UNIVERSITÀ DEGLI STUDI DI CAGLIARI

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA SPECIALISTICA IN INGEGNERIA ELETTRONICA

Dipartimento di Ingegneria Elettrica ed Elettronica

Nancy-Université INPL

Centre de Recherche en Automatique de Nancy

ENSEM - INPL

ALGORITHM FOR MULTI-VEHICLE GUIDANCE ON A FIXED LATTICE FORMATION

Advisors

Prof. Jamal Daafouz

Prof. Alessandro Giua

Prof. Claude Iung

Master Thesis by:

Andrea BACCARA

Academic Year: 2007-2008

Theory is when all is known and nothing works.
Practice is when all works but no-one knows why.
We have joined theory and practice:
nothing works... and no-one knows why!
Albert Einstein

Contents

Chapter 1

Introduction

Goal of this document is to introduce a new algorithm for the control of a group of mobile robots in an unknown horizon with obstacles, moving in a fixed lattice formation. Cooperation between multiple agents working on the resolution of a common objective is quite a problem today. Indeed many agents may efficiently reduce the length of the mission. We will try to highlight the most current problems typical in literature as well as the solution that can be applied in certain environments such as transports (highways, cars, satellites, ships...), health (tele surgery), communication systems, operating system design and others.

Recent works, like the ones considered, in this thesis, extend the old algorithms to address the computational complexity of the on-line optimization. The main improvement is given by a division of the algorithm in few levels, where each level can compute a different command for the agent. New technologies like multi-threading softwares and multi-processor hardware enable the use of much shorter planning horizons and quicker-solver algorithms. We are hereby considering a formation of non-holonomic robots without leaders: the control's rules must be obtained by local or global information; we also consider a car-like agent described by unicycle robots over a $2D$ space. We consider the works of Defoort [3] like the reference papers for the agent description: we consider the system in discrete time and a continuous 2D

space. The same problem can be generalized for a generic agent on a $\Re^n$ space, generating the same kind of problems. Centralized systems are easily implemented, although implementation results in an exponential growth of the computational time based on the size of the group: this document will try to focus on a decentralized approach of the problem. Neither global coordination will be needed, or a leader. All the rules are given *a priori*, and decisions will be taken by computing the local information of the sensors and the external information received from the others robots, exchanged via communication systems. It is highly important to focus on the communication problems like band or power availability. In order to have a simple notation we will not consider the delays of communication. These problematics grown linear - not exponential - with the number of agents considered. Finally the goal is to have group arriving at destination, ensemble, yet we do not want to avoid the obstacles in a formation lattice. The avoidance of the obstacle in fixed formation, like in the previous analyzed works [12], is simpler tough not better choice as it would require much more space around the obstacle than needed. On the other hand the classical flocking algorithms [13] do not allow to give an order to the flock, like in the problem of *military formation*.

The original contribution of this work is the introduction of a switched control system for the guidance of the robots. Some controllers have been applied, both developed in this thesis and found in bibliography, and an high level state machine that chooses, in real time, the better strategy to be adopted. Moreover the geometrical approach of the algorithm can minimize the computational load and the sensor's needs. Group stability and safety will be guaranteed by a controller based on the geometrical theory of Voronoi. This controller in according with the theory of Lindhe and Johansson [12] is completely decentralized and is light from a computational point of view.

The second chapter of this thesis will cover all those situations where one may possibly find an agent-interaction control problem without leader or global control. A short review of the article, or the articles, will follow, considering every kind of problem, without going into the details of the so-

lutions adopted in the considered papers. We will consider of guidance over an horizon with obstacles, a problem of guidance in a formation and the consensus problem, which can be divided in: rendez-vous problem, velocity consensus and finally coordination problems. In chapter three we will describe the our considered system, and a modeling of it will be done; a formal and an informal explication of the problems that may be encountered will be drawn.

In chapter four we will take the model of chapter two, of the read articles, and solve the problem. Using a similar technique with [3], this paper develops an algorithm for guidance control. The problem is mainly practical, and the physic of the system must be respected. This point can help us to use a simple geometrical approach for the guidance algorithm that decreases once more the computational load. A high level state machine allows one to switch between the controllers when an obstacle will be found.

Finally in chapter five a simulation with the software Matlab is presented. Some testing will be done aiming to demonstrate the efficiency of the algorithm, the stability of the system and the low computational load requested for each robot. In Appendix A one will find a transcription and an explication of the algorithm developed for the simulation. Appendix B contains the geometrical formulas used in this thesis, and finally in Appendix C some basic issues about the notation of agents and observability are recalled.

Chapter 2

Independent agent guidance

In this first section we will focus on the main situations that can be found over an agent-interaction control problems. An agent is a structure evolved by a given (non)linear dynamic and a possible additive noise. Each agent can control its own dynamics. The goal is to minimize the accumulated joint cost, consisting in a state dependent term and a term that depends on the control.
We can find agent's control problematic in some different application, over a space from 1 to n dimension space. Despite it, the problems are always the same: for this reason in this chapter we will not go into details of the specific solutions found on the literature but we will give a general theory for each kind of problematic. When not specified we always consider non-holonomic agents (see Appendix C).

2.1 Trajectory decision over an horizon with obstacles

In this section we cover the problem of the cooperative control of a group of agents, or [3] mobile and autonomous robots, over an unknown horizon. The presence of constraints for all the agents says that we can avoid collisions only if we can ensure the communication between the agents. Moreover,

since the available power is limited, the distance between the agents and the information exchanged are reduced at best. We don't consider here other aspects of communication like dead zones, latency, ecc.
The problem is to compute the trajectory respecting the constraints over the initial and final configuration. The environment is not *a priori* known and then must be found in real time.

2.1.1 Model dynamics

In this chapter and in the follows the index i, indicates the agent index, for a total of N agents; k denotes the current time step. The dynamic can be described [11] by a LTI model in continuous time:

$$\dot{q}_i(t) = A_i q_i(t) + B_i u_i(t) + \eta_i(t) \tag{2.1}$$

and in discrete time:

$$q_n(k+1) = A_i q_i(k) + B_i u_i(k) + \eta_i(k) \tag{2.2}$$

or by a non-linear model as:

$$\dot{q}_i(t) = f(q_i(t), u_i(t), \eta_i(t)) \tag{2.3}$$

for $i = 1, \ldots, n$, where $q_i(k) \in \Re^n$ is the state vector, $u_i(k) \in \Re^m$ is the input vector and $\eta_i(k) \in \Re^n$ is the disturbance vector for the vehicle i. $\eta_i(k)$ is unknown zero-mean variable, bounded by η_{max}. We do not consider now the second level dynamic of the agents, hence we will not introduce the effects of inertia on top of our problems. In chapter three we will give a formulation of these effects on the mathematical computation, considering them as negligible.

General output is bounded by local constraints then bounded by $\mathcal{Y}_i$:

$$y_i(k) = C_i(k) + D_i u_i(k) \in \mathcal{Y}_i \tag{2.4}$$

For each agent we also have constraints over the maximum, for each component $l \in \{1, \ldots, m\}$ of the input u_i.

$$|u_{i,l}(t)| \leq u_{i,l,max} \tag{2.5}$$

It's widely considered in literature that the state

$$q_i(t) = \begin{bmatrix} p_i(t) \\ \theta_i(t) \end{bmatrix} \tag{2.6}$$

simply represent the position $p_i(t)$ and the orientation $\theta_i(t)$ of our agent over the n-dimensional space: our notation here under will consider this convention. It is important that the trajectories of the agent never collide with each other so, in order to ensure that:

$$||p_i(k) - p_j(k)|| \geq S \tag{2.7}$$

where $i, j \in \{1, \ldots, n\}$ and $i \neq j$. S is the minimum gap between two agents at the k time.

2.1.2 Obstacle avoidance

Let us now suppose to be facing the problem of the obstacle avoidance sensor-based. Obstacles can be either physical or merely non-driving zones. The controller for the obstacle avoidance used alone, does not guide the agent to a destination state but can prevent it to drive into an obstacle. There are two families of controllers: the first is a reactive on-line controller [9] and the second one is an off-line controller [10] based on a replan of the trajectory.

Each agent i finds an ensemble $\Omega_i(t) = [O_1, O_2, \ldots, O_{M_i}]$ of M_i obstacles. The ensemble is time varying because the sensor finds new obstacles with the movement of the agent. The goal is not to collide with the obstacles. We can see each obstacle O_o as a infinite set of points:

$$distance(p_i(t), O_o(t)) > \rho_i \qquad \forall O_o; \forall o \in 1, \ldots, M_i \tag{2.8}$$

See Appendix B for the function *distance*. ρ_i indicates the radium of the agent while o_o indicates the position of the generic (external) point in the obstacle.

2.1.3 Path following and waypoints

Once given an initial position and a destination state there are many ways to reach the goal. The agent can arrive directly at destination minimizing the distance; it can be required to follow a path or to reach some intermediate waypoints before reaching the goal (or the possible route of the agent can be bounded). If we got waypoints the optimum control problem associated to each i agent is to find the input u_i such as:

$$min_{u_i} \int_{t_k}^{t_k+T_d} (\alpha||p_i(k) - w_i||^2 + \beta||u_i(k)||^2)dt \tag{2.9}$$

where w_i is the next waypoint for the agent i. α and β are just correctional proportional factors. The problem in (2.9) can be bounded by all the constraints as (2.7), (2.5) and (2.8). If we have a goal to be reached without intermediate waypoints [9], we'll have a law like: $q_d = q(s)$ with $0 \leq s \leq s_f$, and q_d that indicates the desiderate state to be reached, then a path to follow bounded with a range s_f (independent in all the dimension of the space), the control objectives will be:

$$lim_{t \to +\infty} \delta(t) \leq \epsilon_\delta \tag{2.10}$$

with ϵ_δ is a positive number arbitrarily small, $\delta(t)$ is the distance with the path at time t:

$$\delta(t) = ||q(t) - q_d(t)||$$

We also cite [15] and [8] where you can find a virtual vehicle approach for guidance over a given path, ruled by differential equation with error feedback, and a dynamic path following approach with the criterion to stay close to the path to have a great strength against measurement errors and external disturbance. In flocking [13] the waypoint is considered as a special agent,

where the potential with another agent is an attractive one. In literature we can find some solutions for the problem explained. Depending on the particular case analyzed, indeed, it is possible to face the physics of the problem to define the optimal controller.

2.2 Formation mobility

Here we will present the problem of stability control of a formation of autonomous agents. The task is to find a control with an approach which yields a good fixed formation convergence and disturbance rejection. For the moment we consider each as a point over the space, then it will be defined only by its center

$$q_i(t) = \begin{bmatrix} x_{i,1}(t) \\ \dots \\ x_{i,n}(t) \end{bmatrix} \tag{2.11}$$

Considering N robots placed over the n-dimensional space, equipped with sensors for the relative distance with the other robots, that follows that simple given discrete dynamic on $\Re^n$:

$$q_i(k+1) = q_i(k) + u_i(k) \tag{2.12}$$

The goal is to find a control law which satisfies the following aims:

- Agents should never collide with each other;
- Formation should be asymptotically stable;
- Algorithm should be decentralized;

We can describe as a vector the state χ of the formation:

$$\chi = \begin{bmatrix} q_1^T & \dots & q_N^T \end{bmatrix}^T = \begin{bmatrix} x_{1,1} \\ \dots \\ x_{1,n} \\ \dots \\ x_{N,1} \\ \dots \\ x_{N,n} \end{bmatrix} \tag{2.13}$$

To ensure the collision rejection we have to ensure the respect of a security distance d between the robots as defined in (2.7). In this section we will not consider the high level algorithm that will computes the waypoints. Here we only consider a group of *holonomic* agents.

2.2.1 Flocking - global interaction

Flocking [13],[16] is a collective behavior of large number of interacting agents with a common group objective. In flocking we can ensure the three points listed in the previous subparagraph with an algorithm of velocity consensus: see section 2.3.2. Moreover a controller algorithm it is necessary to ensure the achieving of the common objective. Fixing a distance

$$d(t) = ||p_i(t) - p_j(t)|| > 0 \qquad \forall i \in \mathcal{N}_i(t) \tag{2.14}$$

between an agent and the others, defining the neighbors set of i as $\mathcal{N}_i(t)$. We can introduce the notion of communication graph as follows

Definition 2.2.1 *a communication graph $G = (V, E)$ is a undirected graph consisting of a set of N vertices $V = \{1, \dots, N\}$ and a set of edges $E = \{(i, j) \in V \times V | i \in N_j\}$ containing pairs of nodes that represent interagent communication specification. The graph is undirected because $(i, j) \in E \Leftrightarrow (j, i) \in E$. The adjacency matrix $\mathcal{A}(G) = [a_{ij}]$ of the undirected graph G is then a symmetric matrix with $a_{ij} = 1$ if the two vertices i and j are neighbors, and $a_{ij} = 0$ otherwise.*

For simplicity we will always consider undirected communication graphs. In flocking, the lattice can be completely connected or quasi-connected; in this last case we have a lattice with some holes. Inside the lattice, all the agents respects the defined constraints of the velocity (2.5) and of the distance between each other (2.14).
Usually the two kinds of lattice are made together by the flocking algorithm. If we want a stable, but ordinate lattice group of agent we need to use an approach of "military formation" as explained in the next section.
To ensure the stability of the group, we can introduce a potential function that depends on the distance between two agents.

$$V_i = \sum_{j=1, i \neq j}^{N} V_{i,j}(d_{i,j}) \tag{2.15}$$

where $d_{i,j} = |p_i - p_j| = d_{j,i}$, and $V_{i,j}(d_{i,j})$ represents a decreasing potential function that ensure the stability.

2.2.2 Flocking - local interaction

Global interaction requires a lot of communication power and band availability in order to ensure the need to have the state of any agent available to the others. A more interesting approach is then to reduce communication needs, allowing only interactions between neighbors. Defining the neighbors set as the set $\mathcal{N}_i(t)$ of all the agents j whose $||p_i - p_j|| < d$.
The graph sketched in definition 2.2.1, will be changed with a neighbors graph that will have a number $\mathcal{N}_i(t)$ of vertex and a consequent, reduced, number of edges. Also our potential function (2.15) will be then modified as:

$$V'_{ni} = \begin{cases} V_{i,j}(d_{i,j}) & d_{i,j} < d, \\ V_{i,j}(d) & d_{i,j} \geq d \end{cases} \tag{2.16}$$

2.2.3 Fixed lattice formation

The fixed lattice formation controller builds a well ordinated formation lattice - like an army corp - in the way chosen by the implemented algorithm. The

net movement of the group should be externally controllable (by manual or high level controller), in order to move the whole formation around obstacles and choose a goal. This high-level control change the value of a constant factor - depending on the form of the algorithm - that moves the whole group in the chosen direction.

