center of instant rotation but even by the methods and functionalities provided, useful not only for the user but also for the other components of the simulator, that communicate with each other using a specify object interface created by different abstract classes. At the beginning, the Matlab Object Programming was quite difficult because the software was not born to this purpose but the programming structure provided by Matlab in order to use this kind of paradigm is quite simple and with some practice it is possible to handle Matlab Object with good results. There is no difference between a 2D or 3D object, in the first case the height of the 3D object is equal to zero. This way the properties remain the same for the two different representations as well as the interactions with the other components. The obstacles inside the environment have been programmed with the same technique used for the simulated objects, inheriting all the functionalities just described. Furthermore, each obstacle has the data of its own dimension, with respect to the base frame, inside. This way it is possible to define the frontiers of the obstacle in a specific plane of intersection only by asking to it the data. This feature is more useful during the obstacle detection. Finally when an obstacle is created inside the environment by the simulator, the obstacle's frontiers are stored in a modified B-Tree database, which is a well known binary structure used for store data in a useful way, in order to provide a quick search of the interested data. This database is internal to the Environmental Manager and used as support to detect if a Unit, during its motion, is close to an object or not. Indeed at each time step of the simulation, the Environmental Manager ask to the database (by producing a *Collisions Query*) what the obstacles are near to the interested Unit, in order to compare obstacles' frontiers and unit's position. Moreover, in that moment, the Unit's position has been stored in order to provide a quicker search when a new query is done at next time steps. On the way to build components useful to the simulation, after the construction of obstacles, database and raw-units, which represent a Unit from the graphical point of view only, it has been necessary to build Unit's sensors, in order to permit to the agent inside the Unit to detect the obstacles. For each robot a Sensor Device has been built, planar or spherical in relation to the type of robot and simulation. The basic sensors' structure is planar and the spherical sensor device used e.g. in Chp. 5 has been obtained just rotating the planar one on its center and collecting the data in a rack of planar sensors. An example is depicted in figure-set 9.7.

9.2.1 Sensor Device

Each robot is equipped with a Sensor Device able to detect obstacles in the surrounding environment and to provide the related distance to its agent. Each device is equipped with 36 proximity sensors equally spaced around each robot. Each sensor is built instead as a line that starts from the center of the robot and moves radial outside the unit to the end of the sensors' range. On this line a number of sensible points are distributed, on which the Environmental Manager acts the obstacle detection, verifying if this sensor point is inside or outside the obstacle. Collecting this data, a Unit can detect the presence of an obstacle. Moreover the sensible distance is freely configurable as well as the quantization of the sensors' area. The sensors' area



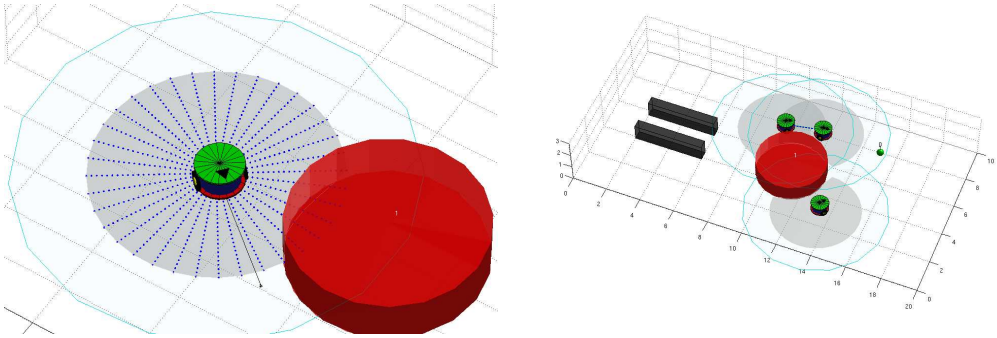
(a) (b)