2.3 Consensus of agents

The third problem that we are about to present is called in the literature "Consensus Problem" [1],[6]. Consensus gathers the experiences from the whole group: in consensus the input of every agent is carefully considered and an outcome is crafted that best meets the *needs* of the group. Using updates in discrete time it is possible to reach the consensus using algorithm of weighted average of the state of the agents. It is important to have at least a partial observability of the scenario i.e.: assuming the observed output defined as $y_i(t) = Cq_i(t) + \eta_i'(t)$ we have to ensure the classic condition of observability over the matrix C (see Appendix C.2). It's here considered an output $y(t)$ afflicted by noise ($\eta'(t)$) Describing each agent in discrete time by its state as in (2.2), the average consensus algorithm takes the general form of:

$$\begin{aligned} z_i &= (1-l)\cdot(q_i(k)) + l\cdot(y_i(k)) \\ q_i(k+1) &= Q_{i,j}z_i \end{aligned} \tag{2.17}$$

Where [2] $0 < l < 1$ is the optimal gain that depends on the noise co-variance. Q is the Consensus matrix with proprieties:

- $Q_{i,j} \geq 0$;
- $\Sigma_i Q_{i,j} = 1$;
- $\Sigma_j Q_{i,j} = 1$,

The form of Q depends on the kind of mean we want to introduce in the consensus algorithm. We consider three types of consensus: the first called

Rendez-Vous has the goal to lead all agents in a consensus point in the space; the second one *Velocity consensus* ensures homogenous speed to all agents; and the last one - *coordination problem* considers as state exclusively the direction of the agents without taking care of their position in the space. It is also possible combine the three problems in a generical form, however we will analyze each problem separatly to have a simple notation. Communication problems - that should be accounted for - are not being considered in our analysis.

2.3.1 Rendez-vous in multi agent systems

The problem is to drive a group of agents towards a consensus point of the space in an appointed finite time. Then we have to design a controller that, with no communication or with a limited one, can drive our multiple agents to a certain point, ensuring that they arrive at destination only once and all at the same time.
Agents may not "agree" to the rendezvous location and therefore they will have to adapt to different conditions, react accordingly, and learn how to avoid the factors that complicate the process of rendezvous. Defining the agents with (2.6) and the configuration space like on (2.13), the design objective is to construct feedback controllers that lead multi-agent system to rendezvous, i.e. all agents should converge to a common point q^* in the state space defined by $p_i(t)$
We can have a Radio-based approach, in this case each agent can receive news from the others using a communication system (see definition (2.2.1). As defined above, each agent is aware of the state of only those agents belonging to its communication system in the considered time instant. The control law in discrete time will be (defying $u_i(k) = [\nu_i(k)\omega_i(k)]$) in the form [5]:

$$\begin{aligned} \nu_i(k+1) &= \nu_i(q_i, q_j, t_k) \\ \omega_i(k+1) &= \omega_i(q_i, q_j, t_k) \end{aligned} \tag{2.18}$$

that accounts only for the state of the agent considered and of its neighbor to compute the new command law. This approach is so easy to implement but requires focus on all the communication different problematics such as band allowed, power management and noise. Another approach is the one sensor based, where each agent must have the same horizon perception and a good landmark. Obviously, the fundamental rendezvous strategy implemented in each agent must be the same. They will compute their command law basing on the perception, with sensors, of the state of the other agent. Algorithm to solve this problem can be of many types. In [14] we have a list and a comparison of each algorithm type, from the deterministic to the probabilistic ones.
It is no question of better or worse algorithm: the success of an algorithm will depend upon the physics of the problems to be faced: sensor noise, sensor dissimilarities, a-synchronicity and non identical landmarks.

2.3.2 Velocity consensus

Let us consider a finite number N of autonomous ideal punctiform agents, then described by (2.2). Let us suppose a uni-dimensional space for simplicity: all the agents will move in the same direction but at different speed. The request is to have, in a finite time, that all agents move asymptotically at the same velocity. The problem is extendable without noticeable changes to the n-dimensional space. We want to avoid the trivial solution $\nu = 0$ (ν represents the linear speed of our agent), so:

$$\begin{cases} \sum_{i,j \in N_i(k)} a_{ij} ||\nu_j - \nu_i||^2 \leq 0 \\ \nu_i(t) \geq \nu_m in \quad \forall i \in N_i(k); \;\; k > 0 \end{cases} \tag{2.19}$$

where a_{ij} is a constant that depends on the position of the agents. We've also introduced a constant $\nu_m in$ to avoid an asymptotical decreasing solution.

2.3.3 Coordination problems

We consider the same N agents of section 2.3.2 but now we suppose that they move in the n-dimensional space at the same linear speed $\nu_i = \nu$, but with different and variable heading $\theta_i \in [0, 2\pi[$. Considering the discrete-time evolution of the whole system we can modify the direction of each agent at each time step, according to its velocity. We want all the agents to coordinate themselves towards an all-together direction θ_d not *a-priori* given but computed step by step on the basis of the actual direction of the agents. To know the state of others agents we have to build directed weighted graph that represents the communication between neighbors agents.
The updated direction of each agent is given by a weighted average of the direction of its neighbors $\mathcal{N}_i(k)$, meaning [17]:

$$\theta_i(k+1) = \frac{\theta_i(k) + \Sigma_j w_{ji}(k)\theta_j(k)}{1 + \Sigma_j w_{ji}} \qquad j \in \mathcal{N}_i(k) \tag{2.20}$$

w it's just a weight for the link between the two agents i and j at t. This equation is nothing but approximation, depending on the chosen model. In this case, we are facing a stability problem of a linear time-varying switched system described by:

$$\theta(k+1) = A_{i,l}\theta(k) \quad \text{with} \quad A_{i,l} = \begin{cases} \frac{1}{1+\Sigma_j w_{ji}} \text{ if } l = i \\ \frac{w_{li}}{1+\Sigma_j w_{ji}} \text{ if } l \neq i \end{cases} \tag{2.21}$$

Furthermore, we have a system without perturbation as (2.2)where the state $q_i(t)$ represents - we are not interested in the position - only the direction of each agent i with $B_{i,j} = 0$ for $\forall j \in 1 \dots m$. We got some advantages over the other problematic. Indeed here we can simplify our system knowing that the A matrix is stochastic and square; knowing that its rows all sum to 1 with non-negative entries and positive on the diagonal. In case of non-perturbation we will not have to care about possible stability problems.

Chapter 3

System modelling

We consider a group $\mathcal{R}$ of N agents that will move over a bi-dimensional, continuous and infinite horizon in a discrete time. Any agent will compute each independent trajectory, knowing only where the waypoint $\mathcal{W}$ is and with the only help of his sensors. As already stated, we do not have leader neither do we have a global sensor or controller.
Every agent $R_i \in \mathcal{R}$, with $i \in \{1, \ldots, N\}$, it is an unicycle robot with two independent drive wheels like in picture 3.1. The robots are initially placed as a formation and the goal is to arrive to destination in the same formation form that they were placed as per picture 3.3. By the way, they have to arrive to destination via the shortest way and in a finite time, or better in the shortest time possible.

3.1 Robots modeling

It's possible to describe each robot like a circle with radius ρ and center x_i, y_i; θ_i describe the orientation of the robot. Then we can describe each robot as:

$$\begin{cases} \dot{x}(t) = \nu(t)\cos\theta(t) \\ \dot{y}(t) = \nu(t)\sin\theta(t) \\ \dot{\theta}(t) = \omega(t) \end{cases} \qquad with\ \theta \in [0, 2\pi[\tag{3.1}$$

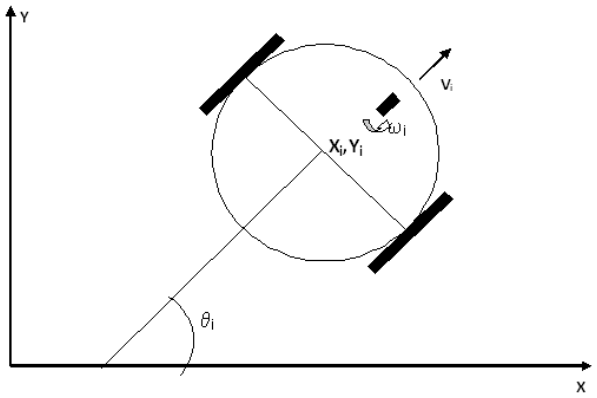

Figure 3.1: Unicycle car-like robot

$$q(t) = \begin{bmatrix} x(t) \\ y(t) \\ \theta(t) \end{bmatrix} \tag{3.2}$$

where ν and ω are the linear and the angular speeds respectively bounded by:

$$|\nu_i(t)| \leq \nu_{max} \quad |\omega_i(t)| \leq \omega_{max} \quad \forall i \in \{1, \ldots, N\} \tag{3.3}$$

Then in a compact form:

$$\dot{q}(t) = B(q) \cdot u(t) \tag{3.4}$$

With $B(q) = \begin{pmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{pmatrix}$ and

$$u = [\nu, \omega] \tag{3.5}$$

It can be useful also have a reduced form, not considering the dimension of the robot, so that its angular speed can be ignored:

$$p(t) = \begin{bmatrix} x(t) & y(t) \end{bmatrix}^T \tag{3.6}$$

3.1.1 Inertial effects

To complete the description of the system we give also a second degree dynamic [4]. Here we consider the inertial effects and give the dynamic in terms of acceleration: those effects, however, will be taken as negligible, in order to simplify the algorithm.

$$M\ddot{q}(t) = B(q)\tau - A(q)\lambda \tag{3.7}$$

where M in the inertial matrix, B is the transformation matrix (to change from cartesian to polar coordinates), λ represent the constraints on the system as a vector and finally τ gives the command input as the torque control of the wheels.

3.1.2 Discrete time

Till here, all the laws are in a continuous time. Now we define the same model in a discrete time, how we will use it in the next chapter for the algorithm, and obviously how we have used it in the simulation. The dynamic law will be formulated as follows:

$$q(k+1) = q(k) + B(q) \cdot u(k) + \eta(k) \tag{3.8}$$

Our time-line t is then sampled with a sampling time t_s and bounded by a t_{max}. The index of the sampling time is indicated by k. η is a random unknown perturbation over the system, bounded by $\eta \leq \eta_{max}$. We suppose that the computing time of all our algorithm plus the refresh time and the intercommunication time, result in a lesser time than the sampling time t_s.

3.2 Obstacles

M obstacles have been placed on the robots' trajectory. Placement of the obstacles is unknown to the robots till their sensors detect it. The ensemble of M obstacle $\Omega = \{O_1, O_2, \ldots, O_o, \ldots, O_M\}$ is fixed. We need not know an

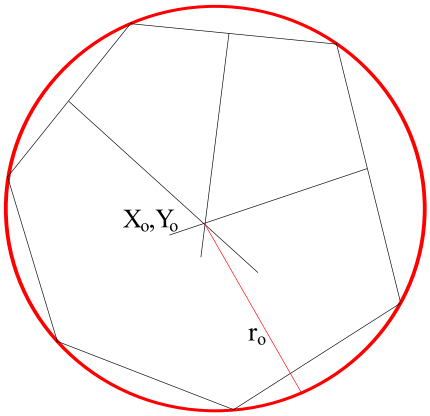

Figure 3.2: Circular obstacle starting from a generic convex one.

a-priori trajectory, but only the obstacles that can cause crash in the next time step. Obstacles can be of two types: convex obstacles and infinite walls. By the combination of this two types we can define all the possible situations.

3.2.1 Convex obstacles

Every convex obstacle is taken as circular (see picture 3.2) then defined by his center $[X_oY_o]$ - computed as centroid of the convex obstacle - and his radium r_o, then the following inequalities are just an application of (2.8). Even a neighbor robot may be taken as an obstacle, if it is placed over the path. Once the robot finds the obstacle in his way, the sensor gives its coordinates (position and radius) to the controller then the agent compute a crash-free trajectory. We shall call $\mathcal{O}_i(t) \subseteq \Omega$ and $\mathcal{Q}_i(t) \subseteq \mathcal{R}$ the set of obstacles and robots respectively than may crash with the robot i at time t. We write $\mathcal{C}_i(t) = \mathcal{O}_i(t) \bigcup \mathcal{Q}_i(t)$, and r_{c_o} the radium of the object $c_o \in \mathcal{C}_i(t)$ then we want to ensure the following constraints:

$$distance(p_i(t), O_o(t)) > \rho + r_o \quad \forall O_o(t) \in \mathcal{O}_i(t) \tag{3.9}$$

$$distance(p_i(t), p_j(t)) > 2\rho \quad \forall p_i \in \mathcal{Q}_n(t) \tag{3.10}$$

ρ as ever represents the radius of each robot. We can resume the two conditions above in only one condition as follows:

$$distance(p_n(t), c_i(t)) > \rho + r_{c_i} \quad \forall c_i \in \mathcal{C}_n(t) \tag{3.11}$$

In the same way each robot needs to know the position of the neighbor robots when it is in formation, but does not need the position of the other robots when the formation is broken to avoid an obstacle: that must be anyway under radio coverage.

3.2.2 Walls

We consider only infinite walls. Indeed an eventual finite wall can be described by the sensor of the robot like an infinite wall in t_{k-1} and like an empty space in t_k where t_k is the time where the sensor stop to see the wall like a problem for the robot's trip in the next sampling time. We can observe that the half space is a particular case of convex obstacle. Indeed an half space is always a convex connected region.
An infinite wall divide the horizon in two parts. Describing our wall like a straight line $y = ax+b$ we can ensure that the robot is in the closed half-space considering it as the center of cartesian system the position of the robots

$$y - y_i = a(x - x_i) + b \tag{3.12}$$

where b is the intersection of the infinite wall with the new y-axe. If $b > 0$ all the infinite straight lines with same a and values of $b' > b$ generate the non-permitted half-space. To avoid crash we need to compute the distance between this straight line and the origin (see Appendix B.3):

$$d_w = \frac{-b}{\pm\sqrt{1 + a^2}} \tag{3.13}$$

We must then simply ensure that this distance is less than than the dimension of our robot:

$$d_w < \rho \tag{3.14}$$

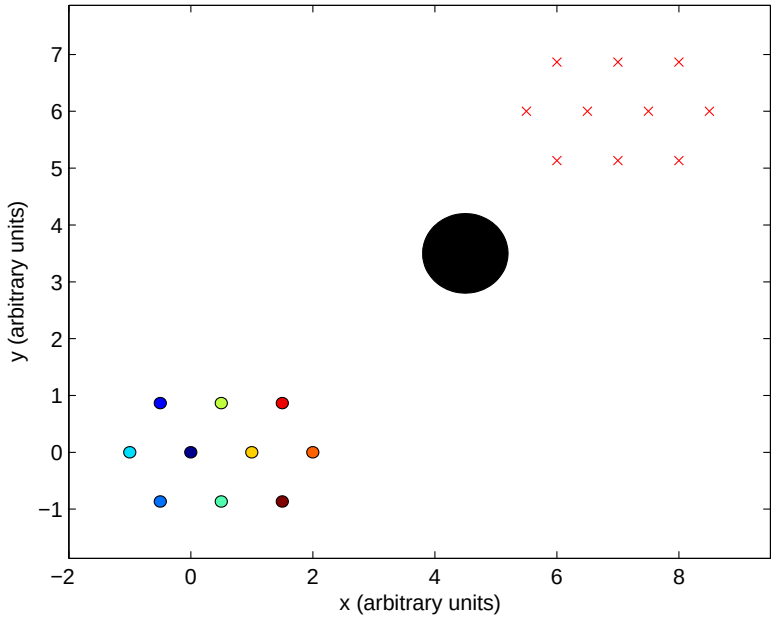

Figure 3.3: Formation: the black circle indicate the obstacle, while the red X indicate the waypoints.