Figure 9.5: Two different simulations about obstacle avoidance and tracking point test with M.U.S.E. in a 2D view

of each robot is depicted in Fig. 9.5a, deformed by collisions with obstacles, while in Fig. 9.6a the sensors' cloud during a simulation is shown. Notice that the area is linearly quantized but it is possible to choose any other function to define sensors' distances. The sensor cloud is visible only in debug mode as well as the collision ring visible in Fig. 9.5a. Being the Sensor Device a component of the structure,it can interact with the other components of the Unit exploiting the same command interface common to any other object. Moreover it is possible to remove and to substitute it without affecting any other components of the Unit. Sensor Device, as described in Chp. 6, is the component responsible for the generation and maintenance of the Gaussian field. When the Environmental Manager changes the values on the sensible points, the Sensor Device updates the field in order to provide correct environmental data. It has been adopted this sensors' representation in order to disjoint the representation from the related real structure, because it is always possible to configure a virtual Gaussian field around a robot, real or simulated, on the basis of the sensors' values.

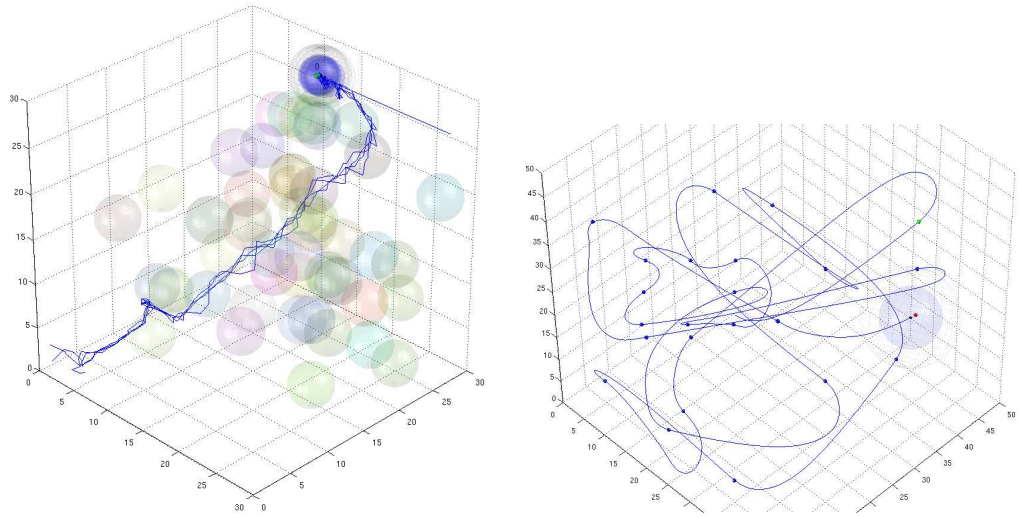
9.2.2 Communication Device

Each robot is equipped with a communication device whose activity is shown in Fig. 9.6b during an obstacle avoidance sequence. The two upper robots are connected by a blue line while the bottom one is not connected because its position is outside the communication range (i.e. the blue ring surrounding each robot). Moreover, the Communication Device stores in its *internal database* the positions of the robots encountered in the simulation, during the last m-time-steps where m is a parameter that defines the device's memory. In the simulations done in this thesis $m = 3$. This helps the Unit's agent to exploit the information on team-mates positions during the simulation, in order to forecast the trajectory at the next time step. Moreover, the communication device, on the basis of the connected robots, computes the local Laplacian matrix useful to calculate the Consensus agreement protocol [14] exploited by the technique presented



(a) (b)

Figure 9.6: The sensible points' cloud projected around the robot by the sensor device, is exploited to detect the distance from obstacles (6.4a). Normally only the sensible area is visible while the points' cloud is hidden. In the debugging mode it is possible to show the cloud. Moreover, when the robots move around, they have the possibility to communicate with each other, indeed an obstacle avoidance sequence, where communication device action is visible, is shown in Fig. 6.4b. The two upper robot are connected by a blue line while the bottom one is not connected because its position is outside the communication range (i.e. the blue ring surrounding each robot.)