3.3 Destination

Once given the destination point $\mathcal{W} = [X_w, Y_w]^T$ and placed each robot R_i in a state $q_i = [x_i, y_i, \theta_i]^T$, each robot R_i do not will arrive exactly in $\mathcal{W}$, but in a point $\mathcal{W}_i$ computed as follows:

$$\mathcal{W}_i = \mathcal{W} + (\hat{q}_{i,0} - CG_{form}(t = 0)) \tag{3.15}$$

as we can see in figure 3.3. CG_{form} is the center of gravity of our lattice formation. At this point we have to compute the problem (2.9) with $\alpha = 1$ and $\beta = 1$. If we do not have obstacles in our route, the shortest way will be a straight line joining the agent with the waypoint, but this will not be the real trajectory because we do take into account the finite, angular speed of the robot (see also picture 3.4) and the initial orientation of each robot $\theta(t = 0) = \theta_0$. Indeed the problem (2.9) will be bounded by (3.3), (3.9) and (3.10).

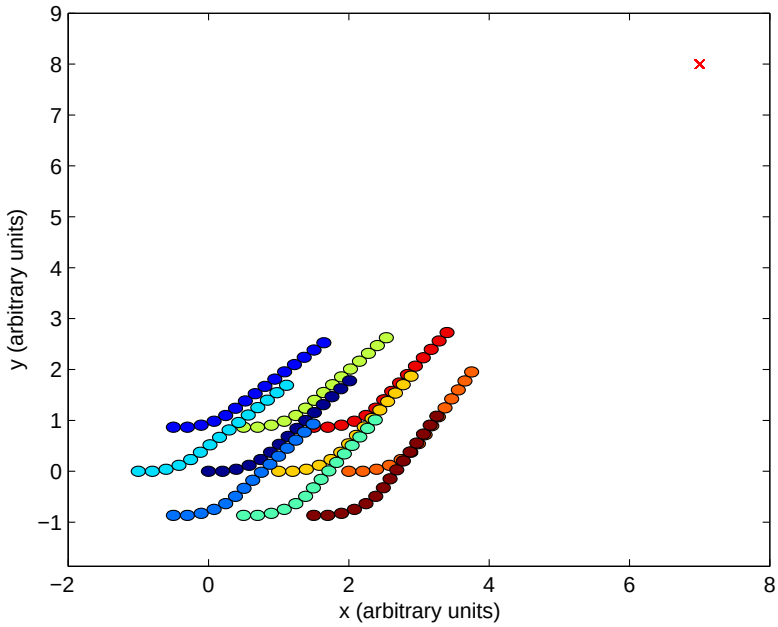

Figure 3.4: Middle of the guidance simulation

Chapter 4

Control strategy and stability analysis

The considered problem will be solved with three independent controllers. Indeed every robot compute its own trajectory with a **guidance controller**, that can guide the robot on its route avoiding the obstacle. The second controller called **Formation controller** will ensure that the robots will stay in a fixed formation. Once arrived in the nearness of an obstacle, we don't want to avoid it in formation, therefore will not need the formation controller. We will switch to a **rendez-vous algorithm** that will drive each agent, one by one, to a new group placed in the other side of the obstacle, in order to re-gather towards destination.

4.1 Guidance control

The planning problem is to compute the trajectory for every robot in every sampling time. We use the idea of [3] to divide the guidance coordination in two steps. First of all, every robot has simplified trajectory $\check{q}_i(k)$, for the next sampling time t_{k+1}, that considers only the constraints over the speeds(3.3), and over the distance from the obstacles (3.9). A cost function like (2.9) will then be applied. The associated command law is $\hat{u}_i(k)$, to

compute the trajectory using (3.8). As said if $\mathcal{O}_i(t_k)$ is empty for all the $0 \leq t_k \leq t_{max}$ all that the robot does, is to go by a straight line to the waypoint $\mathcal{W}_i$ defined by the coordinates as in (3.15). The guidance command $u_{g,i}(k+1) = [\nu_i(k+1)\omega_i(k+1)]$ will be computed in the next way (to simplify the notation here we write a generic variable $x(k)$ as x_k). First of all we try to reach waypoint at the max speed as possible. Only when we reach the waypoint we need to decrease the linear speed.

$$\begin{cases} \check{\nu_{k+1}} = \nu_{max} & \text{if } Y_{w_i} \geq y \\ \check{\nu_{k+1}} = -\nu_{max} & \text{if } Y_{w_i} < y \end{cases} \tag{4.1}$$

After that, we will compute the angular speed. As shown in picture 4.1 maybe our robot is oriented in a different way than desiderated. But we have to take count of the limited angular speed, that limits the maximum variation θ_{max} of rotation. Introducing $\theta_{w_i} \in [0; \pi[$ as the angle between the reference and the waypoint:

$$\begin{cases} \check{\theta_{k+1}} = \theta_k + \theta_{max} & \text{if } \theta_{w_i} > \theta_k + \theta_{max} \\ \check{\theta_{k+1}} = \theta_k - \theta_{max} & \text{if } \theta_{w_i} < \theta_k - \theta_{max} \\ \check{\theta_{k+1}} = \theta_{w_n} & \text{else} \end{cases} \tag{4.2}$$

Therefore, in absence of obstacles, the command law will be:

$$\begin{cases} \hat{\omega}_{k+1} = \check{\theta_{k+1}} - \theta_k \\ \hat{\nu}_{k+1} = \check{\nu_{k+1}} \end{cases} \tag{4.3}$$

In presence of the obstacles $\mathcal{O}_i(t)$ (indicating as $\theta_{i,o}$ the angle between the reference system and the obstacle $O_{i,o}$), the command law will be computed to minimize the distance of the path to avoid the obstacles. Using the Carnot's theorem (see Appendix B.4) we can find the $\alpha_{i,o}$ angle: with the notation of picture 4.2 α is the angle to compute, c is computed knowing the maximum linear speed ν_{max}, while a and b are bounded by (3.9). In this way, we have to choose between two different points marked as the two red Xs:

$$\begin{cases} \acute{\theta_{k+1}} = \theta_{i,o} - \alpha_{i,o} & \text{if } \theta_{i,o} \geq \check{\theta_{k+1}} \\ \acute{\theta_{k+1}} = \theta_{i,o} + \alpha_{i,o} & \text{else} \end{cases} \tag{4.4}$$

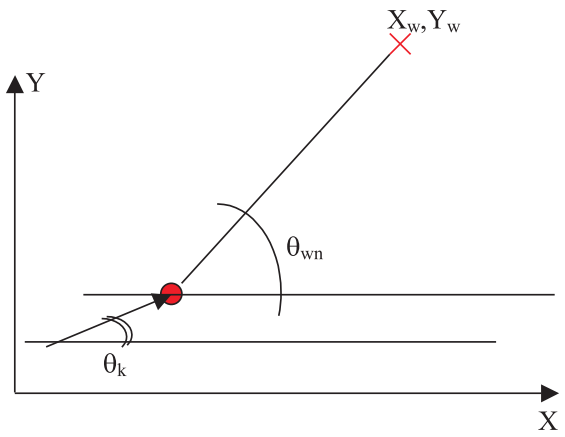

Figure 4.1: Orientation of the robots without obstacles

$\check{\theta_{k+1}}$ will be our new chosen orientation. The new arriving point can be, once again bounded by the maximum speed (in this case, angular speed) available for the robot. But now we cannot let the robot free to go as it is bearing a different angulation than desired and risk of crash with the obstacles becomes possible. Linear speed must be turned on zero till robots are back in the initial direction. With the same formulation of (4.1),(4.2) and (4.3) we will obtain

$$\begin{cases} \hat{\theta}_{k+1} = \theta_k + \theta_{max} \\ \hat{\nu}_{k+1} = 0 \end{cases} \qquad if\ \theta_{k+1}^{'} - \theta_k > \theta_{max} \tag{4.5}$$

$$\begin{cases} \hat{\theta}_{k+1} = \theta_k - \theta_{max} \\ \hat{\nu}_{k+1} = 0 \end{cases} \qquad if\ \theta_{k+1}^{'} - \theta_k < -(\theta_{max}) \tag{4.6}$$

$$\begin{cases} \hat{\theta}_{k+1} = \theta_{k+1}^{'} \\ \hat{\nu}_{k+1} = \nu_{max} \end{cases} \qquad else \tag{4.7}$$

Once the simplified trajectory $\hat{q}_n(k)$ is computed (as 3.4), each robot sends it to one another. Those trajectories will be seen as an obstacle by other robots, therefore we shall apply the same algorithm explicated above with a new set of obstacle $\hat{\mathcal{C}}_i(k)$ that is merely the sum of the true obstacles with

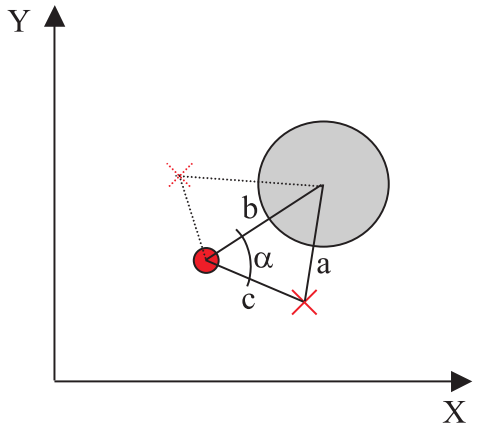

Figure 4.2: Computing of the angle for the obstacle's avoidance

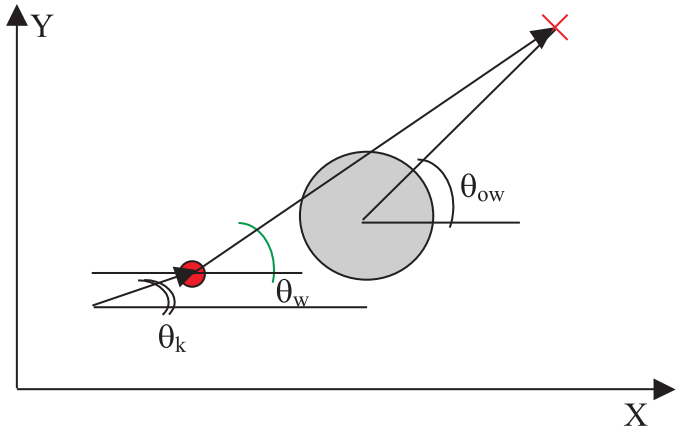

Figure 4.3: Orientation of the robots with obstacles

the simplified trajectories of the robots that can cause crash as in (3.11), obtaining in this way a new command and than the optimal planned trajectory $q*_i(k)$. This way, we have a guidance control that guarantees the reach of the waypoint with a full avoidance of any obstacle, and no crash between the robots. Of course we must be satisfied, that the request that the two trajectories will be computed in a shorter time than the sampling one. We also note that each robot doesn't know the destination point of the other robots, having then a full decentralized control.

4.2 Formation control using Voronoi's regions

Now we explained the technics based on Lindhé and Johansson's work [12] for the stability of a mobile-robots formation. The proposed solution is to fix the position of each agent inside the formation as the center of the Voronoi region computed. Afterwards we shall introduce the concept of Voronoi Regions.

Definition 4.2.1 (Voronoi Region) *Let $Z = \{z_1, \ldots, z_m\}$ be a set of points in $\Re^2$. The Voronoi region $R_i(Z) \subset \Re^2$ consist of all points that are closer to z_i than any other point in Z:*

$$R_i(Z) = \{x : |x - z_i| < |x - z_j| \forall j \neq i\} \tag{4.8}$$

The line segments

$$l_{li}(Z) = \{x : \exists l \neq i : |x - z_i| = |x - z_l| < |x - z_j| \forall j \neq i, l\} \tag{4.9}$$

represents the boundaries of $R_i(Z)$. The set of Voronoi vertices $V_i(Z)$ for z_i are the points:

$$V_i^{lj} = l_{l,i}(Z) \cap l_{j,i}(Z) \tag{4.10}$$

Definition 4.2.2 (Neighbor set) *It is finally easy to define the set of neighbors, $\mathcal{N}_i$ of the agent i as all agents that share a Voronoi vertex with it. We denote as $C_i \subset \mathcal{N}_i$ the close neighbors set, i.e. the set of all agents that share two or more vertex with the agent i.*

The algorithm of Lindhé and Johansson can stabilize a formation, with a collision safety and formation cohesion. The algorithm works fine with non-holonomic agents and is completely decentralized; moreover it is very light from the computational point of view and finally does not need a communication system. For all this reasons I choose this algorithm from the literature as my formation control. On the other hand this algorithm has some very strong requests:

- The formation's lattice must be exagonal then each agent has exactly 3,4 or 6 neighbors;
- at start all at interagent distance d;
- we must have minimum 7 robots in the formation;
- the perturbation over the process must be small enough (later we will give a dimension at this "small");

4.2.1 Algorithm description

The idea at the base of the algorithm is that each agent has to compute his Voronoi region inside the formation and then moves to the centroid of it. To be sure that also externals agents (agents with $\mathcal{N}_i < 6$) have a finite Voronoi region, we build imaginary neighbors though mirroring of real agent (then we call as "mirrors" this fakes agents):

$$M(p_x, p_y) = p_x - \frac{p_y - p_x}{||p_y - p_x||} d \tag{4.11}$$

where p_x represents the position of our considered agent and p_y the position of the agent to be mirrored, d their distance. Once computed the mirrors we compute the centroid of the vertex of the Voronoi region as:

$$c_i = \frac{\sum_{z \in V_i(\mathcal{N}_i)} z}{size(V_i(\mathcal{N}_i))} \tag{4.12}$$

then we apply the command:

$$u_{voro,i} = u_{f,i} = c_i - p_i \tag{4.13}$$

Having a limited linear speed bounded by (3.3) we must ensure that $|u| < \nu_{max}/t_s$ where t_s is the sampling time. We don't care about the angular speed that here we supposed infinite. Indeed in this algorithm we suppose no-dimensional agents. As formulated above, algorithm requires to be computed in the same time by all the agents, then we required a radio synchronization of the clock.

4.2.2 Proprieties of the algorithm

Safety

Being the Voronoi's regions normally defined as convex and being the centroid of them inside the region, the centroid as defined is a convex combination of the points of the regions. On the other hand the agents move from a point inside the regions to the centroid of it and we can then assume that there will be no interagent collisions. We have also to ensure that each robot in one iteration does not move farther than $\frac{1}{2}R_{max}$ where R_{max} indicates the distance covered by the sensor.

Stability

Having, as said, a convex region, around the robot described by $q*$, will be applied a convex function f that computes them with a radium maximum of d, where d indicates the distance between two robots, the Jacobian of our function:

$$J = \frac{df}{dq}|_{q=q*} \tag{4.14}$$

has two eigenvalues $\lambda_{1,2} = 1$ that correspond to the 2 degrees of freedom, and others eigenvalues all inside the unit circle that tell us that we can have and preserve the stability of the formation if the perturbations are $\eta < d$ [12].

Region of attraction

Having others controllers in parallel with the formation controller we want to know how much we can move our robot without breaking up of the formation. Simulations done by Lindhé and Johansson tell us that any perturbation to the original position that is inside of a circle of radius $d/\sqrt{3}$ will be rejected by the controller, otherwise the group may not converge to desired formation. Then we have to limit to this value the range of our guidance control.

4.3 Rendez-vous algorithm

If the formation of robots will find an obstacle we want to have an independent avoidance of the obstacle for each robot, Then we have to "turn off" the formation control. Our problem is that, as defined in previous paragraph, the formation control needs that the robots are in an hexagonal lattice, yet done, before to be turned on. We now need an algorithm that allows the robots to come back in a hexagonal lattice formation (obviously, always without a global control).

4.3.1 Proposed algorithm

The idea is to have, in each robot a memory of the form of the initial lattice formation, and the initial position of the robot in it. Using the simplified form (3.6) of the lattice at starting point (for $t = t_0$) we can compute the barycentre of it as:

$$CG(t_0) = \frac{\sum_{i=1}^{N} p_i(t_0)}{N} \tag{4.15}$$

The position of each robot can be then described as a relative position compared to the barycentre.

$$\hat{p}_i(t_0) = p_i(t_0) - CG(t_0) \tag{4.16}$$

We can also assume as radius of the formation the maximum value of $\hat{p}_i(t_0)$:

$$R_f(t_0) = (max||\hat{p}_i(t_0)||) \tag{4.17}$$

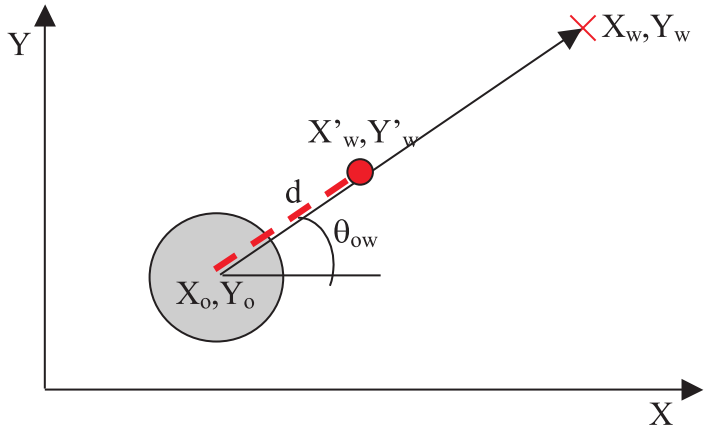

Figure 4.4: Computation of the new waypoint; $d = \varsigma(rf(t_0) + r_o)$

keeping in memory these two values, it is easy to think that our problem now is to compute the position of the barycentre of the formation CG' after the obstacle avoidance. The idea is to put it on the other side of the obstacle towards the waypoint (see picture 4.4). The computed waypoint will be:

$$\begin{cases} X'_W = X_o + \varsigma(rf(t_0) + r_o)cos(\theta_{o,w}) \\ Y'_W = Y_o + \varsigma(rf(t_0) + r_o)sin(\theta_{o,mw}) \end{cases} \tag{4.18}$$

Where:

- $\varsigma > 1$ is a real constant used as a safety factor correction;
- X_o and Y_o indicates the position of the obstacle $O_o \in \Omega$;
- r_o is the radius of the considered obstacle;
- $\theta_{o,w}$ is the angle between the reference system and the straight line that join O_o with the destination W_i of the agent R_i.

Finally we can use this waypoint as the new destination in the guidance control while all the robots are in position, forming the new formation ready to start once more to the final destination.

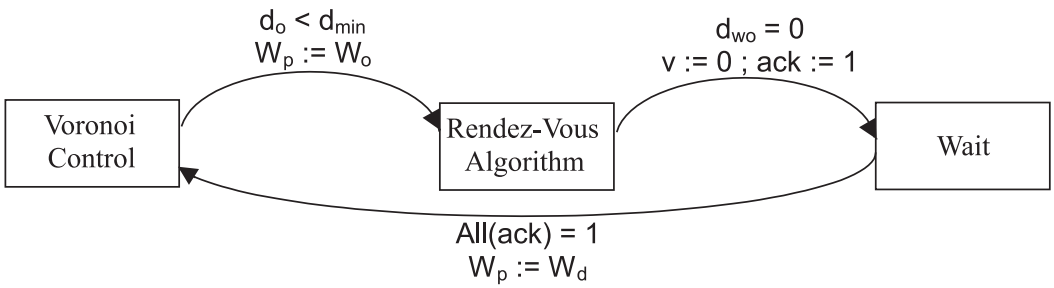

Figure 4.5: State machine

4.4 State machine

Now, as explained, we have three totally independent controller for the guidance of our robot to destination: a guidance controller, a formation stability controller (Voroni based) and a rendez-vous control. We need a high-level controller, in our case implemented as a state machine, to switch between the Voronoi control and the Rendez-vous control. The guidance control will be always active, but will switch the waypoint towards which it is heading. See figure 4.5. When the detector finds a close obstacle O_o so that $d(p_i(t), O_o(t)) \leq d_{max}$ we switch from the first state (controller of Voronoi) to the second one (rendez-vous). During this switching a new waypoint W_o will be computed, as (4.18).
Once the robot i arrives to the waypoint it sends an $ack(i) = 1$ and goes in the wait state, where it waits for the others. When:

$$ack(i) = 1 \quad \forall i \in 1 \dots N \tag{4.19}$$

the waypoint as input of the guidance controller will be restored to the destination point W, and the state will be switch back to the first state.

Controls - resuming table:			
State	Name	Active controls	Resuming formula
♯1	Normal guidance	Guidance; Voronoi	$q(k+1) = q(k) + u_g(k) + u_f(k) + \eta(k)$
♯2	Obstacle avoiding	Guidance	$q(k+1) = q(k) + u_g(k) + \eta(k)$
♯3	Waiting	-	$q(k+1) = q(k) + \eta(k)$

Chapter 5

Simulation results

Aiming to test our system, we realize a model with Matlab. We will test each part of the system independently and then the ensemble of the algorithm will be tested. Testing with and without perturbation will be done with the task to show the goodness of the algorithm and the stability of the system. For all the simulation we have choosen (when not otherwise stated) the same initial condition so it is easier to make a comparison between the picture attached. We simulate a formation of 10 robots (only in one case we will choose a small formation of 3 robots), with a dimension of 10 cm. The sampling time is 0.1 seconds ($f_s = 10Hz$), and we choose a time of 30 secs, then 300 sampling times, for the whole simulation. The robots have a maximum linear speed ν_{max} of 4 m/s, a speed slow enough to permit the real time computation of the trajectory. The angular speed is also bounded by a max value of $\omega_{max} = 2rad/s$. The initial orientation of the robots relevant to X is $\theta = 0$: horizontally in the pictures. At the end of each simulation it will be gave a resuming table useful to reproduce the test.

5.1 Guidance tests

First of all, we simulate the guidance algorithm without noise and with no obstacles. We do the test on a 10-robots formation, in a hexagonal lattice as

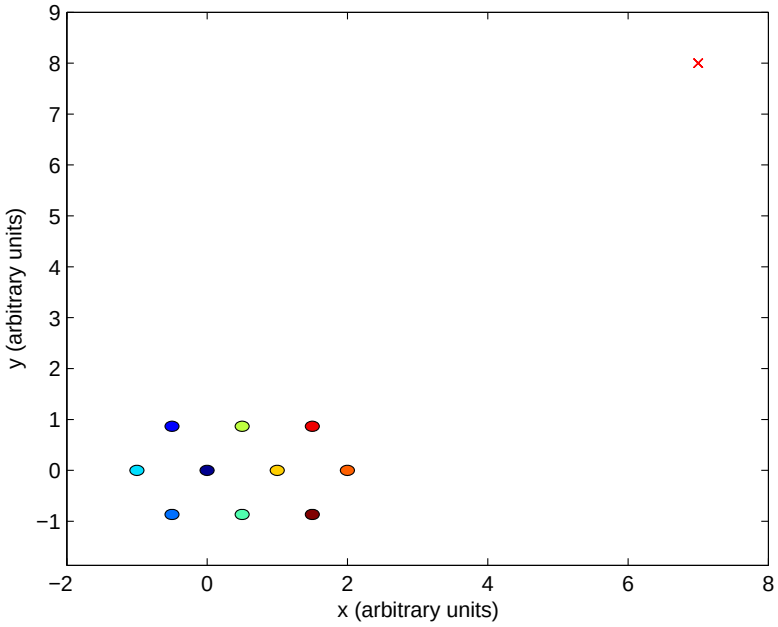

Figure 5.1: Begin of the simulation

shown in picture 5.1 (but any initial placement of the robots can be choose for this test). In figure 5.2 we can see how the robots move over the horizon. It is easy to note that the choice of initial orientation of the robots with $\theta = 0$ (the direction of the X axis on the picture) combined with the limited angular speed draws a curve trajectory. In picture 5.3 it is showed how, even in absence of a controller for the stability of the formation, we can have all robots converging in the same destination point and reaching the waypoint in a finite time as shown in figure 5.4. The pictures 5.5 and 5.6 show that the minimum distance from each robots to the others starts, as said, with a value of 1 meter, but never goes below the imposed distance of 30 cm between the centers of the two circular robots. Indeed the robots will arrive at destination without crash between them. Discontinuities, as you can easily observe on the picture, after about 75 and 110 sampling time, will occur when a robot will reaches its destination point.

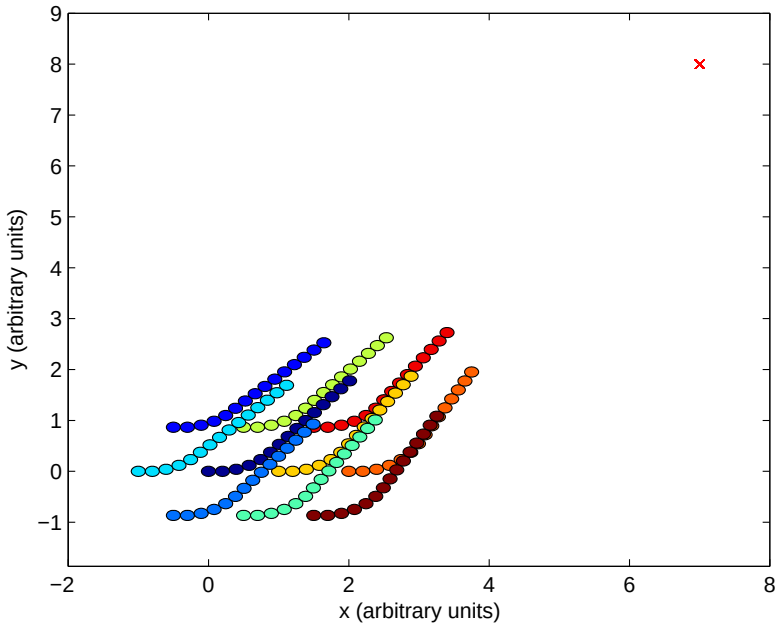

Figure 5.2: Middle of the guidance simulation

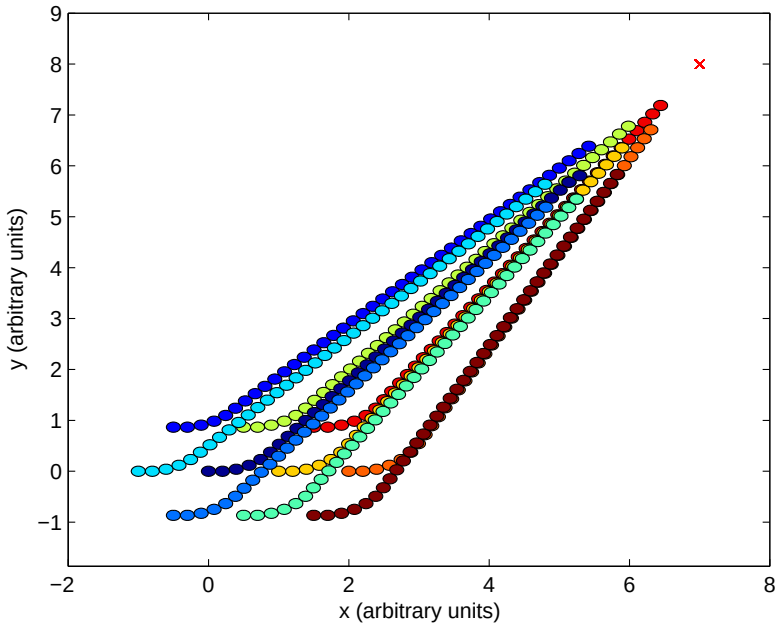

Figure 5.3: Trajectories of the robots

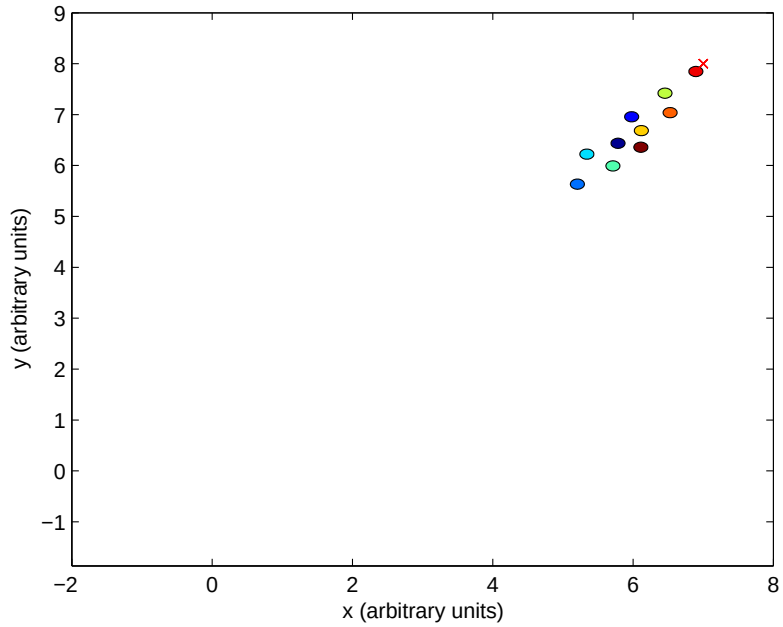

Figure 5.4: Arrive of the robots

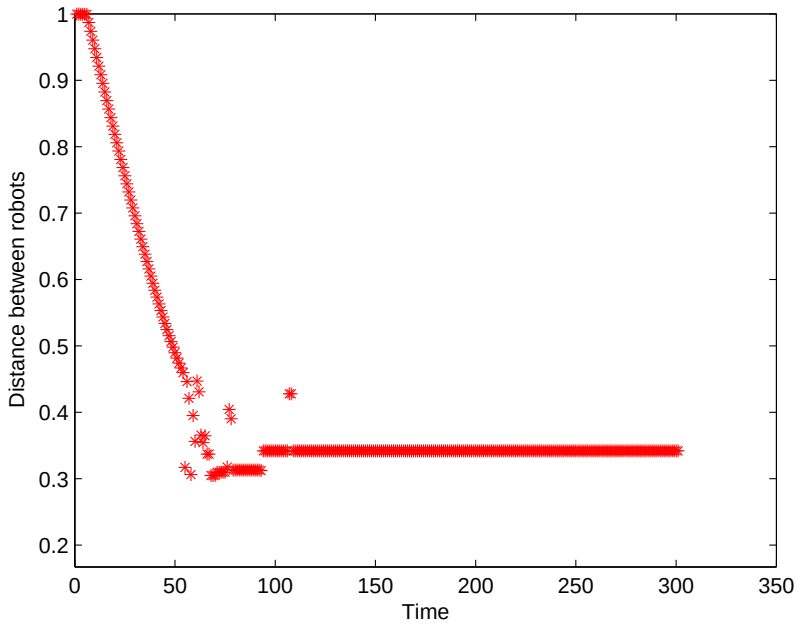

Figure 5.5: Minimum distance between the robots (large view)