(a) (b)

Figure 9.7: Simulations are fundamental for testing the designed techniques. A 3D simulation, on which the PSO technique presented in Chp. 4 is running, is depicted in 9.7a. Many simulation have been done in order to test AUV's behaviour and its orientation capability. One of this tests is shown in 9.7b.

in Chp. 6. Also for this component, a sort of environmental supervisor called Communication Manager is present, with the same meaning already described for J.U.S.E.

9.2.3 Unit's Level

As a real robot, the Unit is structured in many different levels: from a logic point of view and from the bottom to the top, at the lower level it is placed the graphical object previously described. Above the graphical object level, the robot level runs, implementing the robot's kinematics and eventually the dynamics, providing virtual motors, engine or any sort of virtual actuators useful to manage the attitude of the robot in the space. The level provides the command inputs. Above the robot level is implemented a middle-ware level, it provides a unique interface to all the robot's resources and moreover, it provides to the agent the necessary methods in order to read sensors' data, to command motors, to recover its position in the environment, to recover information by neighbours. Finally the middle-ware level elaborate the robot raw-data in order to transform them in information more comprehensible by the agent. At the top level is implemented the software agent level, where the unit's mind operates in order to compute the control algorithms

9.2.4 Software Agent

As just mentioned the software agent is the component that computes the algorithm, takes decision and manages information coming from the robot, the environment and the team-mates. From its point of view the agent is completely ignorant of the fact that it operates on a virtual robot. It is completely disjoint from the external world and it can see the surrounding environment only by the sensor's data interpretation.

9.2.5 Class Vector and Components' Systems

In order to increment the safety interaction of the components, structures that collect many identical components with the same needs have been created, following the *Iterator* pattern, such as the *Units' System* component. Indeed, it is a component that manages a population of Units and provides a large number of services for the needs of the simulator or the final user. Another example of components' system is the *Obstacles' Map* where all the obstacles are collected and managed. The *Units' System* and the *Obstacles' Map* are parts of the super components' system *Environmental Manager* that connects all the sub-components' systems in order to manage the simulation.

9.2.6 M.U.S.E. - Blender Interconnection

M.U.S.E. is continuously updated, modified and improved in order to find always better ways to create a good simulator. Despite of this, and despite of the great power of this software, it is opinion of the author that Matlab is not the answer for expanding M.U.S.E. in collision detection and hard real simulation. For these reasons M.U.S.E. has been equipped with Java

objects that provide a simple network bridge with external devices and infrastructures. One of these new possibilities of expansion regards Blender.

> From *Wikipedia*: " Blender is a free and open-source 3D computer graphics software product used for creating animated films, visual effects, interactive 3D applications or video games. Blender's features include 3D modeling, UV unwrapping, texturing, rigging and skinning, fluid and smoke simulation, particle simulation, soft body simulation, animating, match moving, camera tracking, rendering, video editing and compositing. It also features a built-in game engine."

Obviously the aim of this interaction is to exploit the powerful embedded game-engine of Blender that provides a more accurate 3D collision detection than the one integrated in M.U.S.E. Some successful interaction tests have been made (see Fig. 9.8) exploiting a bridge-code written in Python language, the native Blender interface language, connected to teh M.U.S.E. Java bridge just described.

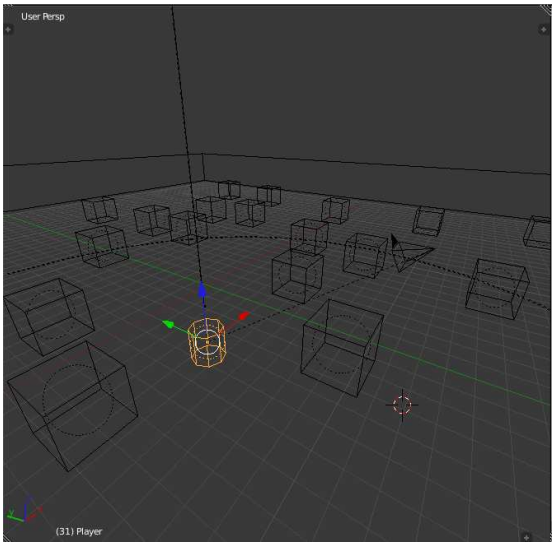
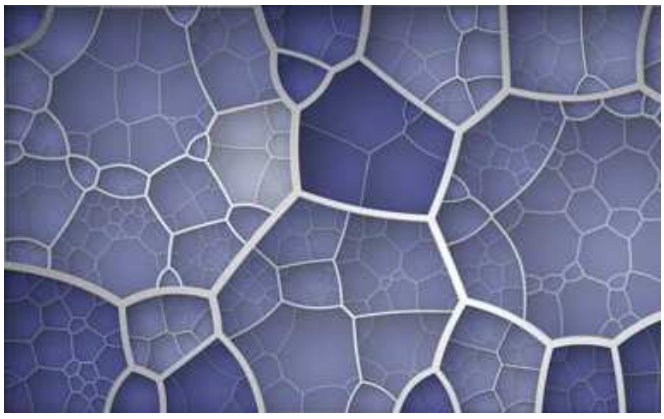


Figure 9.8: An interaction test with the game-engine of Blender through UDP/IP Internet protocol.

10

Conclusions and future work



In this work different techniques, able to coordinate and control groups of autonomous robots, have been presented. The main feature of these techniques is to drive the group of robots as a unique entity, a swarm, from a starting area to a final one avoiding the possible obstacles detected on the path. The environments on which the robots operate are a-priori unknown. The collisions detected are stored by the units and shared among the swarm in order to create a sort of *collective memory* useful to help robots in path planning toward the final destination. In the case of the technique presented in Chp. 4, the storing is only locally helpful because the information is volatile, otherwise, in Chp. 5 the information stored are not volatile and can be used to create an environmental map. The environment on which the robots have been tested are both terrestrial and submarine, in order to apply the techniques in a 2D and 3D fashion. These techniques are based on the fairly recent theory of Swarm Intelligence exploiting, in different fashions, the powerful optimization algorithm called Particle Swarm Optimization (PSO). Only in the last technique, presented in Chp. 6, the PSO algorithm has been combined with the Consensus protocol coming from the Graph Theory. The aim of this

combination derives from the idea of exploiting the intrinsic ability of Consensus protocol in regulating distributed processes, in order to control the randomness of PSO and to maintain its proved adaptability as navigation algorithm. Moreover the Consensus can be used to control the geometry of the robots' formation, which can be made as deformable as desired, and can be rigid or elastic. This way the swarm has the ability to adapt itself and to overcome the encountered obstacles maintaining the contact between team-mates. The feature of the Consensus algorithm to be able to adjust the connection rigidity between the units as desired, has given the possibility to study also undeformable geometry in order to achieve the final aim of this thesis (see Chp. 1) and moreover to design, in a future work, a distributed transportation platform, whose control technique is based on the one presented. Furthermore, in this thesis, the platform and the control infrastructures developed (see Chp. 8), with the aim to support the research activities done, have been illustrated. Finally, although in a brief and unfortunately incomplete fashion, the simulators built as the basic and fundamental support for the designed techniques have been presented.

References

[1] M. Anthony Lewis and George A. Bekey. **The Behavioral Self-Organization of Nanorobots Using Local Rules**. *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems.*, 1992. 13

[2] Tad Hogg. **Coordinating microscopic robots in viscous fluids**. *Autonomous Agents and Multi-Agent Systems*, **14**:271–305, 2007. 13

[3] G Bekey and B Khoshnevis. **Centralized sensing and control of multiple mobile robots**. *Computers and industrial engineering*, page 503506, 1998. 14

[4] D. Milutinovic, P.; Lima. **Modeling and optimal centralized control of a large-size robotic population**. *Robotics, IEEE Transactions*, page 22(6):12801285, December 2006. 14

[5] Michael Bonani, Francesco Mondada, Marco Dorigo, Stefano Nolfi, Gianluca Baldassarre, and Vito Trianni. **Self-organized coordinated motion in groups of physically connected robots**. *IEEE Transaction on System, Man and Cybernetics*, page 37:224239, February 2007. 15, 16