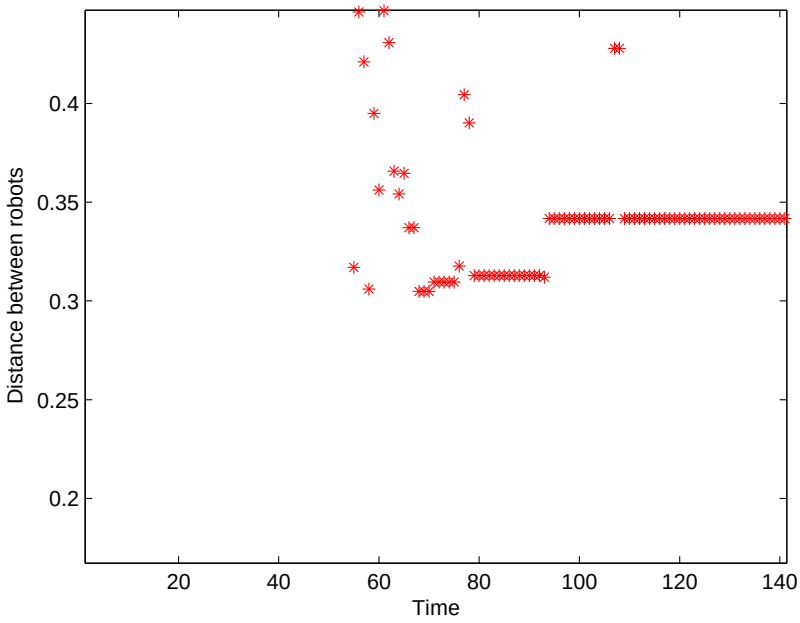

Figure 5.6: Minimum distance between the robots (close view)

Guidance control test - resuming table:	
number of agents	10
number of obstacles	0
radius of each agent	10 cm
sampling time	0.1 s
maximum linear speed	4 m/s
maximum angular speed	2 rad/s
position of waypoint	[7,8]
$\theta(t_0)$	0
noise maximum value	0

5.2 Obstacle avoidance test

We introduce now, in the model, some obstacles on the route of our robots. In pictures 5.7, 5.8 and 5.9 we can see the evolution of the system once the

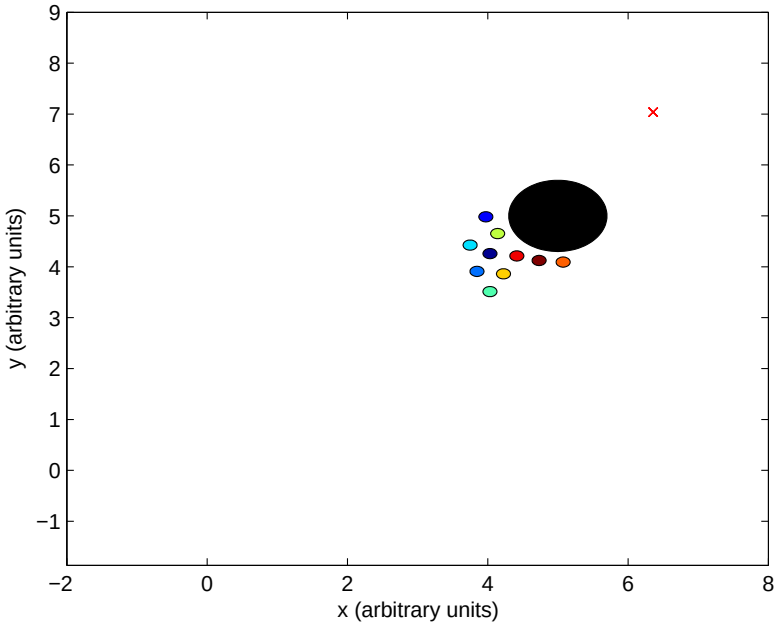

Figure 5.7: Obstacle avoidance - 1

robots find the obstacle. In picture 5.10 it is shown another application with a model consisting of only 3 robots and three obstacles. We can note that the algorithm works fine as far as avoidance of the obstacles since each robot safety keeps at a security distance from them. It is also interesting to note that, once the robot has reached the obstacle, it chooses the trajectory to follow and, bounded by his angular speed, waits to be oriented in the correct direction before continuing. This fact it can be observed when the blue robot reach the second obstacle in picture 5.10.

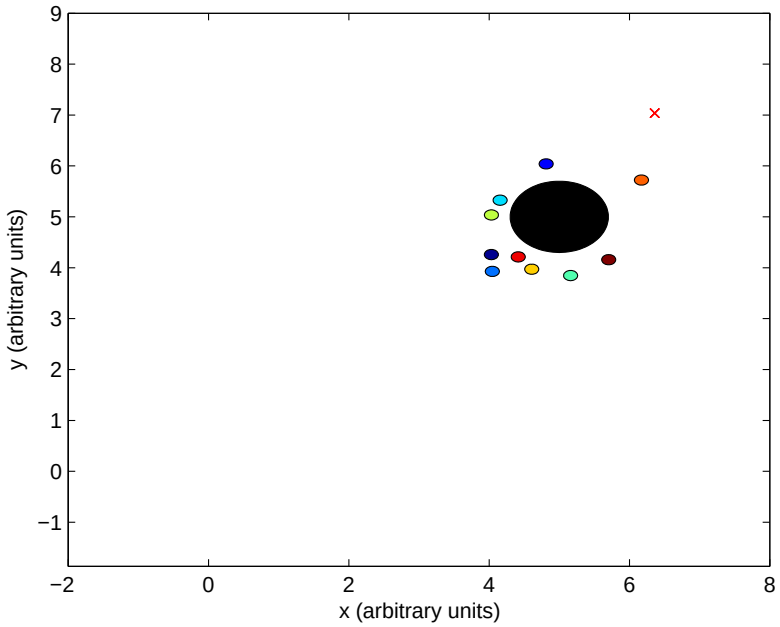

Figure 5.8: Obstacle avoidance - 2

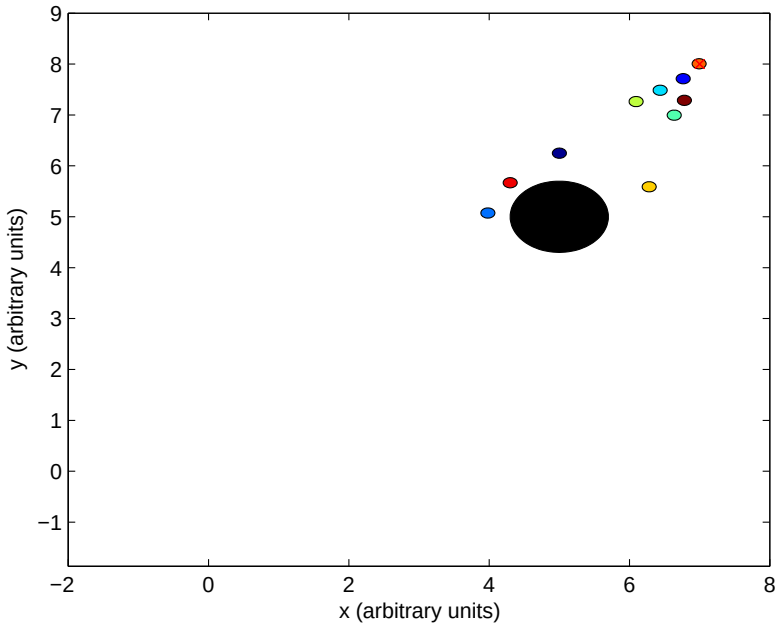

Figure 5.9: Obstacle avoidance - 3

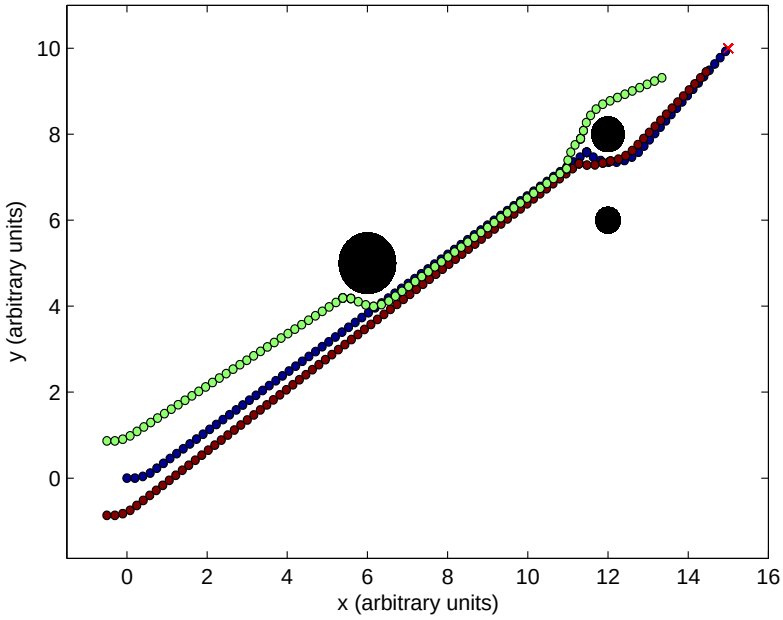

Figure 5.10: Obstacle avoidance of a system of 3 robots in a Horizon with 3 obstacles

Obstacles avoidance test ♯1 - resuming table:	
number of agents	10
number of obstacles	0
radius of each agent	10 cm
radius of the obstacle	70 cm
sampling time	0.1 s
maximum linear speed	4 m/s
maximum angular speed	2 rad/s
$\theta(t_0)$	0
position of waypoint	[7,8]
noise maximum value	0

Obstacles avoidance test ♯2 - resuming table:	
number of agents	3
radius of each agent	10 cm
number of obstacles	3
radius of the obstacle 1	70 cm
position of obstacle 1	[15,10]
radius of the obstacle 2	40 cm
position of obstacle 2	[6,5]
radius of the obstacle 3	30 cm
position of obstacle 3	[12,6]
sampling time	0.1 s
maximum linear speed	4 m/s
maximum angular speed	2 rad/s
$\theta(t_0)$	0
position of waypoint	[15,10]
noise maximum value	0

5.3 Formation stability test

The goal of the formation controller (Voronoi controller) is to stabilize the formation against external perturbations. Perturbation can represent incertitude of measures of the sensor and a rough terrain in which the robot moves. We shall then apply an external perturbation model by means a random matrix to be added to the control. The maximum value of this random noise is 5 centimeters in both directions (for each sampling time). In picture 5.11 and 5.12 we can observe that the shape of the lattice formation does not change in time as we wish: all we have is a translational movement of the formation as a group in a (random, in reality) direction. We should not be worried by this fact as this movement is not a problem of the formation controller. Picture 5.12 shows longs regions of voronoi in the external of the lattice, derived by a computational error of the software that considers those regions not infinite like the others outside our formation. Indeed the

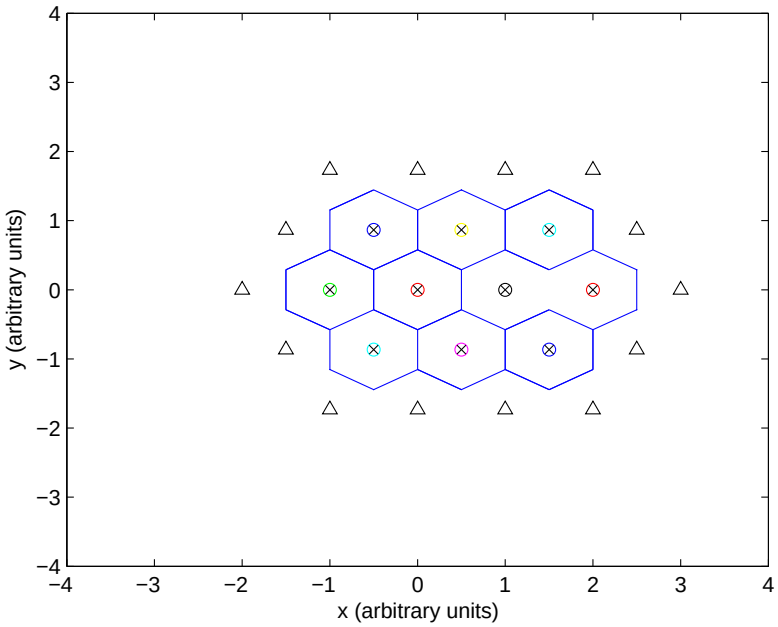

Figure 5.11: Starting voronoi regions

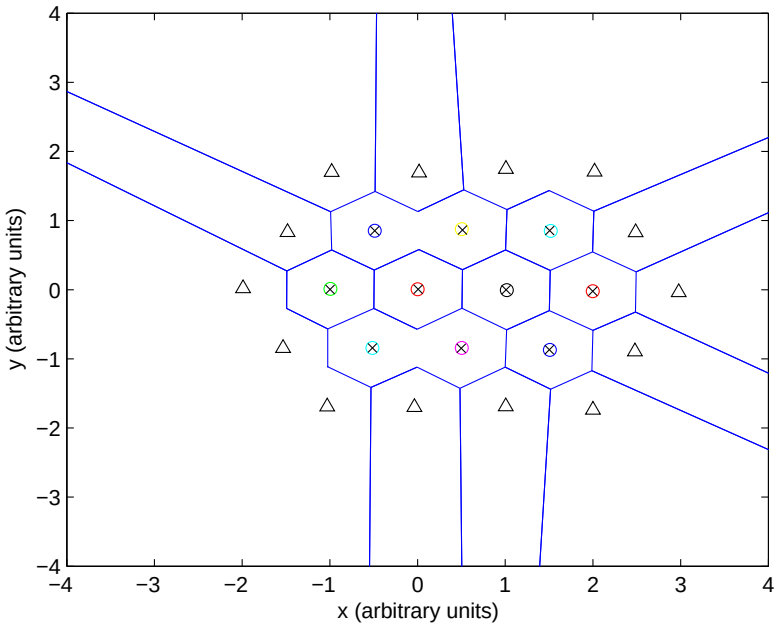

Figure 5.12: Voronoi regions of the formation after 10 seconds

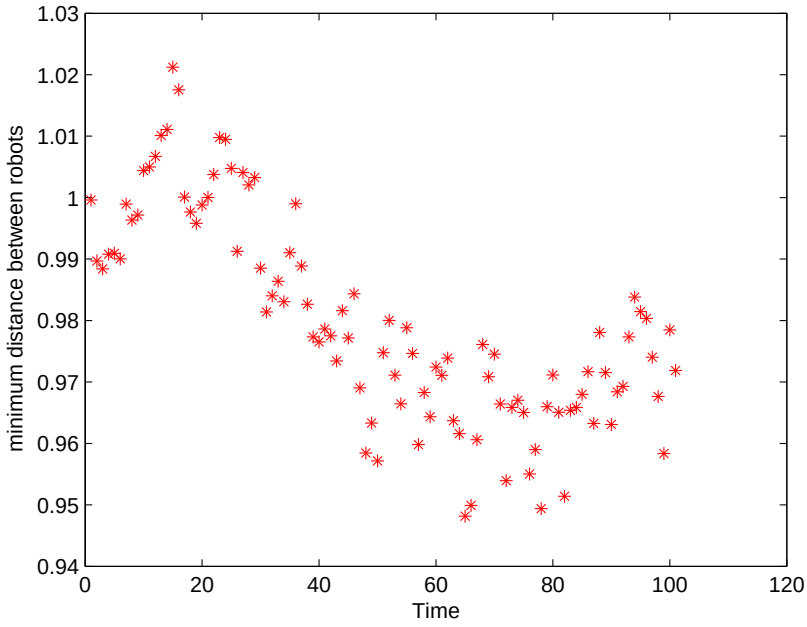

Figure 5.13: Distance between the robots applying over the formation a random noise and the voronoi controller