[6] C.W. Reynolds. **Flocks, herds and schools: A distributed behavioral model**. In *ACM SIGGRAPH Computer Graphics*, **21**, pages 25–34. ACM, 1987. 16, 19, 22, 37, 42

[7] M. Mataric. **Designing emergent behaviors: From local interactions to collective intelligence**. In *International Conference on Simulation of Adaptive Behavior*, 1992. 16

[8] T. Balch and R.C. Arkin. **Behavior-based formation control for multirobot teams**. *Robotics and Automation, IEEE Transactions on*, **14**(6):926 – 939, dec. 1998. 16, 93

[9] Ronald Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, 1998. 16, 93

[10] Magnus Egerstedt. **Graph-theoretic methods for multi-agent coordination**. *ROBOMAT, Sept*, pages 1–10, 2007. 16, 58

[11] K. J. Kyriakopoulos and D. V. Dimarogonas. **On the state agreement problem for multiple unicycles with varying communication links**. *45th IEEE Conference on Decision and Control*, page 42834288, December 2006. 17

[12] Kostas J. Kyriakopoulos and Dimos V. Dimarogonas. **On the rendezvous problem for multiple nonholonomic agents**. *IEEE Transactions on Automatic Control*, page 52(5):916922, 2007. 17

[13] Wei Ren. **Consensus strategies for cooperative control of vehicle formations**. *Control Theory and Applications*, page 1:505512, 2007. 17, 58

[14] Mehran Mesbahi and Magnus Egerstedt. *Graph Theory and Methods in MultiAgent Networks*. Princeton University Press, 2010. 17, 57, 58, 69, 108

[15] G. Beni and J. Wang. **Swarm Intelligence in Cellular Robotic Systems**. *Proceed. NATO Advanced Workshop on Robots and Biological Systems,Tuscany, Italy,*, June 2630 1989. 19

[16] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. 1999. 20

[17] J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. M. Kaufmann Pub., 2001. 20, 42

[18] Christian Blum. *Swarm Intelligence: Introduction and Applications*. 2008. 20

[19] Yuhui Shi. *Innovations and Developments of Swarm Intelligence Applications*. 2012. 20

REFERENCES

[20] Yuhui Shi. *Recent Algorithms and Applications in Swarm Intelligence Research.* november 2012. 20

[21] C. Godsil and G. Royle. *Algebraic Graph Theory.* 2001. 22

[22] J. Kennedy and R. Eberhart. **Particle swarm optimization**. In *Proc. IEEE Int. Conf. on Neural Networks*, **4**, pages 1942 –1948 vol.4, nov/dec 1995. 22, 24, 30, 42

[23] R. Eberhart and J. Kennedy. **A new optimizer using particle swarm theory**. In *Proc. 6th Int. Symp. on Micro Machine and Human Science, MHS'95.* 22, 42

[24] M. Clerc. *Particle Swarm Optimization.* Wiley-ISTE, 2006. 24, 42

[25] Riccardo Poli. **An analysis of publications on particle swarm optimization applications**. *Journal of Artificial Evolution and Applications*, pages 1–57, 2007. 24, 30, 42

[26] Konstantinos E. Parsopoulos and Michael N. Vrahatis. *Particle Swarm Optimization and Intelligence: Advances and Applications.* 2010. 24

[27] S. Burak Akat and Veysel Gazi. **Particle swarm optimization with dynamic neighborhood topology: Three neighborhood strategies and preliminary results**. *2008 IEEE Swarm Intelligence Symposium*, pages 1–8, September 2008. 25, 42

[28] J. Kennedy and R. Mendes. **Population structure and particle swarm performance**. *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02.* 25

[29] F. Bullo, J. Cortes, and S. Martinez. *Distributed Control of Robotic Networks: A Mathematical Approach to Motion Coordination Algorithms (Princeton Series in Applied Mathematics).* Princeton University Press, 2009. 26

[30] Yuhui Shi and Russell Eberhart. **A modified particle swarm optimizer**. In *In Proc. IEEE Int. Conf. on Evolutionary Computation*, 1998. 26, 30

[31] R.C. Eberhart and Y. Shi. **Comparing inertia weights and constriction factors in particle swarm optimization**. In *Proc. of 2000 Congress on Evol. Computation.* 26, 37

[32] I.C. Trelea. **The particle swarm optimization algorithm: convergence analysis and parameter selection**. *Information Processing Letters*, **85**:317–325, 2003. 27

[33] Roland Siegwart and Illah Reza Nourbakhsh. *Introduction to Autonomous Mobile Robots.* The MIT Press, 2004. 29

[34] J. Antich and A. Ortiz. **Extending the potential fields approach to avoid trapping situations**. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 1386 – 1391, 2005. 29

[35] R. Gayle, W. Moss, M.C. Lin, and D. Manocha. **Multi-robot coordination using generalized social potential fields**. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 106 –113, 2009. 30

[36] Tucker Balch and Ronald C. Arkin. **Communication in reactive multiagent robotic systems**. *Autonomous Robots*, **1**:27–52, 1994. 10.1007/BF00735341. 30

[37] C. Liu, H. Wu, G. Yang, and Z. Wei. **Path Planning of Flying Robot for Powerline Inspection Based on Improved Particle Swarm Optimization**. In *Intelligent System Design and Engineering Application (ISDEA), 2010 International Conference on*, pages 48 –52, 2010. 30

[38] E. Masehian and D. Sedighizadeh. **A multi-objective PSO-based algorithm for robot path planning**. *2010 IEEE Int. Conf. on Ind. Tech.*, pages 465–470, 2010. 30, 42

[39] D. Fox, W. Burgard, and S. Thrun. **The dynamic window approach to collision avoidance**. *IEEE Robotics & Automation Magazine*, pages 23–33, 1997. 35

[40] R. Falconi and C. Melchiorri. **A decentralized control algorithm for swarm behavior and obstacle avoidance in unknown environments**. *Proc. 2nd IFAC Work. Navigation, Guidance Control of Underwater Vehicles*, 2008. 35

[41] R. Grandi, R. Falconi, and C. Melchiorri. **UniBot Remote Laboratory: A Scalable Web-Based Set-up for Education and Experimental Activities in Robotics**. *Proc. 18th IFAC World Congress*, pages 8521–8526, 2011. 38, 40, 74

[42] S. Doctor and G.K. Vanayagamoorthy. **Unmanned vehicle navigation using swarm intelligence**. *International Conference on Intelligent Sensing and Information Processing, 2004. Proceedings of*, pages 249–253, 2004. 42

[43] Raffaele Grandi, Riccardo Falconi, and Claudio Melchiorri. **A Navigation Strategy for Multi-Robot Systems Based on Particle Swarm Optimization Techniques**. In *Proc. of the 10th IFAC Symposium on Robot Control, (SYROCO)*, September 2012. 42

[44] J. Pugh and A. Martinoli. **Particle swarm optimization for unsupervised robotic learning**. *Swarm Intelligence Symposium,*, 2005. 42

[45] J. Pugh and A. Martinoli. **Parallel learning in heterogeneous multi-robot swarms**. *2007 IEEE Congress on Evolutionary Computation*, pages 3839–3846, 2007. 42

[46] A. R. Girard, J. B. Sousa, and J. E. Silva. **Autopilots for Underwater Vehicles: Dynamics, Configurations, and Control**. *OCEANS 2007*, pages 1–6, June 2007. 42, 44, 46

[47] M. A. Joordens. **Underwater swarm robotics consensus control**. *Systems, Man and Cybernetics*, (October):3163–3168, 2009. 42, 58

[48] J. Shen and J. Zhang. **Route planning for underwater terrain matching trial based on particle swarm optimization**. *2010 Second International Conference on Computational Intelligence and Natural Computing*, pages 226–229, 2010. 42

[49] Xiaoyong Tang and Fei Yu. **Path planning of underwater vehicle based on particle swarm optimization**. *Intelligent Control and Information*, pages 123–126, 2010. 42