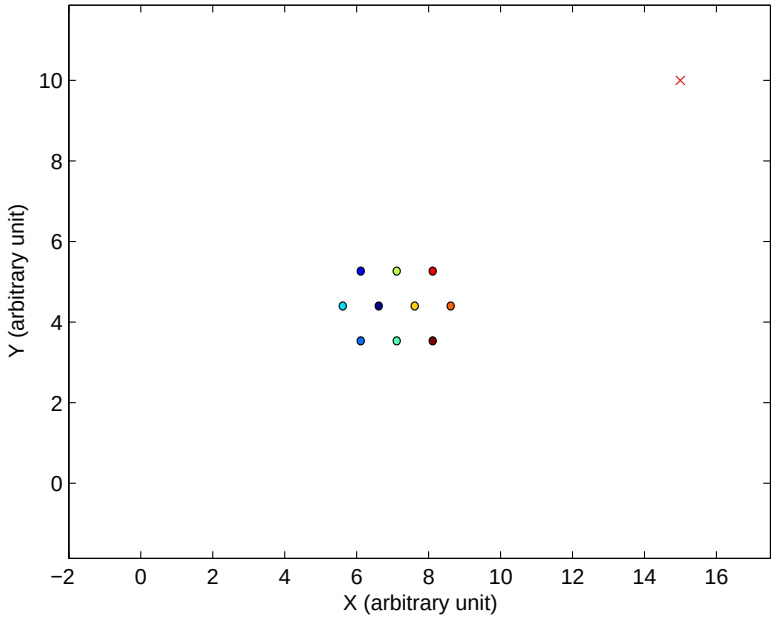

Figure 5.14: Simulation of the system with formation and guidance control, both applied - after half travel

software computes also the "mirrors", explained in chapter 3, represented in the pictures by a black triangle. Thus we ensure a closed and finite region of Voronoi for all the "true" agents. Finally in picture 5.13 it is easy to note that the distance between two robots will maintain in time, more or less the ideal distance of 1 meter. In pictures 5.14 and 5.15 we can see how, it is possible to reach the waypoint in a lattice formation as desiderate with the combination of the two controllers.

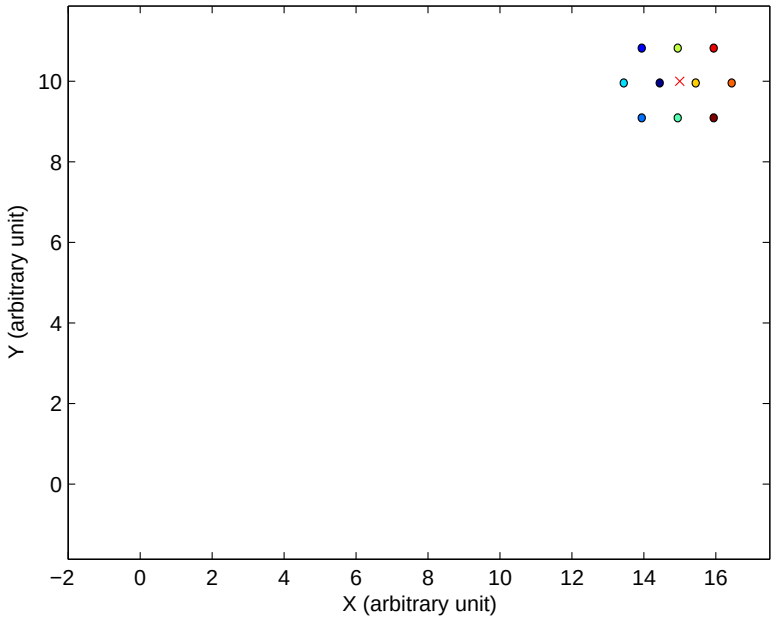

Figure 5.15: Simulation of the system with formation and guidance control, both applied - Destination reached

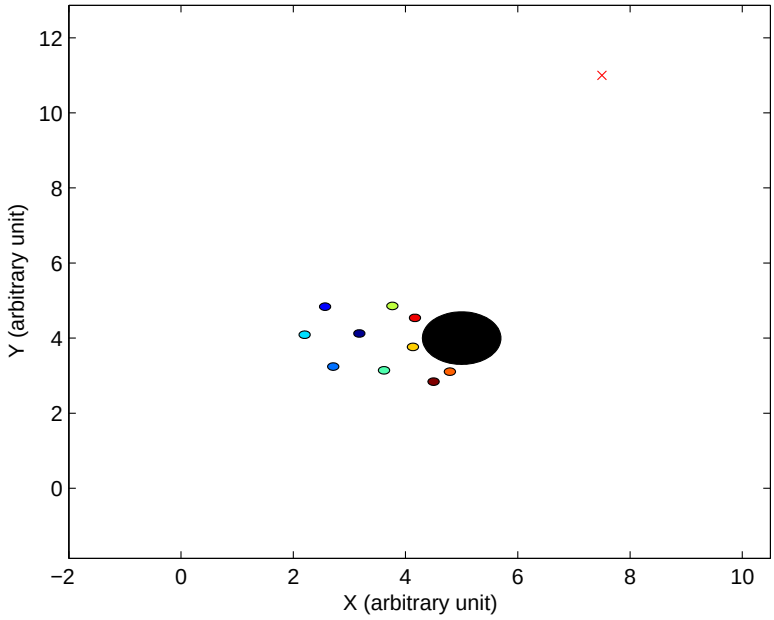

Figure 5.16: Obstacle with formation controller

Formation stability test - resuming table:	
number of agents	10
radius of each agent	10 cm
number of obstacles	0
sampling time	0.1 s
maximum linear speed	4 m/s
maximum angular speed	∞
$\theta(t_0)$	0
noise maximum value	0.5 m/s

5.4 Rendez vous controller test

If we place an obstacle in the model, path of the formation, with no change of state as explained in chapter 3, the situation will become as shown in pictures 5.16, 5.17 and 5.18. We can observe that once that the firsts robots

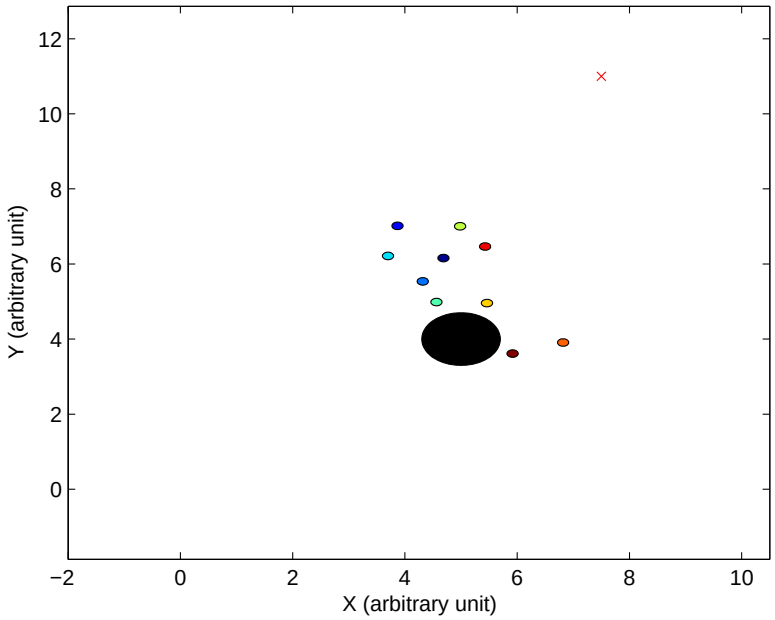

Figure 5.17: Obstacle with formation controller

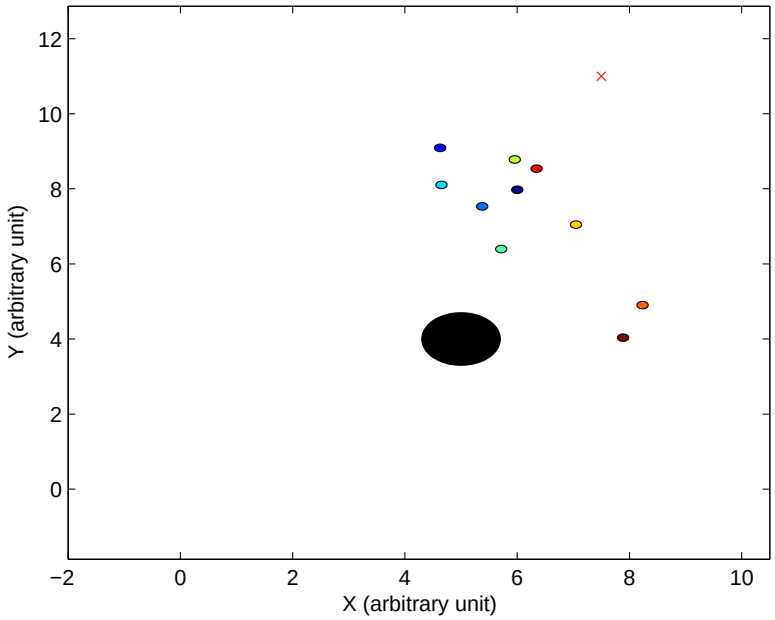

Figure 5.18: Obstacle with formation controller

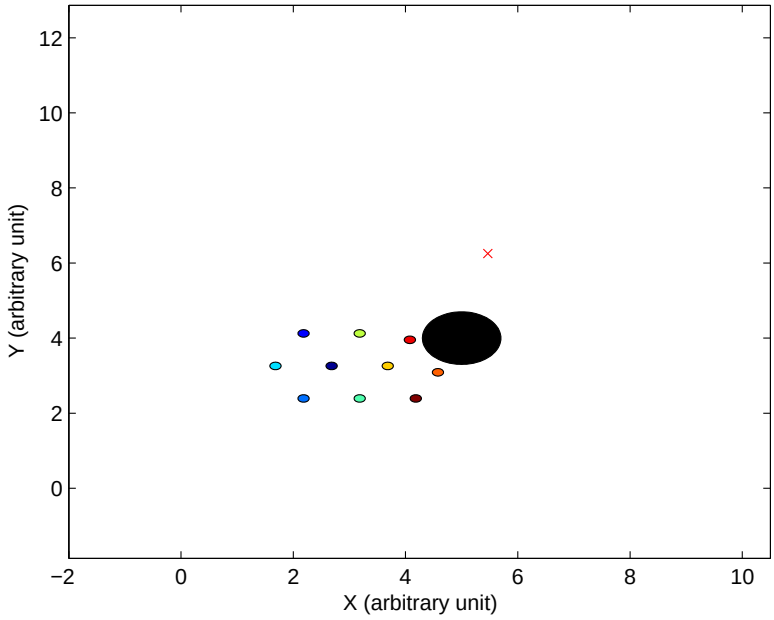

Figure 5.19: Rendez-Vous controller - 1st step: it be build a new waypoint for the formation

find an obstacle, the others go their way, breaking thus hexagonal lattice and jeopardizing the mission of Voronoi's controller. In a better case the Voronoi's controller will compute a bad command on the base of bad regions (as in case of picture 5.18). In the worst case we will have a computational overflow and a block of the system. At this point it is necessary show the work of the last one controller: the rendez-vous controller, that has the task to re-build the formation after avoidance of the obstacle by this lattice. When the first robot finds an obstacle (figure 5.19) it will send via radio the command to others to change the state of the state machine, after which each robot will build the new waypoint, as explained in chapter 3. The new waypoint will be reached without a formation stabilization control as showed in pictures 5.20 and 5.21. After it the formation controlled will be turned on once again and the lattice can drive ensemble till destination. As we can see in pictures 5.22 and 5.23 we have a problem when we pass from the third state to the

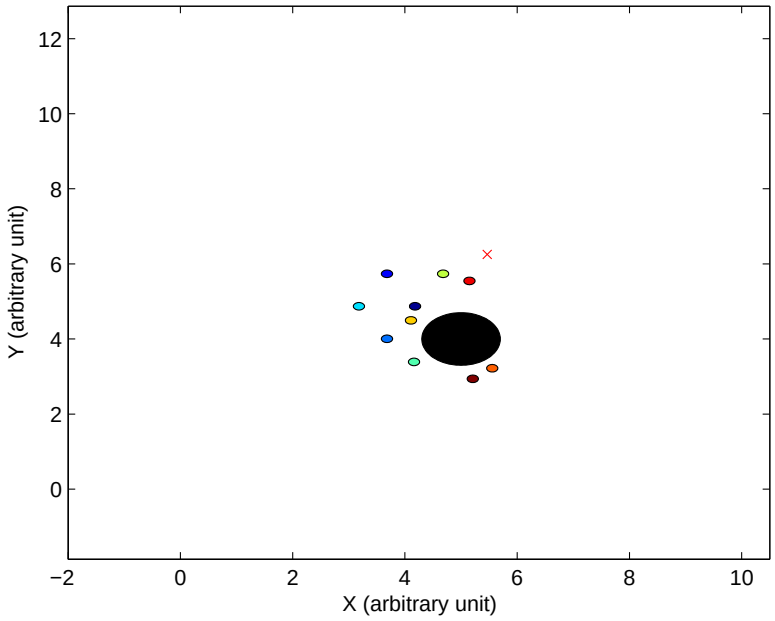

Figure 5.20: Rendez-Vous controller - 2th step: every robot guide himself to the new waypoint