[50] S.B. Akat and V. Gazi. **Asynchronous Particle Swarm Optimization Based Search with a Multi-Robot System: Simulation and Implementation on Real Robotic System**. *Turkish Journ. of Electric. Engineering and Comp. Sciences*, **18**(5):749–764, 2010. 42

[51] J. Pugh and Alcherio Martinoli. **Inspiring and Modeling Multi-Robot Search with Particle Swarm Optimization**. *2007 IEEE Swarm Intelligence Symposium*, (Sis):332–339, April 2007. 42

[52] J.M. Hereford. **A Distributed Particle Swarm Optimization Algorithm for Swarm Robotic Applications**. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1678 –1685, 0-0 2006. 42

[53] J.M. Hereford, M. Siebold, and S. Nichols. **Using the Particle Swarm Optimization Algorithm for Robotic Search Applications**. In *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, pages 53 –59, april 2007. 42

[54] N.A. Chaturvedi, A.K. Sanyal, and N.H. McClamroch. **Rigid-Body Attitude Control**. *Control Systems, IEEE*, **31**(3):30 –51, june 2011. 45

[55] Gunilla Burrowes and Jamil Y. Khan. *Short-Range Underwater Acoustic Communication Networks*, chapter 8, Autonomous Underwater Vehicles. InTech, 2011. 52

[56] Riccardo Falconi, Lorenzo Sabattini, Cristian Secchi, Cesare Fantuzzi, and Claudio Melchiorri. **EdgeWeighted Consensus Based Formation Control Strategy With Collision Avoidance**. *Robotics and Autonomous Systems*, 2012. 57, 58, 69, 70

[57] Riccardo Falconi and Claudio Melchiorri. **A Graph-Based Algorithm for Robotic MANETs Coordination in Disaster Areas**. Proceeding of SYROCO 2012, Dubrovnik, 2012. 57, 58, 69

[58] J.a. Fax and R.M. Murray. **Information Flow and Cooperative Control of Vehicle Formations**. *IEEE Transactions on Automatic Control*, **49**(9):1465–1476, September 2004. 58

[59] Maria Carmela De Gennaro and Ali Jadbabaie. **Decentralized Control of Connectivity for Multi-Agent Systems**. *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 3628–3633, 2006. 58

[60] Reza Olfati-Saber, J. Alex Fax, and Richard M. Murray. **Consensus and Cooperation in Networked Multi-Agent Systems**. *Proceedings of the IEEE*, **95**(1):215–233, January 2007. 58

[61] R. Olfati-Saber and R.M. Murray. **Consensus Problems in Networks of Agents With Switching Topology and Time-Delays**. *IEEE Transactions on Automatic Control*, **49**(9):1520–1533, September 2004. 58

[62] Wei Ren and R.W. Beard. **Consensus seeking in multiagent systems under dynamically changing interaction topologies**. *Automatic Control, IEEE Transactions on*, **50**(5):655–661, 2005. 58

[63] R.W. Beard and E.M. Atkins. **Information consensus in multivehicle cooperative control**. *IEEE Control Systems Magazine*, **27**(2):71–82, April 2007. 58

[64] Luca Scardovi, Naomi Ehrich Leonard, and Rodolphe Sepulchre. **Stabilization of collective motion in three dimensions: A consensus approach**. *2007 46th IEEE Conference on Decision and Control*, (2):2931–2936, December 2007. 58

[65] Asuman Ozdaglar and Pablo A Parrilo. **Constrained Consensus and Optimization in Multi-Agent Networks**. *IEEE Transactions on Automatic Control*, **61801**, 2010. 58

[66] Marriott Waterfront. **Decentralized Consensus Based Control Methodology for Vehicle Formations in Air and Deep Space**. *Control*, pages 3660–3665, 2010. 58

REFERENCES

[67] DA Paley, NE Leonard, and Rodolphe Sepulchre. **Oscillator models and collective motion**. *Control Systems*, (August):89–105, 2007. 58

[68] R. Olfati-Saber. **Flocking for multi-agent dynamic systems: algorithms and theory**. *Automatic Control, IEEE Transactions on*, **51**(3):401–420, March. 58