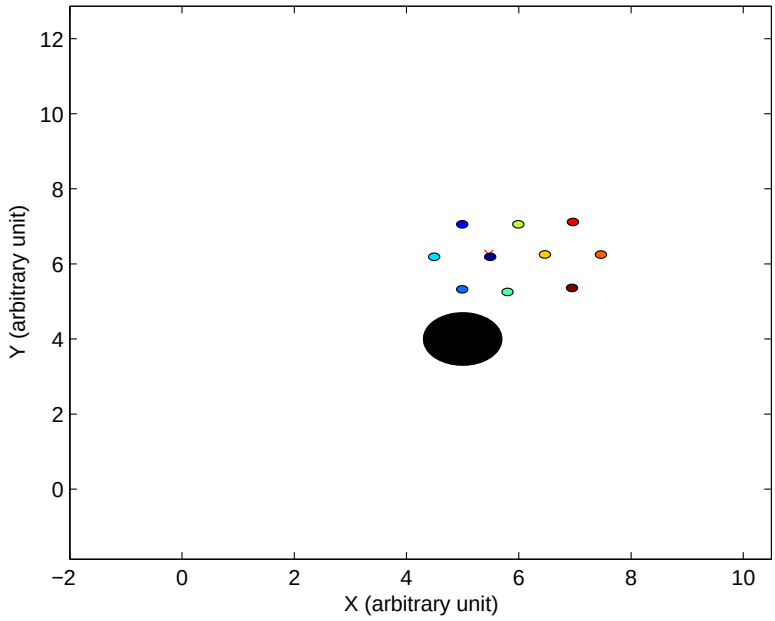

Figure 5.21: Rendez-Vous controller - 3th step: the formation it is been recomposed

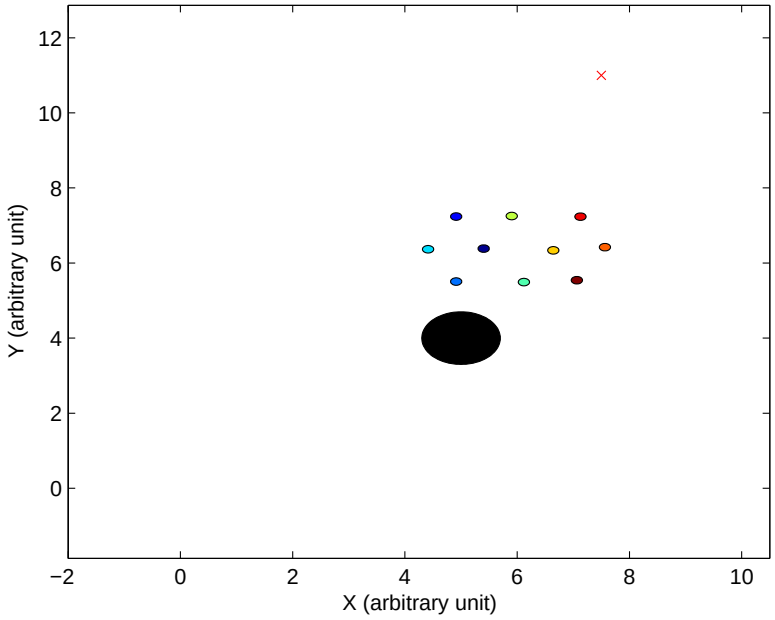

Figure 5.22: State machine - instability on the switch

first one of the state machine. Indeed when we reach the new waypoint on the other side of the obstacle each robot waits for the others in his position. There, all commands are de-activated but the noise is still acting over our robot, so when all other agents will arrive the formation will not be exactly hexagonal. This problem can be solved introducing another kind of controller when the Voronoi controller is not active.

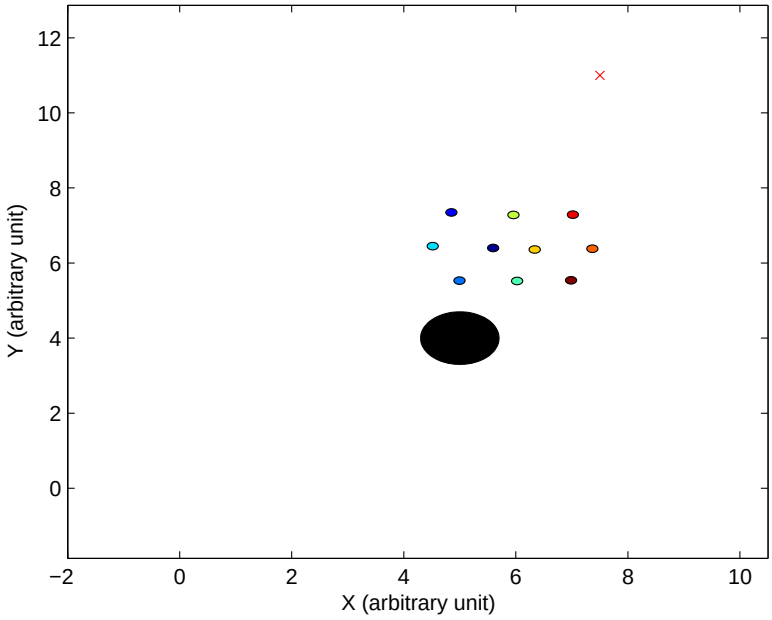

Figure 5.23: State machine - instability on the switch

Rendez-Vous controller test - resuming table:	
number of agents	10
radius of each agent	10 cm
number of obstacles	1
radius of the obstacle	0.7
position of the obstacle	[5,4]
sampling time	0.1 s
maximum linear speed	4 m/s
maximum angular speed	∞
$\theta(t_0)$	0
position of the waypoint	[8,11]
noise maximum value	0.5 m/s

Chapter 6

Conclusions

In this thesis we have take two papers and relative ideas, joined them to solve a new problem. With the help of some literature we have introduced an hybrid system based on a state machine to join the three developed controllers. It will be also developed a Matlab software with the goal to test the goodness of our theory. Hence this thesis completes the previous works analyzed, giving to the scientific community a new approach to solve the formation problem in robotics but also in other fields of study. Indeed as yet said, the work is very specific for a single application, but the theory of the agents it can be applied to many different areas. It is interesting to see how you can be optimize it by using a n-dimensional space, or just 3-D for flying robots. It is possible to improve my work with in other ways, overcoming the limits of this work. First of all, we have used the work of Lindhe-Johansson as idea for the formation control: I also chose a hexagonal lattice, as the initial formation of the group of robots. It is a strong limitation and it would be interesting to find an algorithm that uses the Voronoi's control with a lattice of every convex form. As said we have a great limit over the speed and the intra-robot distance to ensure the stability and the safeness. It is possible to solve this problem using a controller based on a flocking potential (or also a speed consensus) that ensure the stability also for a great disturbance.

Appendix A

Matlab script

Below the Matlab script done for the simulation of the proposed algorithm. The first script is the *main* file: after giving the initial condition there is a *for* cycle that represent the time (every cycle is a sampling time). The control laws will be computed at each sampling time and will be done an high-level control to the choice of the state.

A.1 Main script

```
% Andrea Baccara 2008
% Main file: modeling of a system of N agents in a horizon with obstacles
% Agents must travel in a lattice formation and arrive to destination in
% the same lattice form. Thei have to avoid the obstacles all by themselfs.

%%%%%%%% initial conditions %%%%%%%%

clear all
close all
clc
%ROBOTS:
%-----------------------------------------
% initial position
```

```
P(:,1) = [0 0]';
P(:,2) = [-0.5 0.866]';
P(:,3) = [-0.5 -0.866]';
P(:,4) = [-1 0]';
P(:,5) = [0.5 -0.866]';
P(:,6) = [0.5 0.866]';
P(:,7) = [1 0]';
P(:,8) = [2 0]';
P(:,9) = [1.5 0.866]';
P(:,10) = [1.5 -0.866]';

% radius of each agent
ro = 0.1;

% teta: initial orientation of the agent
% 0<=teta<pi
teta(1:size(P,2)) = 0;

% initial condition of the state machine
state(1:size(P,2)) = 0;

% max linear speed
Vmax = 4;
% max angular speed
Wmax = 2;
% sensor range:
Rmax = 1;

%DESTINATION:
%-----------------------------------------------------------
W = [11 15]';

%OBSTACLES:
```

```
%-------------------------------------------------------------------------
%O: centroid of the obstacles
% rO: radium of each obstacle
O(:,1) = [5 3.5]';
rO(1) = 0.7;
O(:,2) = [9 12]';
rO(2) = 0.4;
O(:,3) = [6 8.5]';
rO(3) = 0.3;

% NOISE:
%-------------------------------------------------------------------------
varns = 0.05;

% TIME:
%-------------------------------------------------------------------------
ck=0.1;
t = 0:ck:30;
maxT = size(t,2);

%%%%%%%% Computing %%%%%%%%

%maximum distance that each robot can run in a sampling time:
dismax = Vmax*ck;
if dismax < Rmax + 1/sqrt(3)
    dmax = dismax-Rmax;
else dmax = 1/sqrt(3);
end

%maximum rotation of the robot in a sampling time
tetamax = Wmax*ck;

% centroid of the group
```

```
baricentro(1,1) = sum(P(1,:))/size(P,2);
baricentro(2,1) = sum(P(2,:))/size(P,2);

% radius of the group
for (i=1:size(P,2))
    bd = P(:,i) - baricentro;
end
radgr = max(max(bd)) + ro/2;

% waypoint for each robot
    for (p = 1:size(P,2))
        Wyp(p) = W;
    end

    % waypoint in the form of initial formation lattice
    for (i=1:size(P,2))
        Wyp(:,i) = W - baricentro + P(:,i);
    end

    %Plotting:
    %-------------------------------------------------
    %plot the waypoint as a red X
    foraxis = [W O P];
    aa = 0:pi/50:2*pi;
    plot (W(1,1),W(2,1),'xr');
    hold on

    %%%%%%%% Simulation %%%%%%%%

    for (time = 1:maxT)

        %plotting:
        %-------------------------------------------------
```

```
hold off
% plot of the waypoint as a red X
plot (W(1,1),W(2,1),'xr');
hold on
% plot of the agents
for (i=1:size(P,2))
    fill (P(1,i)+ro*sin(aa),P(2,i)+ro*cos(aa),i);
end
% plot of the obstacles
for (i=1:size(O,2))
    fill (O(1,i)+rO(i)*sin(aa),O(2,i)+rO(i)*cos(aa),'k');
end
% add axis labels
xlabel('x (arbitrary units)');
ylabel('y (arbitrary units)');
axis ([min(foraxis(1,:))-1 max(foraxis(1,:))+1 ...
min(foraxis(2,:))-1 max(foraxis(2,:))+1])

%plot of mirrors
%plot (M(1,N+1:length(M)),M(2,N+1:length(M)),'^k');

% state machine
%-----------------------------------------------
% state0 -> open space: I want to stay in a formation group
% state1 -> (not a true state)
% I got an obstacle, compute the new waypoint
% state2 -> After an obstacle I want to come back in a formation

for (p = 1:size(P,2))
    for (o=1:size(O,2))
        disO(o) = distance (O(:,o)',P(:,p)');
    end
    disW(p) = distance (Wyp(:,p)',P(:,p)');
```

```
    if any (disO < (3*ro + rO)) & state == 0
        state = 1;
        break
    end
end

if state == 1
    [Wyp] = newWyp(O,rO,disO,Wyp,W,radgr);
    state = 2;
    dmax = dismax;
end

if all (disW < ro)
    if state == 2
        state = 0;
        for (p = 1:size(P,2))
            Wyp(p) = W;
        end
        if dismax < Rmax + 1/sqrt(3)
            dmax = dismax-Rmax;
        else dmax = 1/sqrt(3);
        end
    elseif state == 0;
        break
    end
end


%here it be computed the random noise.
%-----------------------------------------
% Vn = (-1)^time*5*varns*rand(size(P,1),size(P,2));
% Wn = (-1)^time*10*varns*rand(size(P,1),size(P,2));
% noise = rumore(size(P,2),Vn,Wn);
```

```
    noise = (-1)^time*varns*rand(size(P,1),size(P,2));


    % Control laws:
    %-----------------------------------------------------------------

    % guidance control
    [P2,teta,Pn] = main_obst(P,ro,teta,O,rO,Wyp,dmax,tetamax,ck);
    u2 = P2 - P;

    % Formation Control
    if state == 0
        u1 = main_voro(P,Rmax/2);
    else u1 = 0*u1;
    end

    % it apply the control law over the process.
    P = P + u2 + u1 + noise;
end
```

A.2 Guidance control - high level

```
function [P,teta,Pn] = main_obst(P,ro,teta,O,rO,W,dismax,tetamax,ck)

% Andrea Baccara 2008
% Guide of a group of agent over an horizon with obstacles
% Here we consider a system without perturbations.

% P: position of the agents
% ro: radium of each agent
% teta: initial orientation of the agent
```

```
% O: position of the obstacles
% rO: radium of each obstacle
% dismax: max distance bounded by linear speed
% tetamax: max rotation bounded by angular speed
% ck: sampling time
%%%%%%%


%calcolo la traiettoria semplificata
for (p=1:size(P,2))
    [Pn(:,p),tetan(1,p)] = ostacoli (P(:,p),teta(1,p),W(:,p),O,rO,ro,dismax,tetamax);
end

% I build On (with relatives rOn) thei are the sum of O and P.
% Actually I consider the simplyfied trajectory of other agents like an
% obstacle:

rOn = rO;
for (p=1:size(P,2)-1)
    rOn = [rOn ro];
end

for (p=1:size(P,2))
    if p == 1
    On = [O P(:,2:size(P,2))];
else On = [O P(:,1:p-1) P(:,p+1:size(P,2))];
end

% computing of the optimal trajectory
[P(:,p),teta(1,p)] = ostacoli (P(:,p),teta(1,p),W(:,p),On,rOn,ro,dismax,tetamax);
end
```

A.3 Guidance control - low level

```
function [P,teta] = ostacoli (P,teta,W,O,rO,ro,dismax,tetamax)

% Andrea Baccara 2008
% this function compute for one robot the trajectory knowing the position
    % of the obstacle and the waypoint.

    % disW: distance between robot and waypoint
    disW = distance(P',W');

    % it compute the speed
    if disW > dismax
        Vel = dismax;
    else Vel = disW;
    end

    %it compute teta
    % m is the angle between the robot and the waypoint
    m = atan((W(2,1)-P(2,1))/(W(1,1)-P(1,1)));
    if m<0
        m = m + pi;
    elseif m > pi/2 - 0.02
        m = 0;
    end

    %it compute the trajectory without obstacle consideration
    [Pn,tetan] = nobst (P,W,m,teta,tetamax,Vel);

    % it compute the angular coeff. between robot and obstacle
    for (i=1:size(O,2))
        if (O(1,i)-P(1)) > 0.01
            mO(i) = (O(2,i)-P(2))/(O(1,i)-P(1));
        elseif (O(1,i)-P(1)) < -0.01
```

```
            mO(i) = (O(2,i)-P(2))/(O(1,i)-P(1));
        else mO(i)= pi/2;
        end
    end

    % it compute the distance of the point preview with the Obstacle
    for (i=1:size(O,2))
        disO(i) = distance (O(:,i)',Pn');
    end

    if all (disO > (2*ro + rO))
        P=Pn;
        teta=tetan;
    else
        for (i=1:size(O,2))
            if (disO(i) <= (2*ro + rO(i)))
                [P,teta] = obst (P,rO(i)+2*ro,disO(i),Vel,mO(i),m,teta,tetan,tetamax);
                break
            end
        end
    end
```