[69] Bruno Siciliano and Oussama Khatib. *Handbook of Robotics*. Springer, 2008. 68

[70] H. Ferdinando, H. Khoswanto, D. Purwanto, and S. Tjokro. **Design and evaluation of two-wheeled balancing robot chassis: Case study for Lego bricks**. In *Innovations in Intelligent Systems and Applications (INISTA), 2011 International Symposium on*, pages 514–518, June 2011. 85

[71] M. Pinto, A.P. Moreira, and A. Matos. **Localization of Mobile Robots Using an Extended Kalman Filter in a LEGO NXT**. *Education, IEEE Transactions on*, **55**(1):135–144, February 2012. 85

[72] M. Casini, A. Garulli, A. Giannitrapani, and A. Vicino. **A LEGO Mindstorms multi-robot setup in the Automatic Control Telelab**. *Proc. 18th IFAC World Congress, Milano, Italy.*, pages pp. 9812–9817, August 2011. 85

[73] D. Benedettelli, M. Casini, A. Garulli, A. Giannitrapani, and A. Vicino. **A LEGO Mindstorms Experimental Setup for Multi-Agent Systems**. *Proc. 3rd IEEE Multi-Conference on Systems and Control,Saint Petersburg, Russia*, pages pp. 1230–1235, July 2009. 85

[74] T. Wolf. **Assessing Student Learning in a Virtual Laboratory Environment**. *Education, IEEE Transactions on*, **53**(2):216 – 222, may. 2010. 89

[75] M. Casini, D. Prattichizzo, and A. Vicino. **The automatic control telelab: a user-friendly interface for distance learning**. *Education, IEEE Transactions on*, **46**(2):252 – 257, may. 2003. 90

[76] J. Fernandez and A. Casals. **Open laboratory for robotics education**. **2**, pages 1837 – 1842, apr. 2004. 90

[77] N. Swamy, O. Kuljaca, and F.L. Lewis. **Internet-based educational control systems lab using NetMeeting**. *Education, IEEE Transactions on*, **45**(2):145 – 151, may. 2002. 90

[78] G.T. McKee. **The development of Internet-based laboratory environments for teaching robotics and artificial intelligence**. **3**, pages 2695 – 2700, 2002. 90

[79] D.W. Calkin, R.M. Parkin, R. Safaric, and C.A. Czarnecki. **Visualisation, simulation and control of a robotic system using Internet technology**. pages 399 – 404, jun. 1998. 90

[80] Ken Taylor, Barney Dalton, and James Trevelyan. **Web-based telerobotics**. *Robotica*, **17**(1):49–57, 1999. 90

[81] T. Lochmatter, P. Roduit, C. Cianci, N. Correll, J. Jacot, and A. Martinoli. **SwisTrack - a flexible open source tracking software for multi-agent systems**. pages 4004 –4010, sep. 2008. 91, 95

[82] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J. C. Zufferey, D. Floreano, and A. Martinoli. **The e-puck, a Robot Designed for Education in Engineering.** In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, pages 59 – 65, 2009. 91

[83] Valentino Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. The MIT Press, February 1986. 93

[84] G. Antonelli and S. Chiaverini. **Kinematic control of a platoon of autonomous vehicles**. **1**, pages 1464 – 1469 vol.1, sep. 2003. 93

[85] G. Antonelli and S. Chiaverini. **Kinematic Control of Platoons of Autonomous Vehicles**. *Robotics, IEEE Transactions on*, **22**(6):1285 – 1292, dec. 2006. 93

[86] J. Pugh, X. Raemy, C. Favre, R. Falconi, and A. Martinoli. **A Fast Onboard Relative Positioning Module for Multirobot Systems**. *Mechatronics, IEEE/ASME Transactions on*, **14**(2):151 – 162, apr. 2009. 99

[87] Raffaele Grandi. *Coordinamento di uno sciame di robot tramite algoritmi di Particle Swarm Optimization*. Master's thesis, Alma Mater Studiorum - University of Bologna, February 2009. 101, 102, 103, 105