A.4 Guidance controller in absence of obstacles

```
function [Pn,tetan] = nobst (P,W,m,teta,tetamax,Vel)
% Andrea Baccara 2008
% this function compute the trajectory command for the robots in absence of
    % obstacles.

    %it compute the new orientation tetan
    if (m - teta) > tetamax
```

```
        tetan = teta + tetamax;
    elseif (m - teta) < -(tetamax)
        tetan = teta - tetamax;
    else tetan = m;
    end

    % it compute the new arriving point Pn.

    if (P(2,1)-W(2,1)) < -0.1
        Pn(1,1) = P(1,1) + (cos(teta)*Vel);
        Pn(2,1) = P(2,1) + (sin(teta)*Vel);
    elseif (P(2,1)-W(2,1)) > 0.1
        Pn(1,1) = P(1,1) - (cos(teta)*Vel);
        Pn(2,1) = P(2,1) - (sin(teta)*Vel);
    elseif (P(1,1)-W(1,1)) < -0.1 | (P(1,1)-W(1,1)) > 0.1
        Pn(1,1) = P(1,1) + (cos(teta)*Vel);
        Pn(2,1) = P(2,1);

    else Pn=P;
    end
```

A.5 Obstacle avoidance

```
function [C,teta] = obst (A,a,b,c,mO,m,teta,tetan,tetamax)
% Andrea Baccara 2008
% It compute the trajectory command for the robots in presence of
    % obstacles.

    % A: initial position of the robot
    % C: command.
    % a: final distance between robot and obstacle
```

```
    % b: initial distance between robot and obstacle
    % c: a-b
    % mO: angle between robot and obstacle
    % m: angle between robot and waypoint
    % teta: orientation of the robot
    % tetan: new orientation computed by the function 'nobst'


% alfa angle between b and c
alfa = acos((c^2-a^2+b^2)/(2*b*c));
atm = atan(mO);

if atan(mO) >= tetan
    tetan = atan(mO) - alfa;
else tetan = atan(mO) + alfa;
end

if (tetan - teta) > tetamax
    teta = teta + tetamax;
    C=A;
elseif (tetan - teta) < -(tetamax)
    teta = teta - tetamax;
    C=A;
else teta = tetan;
    C(1,:) = A(1,:) + (cos(teta)*c);
    C(2,:) = A(2,:) + (sin(teta)*c);
end
```

A.6 Formation controller

```
function [u] = main_voro(P,dismax)

% Andrea Baccara 2007
% Here we stabilize the formation using a controller that compute the
% region of Voronoi for each robot inside the formation and guide it in
    % the centroid of its region.

    % P: position of the agents

    % M gives the position of all the agents + all the mirrors
    M = [P(1,1:size(P,2));P(2,1:size(P,2))];
    [V,C] = voronoin (M');

    % compute of the mirrors
    [M] = mirror(C,P);

    % computing Voronoi regions (and eventually patch it on the figure)
    [V,C] = voronoin (M');
    % for k = 1:length(C)
        % if all(C{k} =1) % If at least one of the indices is 1,
            % % then it is an open region and we can't
            % % patch that.
            % patch(V(C{k},1),V(C{k},2),k); % use color k.
        end
    end

    C1 = C(1:size(P,2)); % I only consider the true agents

    %it compute the control law:
    [u,B] = vorocontrol (V,C1,P,dismax);

    % plot of the centroids
```

```
    for (b=1:size(B,2))
        plot (B(1,b),B(2,b),'xk');
    end

    % it plot the edges of the regions
    for k=1:length(C)
        plot (V(C{k}),V(C{k}+length(V)));
    end
```

A.7 Voronoi controller

```
function [d,B] = vorocontrol (V,C,P,dismax)
% Andrea Baccara 2008
% this function take as input the position of the agents and the relatives
% Voronoi's regions, compute the centroid of each region and move the agent
% in the direction of it.

N = length (P);
for (i = 1:N)
    B(1,i) = sum(V(C{i}))*1 / (length(C{i})*1);
    B(2,i) = sum(V(C{i}+length(V)))*1/ (length(C{i})*1);
    if B(:,i)  = inf
        d(:,i) = B(:,i) - P(:,i);
    else d(:,i) = 0;
    end

end

for (i=1:(2*length(d)))
    if abs(d(i))>dismax/2
        if d(i) > 0
```

```
            d(i) = dismax/2;
        else
            d(i) = -dismax/2;
        end
    end
end
```

A.8 Mirrors computation

```
function [M] = mirror (C,P)
% Andrea Baccara 2007
% This function take as input the vertex of the regions of Voronoi and the
% position of the Agents, and compute the matrix M of all the agents added
% with the fake agents build by a mirroring action of the real ones.

% Initializing:
% Cn has the same information of C, but put the information in a square
% matrix where in rows we have the vertex of the region for each agent.
    Cn = 0*ones(length(C));
    %it build the Cn matrix
    for (i = 1:length(C))% number of Voronoi's regions
        for (k = 1:size(C{i},2))
            Cn(i,k) = C{i}(k);
        end
    end

    %it delete the infinite from the vertexs
    for (k = 1:length(Cn(:,1))*length(Cn(1,:)))
        if Cn(k) == 1
            Cn(k) = 0;
        end
```

```
end

%it build the Nb matrix:
%Nb é la matrice dei vicini, ovvero mi dirà quanti
%vertici l'agente i ha in comune con l'agente j
Nb = 0*ones(length(Cn));%Nb é la matrice dei vicini, ovvero mi dirà quanti
%vertici l'agente i ha in comune con l'agente j
for (k = 1:length(Cn(1,:))*length(Cn(:,1)))
    if Cn(k)  = 0
        [a,b] = find(Cn == Cn(k));
        for (i = 1:length(a))
            for (j = (i+1):length(a))
                Nb(a(i),a(j)) = Nb(a(i),a(j))+1;
                Nb(a(j),a(i)) = Nb(a(j),a(i))+1;
            end
            Cn(a(i),b(i)) = 0;
        end
    end
end

%it build the NeighNum matrix
%it return the number of Neighbors for each agent
    NeighNum = 0*ones(1,length(Nb));
    for (i = 1:length(Nb(:,1)))
        for (j = 1:length(Nb(1,:)))
            if Nb(i,j)  = 0
                NeighNum(i) = NeighNum(i) + 1;
            end
        end
    end

    %it build the mirrors
    ai=1;
```

```
for (i = 1:length(NeighNum))
    if NeighNum(i) < 6 %then the agent is external and I build the Mirrors

        for (j = 1:length(Nb(i,:)))
            if Nb(i,j)  = 0
                Mirr(1,ai) = 2*P(1,i) - P(1,j);
                Mirr(2,ai) = 2*P(2,i) - P(2,j);
                ai=ai+1;
            end
        end
    end
end

% it delete the duplicate mirrors.
M(:,1) = Mirr(:,1);
M = [P M];
Dmir = 0.4;
for (k=2:length(Mirr))
    N=size(M,2);
    t=0;
    for (i=1:N)
        if norm(Mirr(:,k)-M(:,i))<Dmir
            t=0;
            break
        else t = 1;
        end
    end
    if t==1
        M = [M Mirr(:,k)];
    end
end
```

A.9 Waypoint computation for rendez-vous controller

```
function [wyp] = newWyp (O,rO,disO,W,Wc,radgr)
% Andrea Baccara 2008
% it computed the new waypoint (rendez-vous point)
% W gives the old waypoint. Wc is the destination point, radgr is the
% radium of the formation.

[dism,obm] = min(disO);

if (O(1,obm)-W(1)) > 0.02
    mO = (O(2,obm)-Wc(2))/(O(1,obm)-Wc(1));
elseif (O(1,obm) - Wc(1)) < -0.02
    mO = (O(2,obm)-Wc(2))/(O(1,obm)-Wc(1));
else mO = pi/2;
end
mmO = atan(mO);

% the coefficient 0.7 is for the safety distance
    wypc(1) = (rO(obm) + radgr + 0.7)*cos(mmO) + O(1,obm);
    wypc(2) = (rO(obm) + radgr + 0.7)*sin(mmO) + O(2,obm);

    for (i=1:size(W,2))
        wyp(:,i) = (wypc)' - Wc + W(:,i);
    end
```

A.10 Noise generation

```
function [noise] = rumore (N,Vn,Wn)
% Andrea Baccara 2008
```

```
for (i=1:N)
    noise(1,i) = (cos(Wn(1,i))*Vn(1,i));
    noise(2,i) = (sin(Wn(2,i))*Vn(2,i));
end
```

Appendix B

Geometrical issues

B.1 Angular coefficient of a straight line

Describing one straight line in bi-dimensional space as $y = mx+q$, we call angular coefficient the parameter m that has the propriety to have the value of the tangent of the angle between the X-axe of the reference and the straight line; q is the value of the intersection between the straight line and the Y-axe.

Having two points in the space (x_1, y_1) and (x_2, y_2) we can compute the value of m as:

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

B.2 Distance between a point and a straight line

In a bi-dimensional space, we can write a straight line as $y = mx + q$, and having a point $P = [x_0, y_0]$, we can compute their distance as:

$$d = \frac{y_0 - mx_0 - q}{\pm\sqrt{1 + m^2}}$$

the $\pm$ symbol is placed to ensure to have a positive value of the distance.

B.3 Distance between two points

We compute the distance between two points $x = [x_1, x_2, \ldots, x_{\mathfrak{N}}]$ and $y = [y_1, y_2, \ldots, y_{\mathfrak{N}}]$ in a $\mathfrak{N}$-dimensional space as:

$$distance(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_{\mathfrak{N}} - y_{\mathfrak{N}})^2}$$

B.4 Cosine or Carnot's theorem

Using the Carnot's theorem we can compute, knowing the length a, b and c of a generic triangle, the angle α between the two sides b and c:

$$\begin{aligned} a^2 = b^2 + c^2 - 2bc \cdot cos\alpha \Rightarrow \\ \Rightarrow \alpha = arccos(\frac{c^2 - a^2 - b^2}{2ab}) \end{aligned}$$

B.5 Linear and angular speed

We can define the linear and the angular speed of an agent $x = [x, y]$ in a bi-dimensional space as:

$$\begin{aligned} \nu &= \sqrt{\dot{x} + \dot{y}} \\ \omega &= \frac{\ddot{y}\dot{x} - \ddot{x}\dot{y}}{\dot{x} + \dot{y}} \end{aligned}$$

Appendix C

Basic issues

C.1 Holonomic agent

May be defined as holonomic a system which all constraints are holonomic then if all the constraints can be described as follows: $f(x_1,\ x_2,\ x_3, \ldots, x_N, t) = 0$ where the constraints depend only to the variables x_j and the time t. In robotics, we define holonomic robots the ones that have the number of controllable degrees of freedom equal or greater than the number of all possible degrees of freedom. A robot is considered to be redundant if it has more controllable degrees of freedom than degrees of freedom in its task space.

C.2 Observability

A linear system

$$\begin{cases} \dot{x}(t) = Ax(t) \\ y(t) = Cx(t) \end{cases}$$

is observable if [7], for every initial state $x(0)$, the value of the state can be evaluated basing on the observation of the free evolution for a $t_f \geq 0$. Defying an observability matrix:

$$\mathfrak{O} = [C|CA|CA^2|\ldots|CA^{n-1}]^T$$

if and only if $rank(\mathfrak{O}) = n$ - where n is the dimension of the state - the system is observable.

Appendix D

Notation

a, b, c generic sides of a triangle;

ck sampling time;

d distance;

CG barycenter or center of gravity;

$\mathcal{C}_i$ $\mathcal{O}_i \bigcup \mathcal{Q}_i$;

i index of the generic agent;

j index of the generic agent;

k generic time instant in discrete time;

m number of inputs;

M number of obstacle;

$\mathcal{M}$ Mirror;

N number of agents;

n number of dimensions of the space;

o generic obstacle;

$\mathcal{O}_i$ set of obstacles found by the agent i;

$\mathcal{N}_i$ number of neighbors of agent i;

p position of the agent;

q state of the agent;

$\mathcal{Q}_i$ set of robots that can cause crash with the agent i;

R generic agent/robot;

r radium of a generic obstacle;

t time variable;

t_0 starting time;

t_k generic time in discrete time;

u_n command law (input);

x_i i-th coordinate on a n-dimensional space;

X_o x-coordinate of the i-th obstacle;

X_w x-coordinate of the generic waypoint;

y output;

Y_o y-coordinate of the i-th obstacle;

Y_w y-coordinate of the generic waypoint;

$\mathcal{W}$ waypoint;

α generic angle;

ν linear speed;

ρ radium of each agent;

ω angular speed;

θ orientation of the agent;

Ω ensemble of all the obstacle;

Bibliography

[1] Bauso, D., Giarré, L., and Pesenti, R. (2007). Distributed consensus for swithched networks with unknown but bounded disturbances. In *3rd international workshop on network control systems, Nancy, France.*

[2] Carli, R., Chiuso, A., Schenato, L., and Zampieri, S. (2007). Consensus algorithm design for distributed estimation. In *3rd international workshop on network control systems, Nancy, France.*

[3] Defoort, M. (2007). Commande robuste décentralisée à l'horizon glissant d'une flottile de robots mobiles. Technical report, LAGIS UMR CNRS, Ecole centrale de Lille, BP48, cite scientifique, 59 651 Villeneuve-d'Ascq, France.

[4] Defoort, M., Floquet, T., Kokosy, A., and Perruqueti, W. (2006). Integral sliding mode control for trajectory tracking of a unicycle type mobile robot. In *Decision and Control, 46th IEEE Conference on.*

[5] Dimargonas, D. and Kyriakopoulos, K. (2007). On the rendezvous problem for multiple nonholonomic agents. In *IEEE Transactions on Automatic Control*, volume 52.

[6] Ferrari-Trecarte, G., Galbusera, L., Marciandi, M., and Scattolini, R. (2007). model predictive control schemes for consensus in multi-agent systems with integrator dynamics and time-varyng communication. In *3rd international workshop on network control systems, Nancy, France.*

[7] Giua, A. and Seatzu, C. (2006). *Analisi dei Sistemi Dinamici.* Springer Italia. University of Cagliari.

[8] Hu, X. and Stotsky, A. (2001). Control of mobile patforms using a virtual vehicle approach. In *IEEE transactions on automatic control*, volume 46,N°11.

[9] Hu, X., Alarcon, D., and Gustavi, T. (2003). Sensor-based navigation coordination for mobile robots. In *Proceedings of the 42nd IEEE Conference on Decision and Control Maui, Hawaii USA.*

[10] Khatib, O. (1998). Real time obstacle avoidance for manipulators and mobile robots.

[11] Kuwata, Y., Richards, A., Schouwenaars, T., and How, J. (2006). Decentralized robust receding horizon control for multi-vehicle guidance. In *IEEE American Control Conference*, pages 2047,2052.

[12] Lindhé, M. and Johansson, K. (2006). A formation control algorithm using voronoi regions. Technical report, CTS-HYCON Workshop on Nonlinear and Hybrid Control, Paris, France.

[13] Olfati-Saber, R. (2006). Flocking for multi-agent dynamic systems: Algorithms and theory. Technical report.

[14] Roy, N. and Dudek, G. (1997). Learning to rendezvous during multi-agent exploration. In *Proceedings of the Sixth European Workshop on Learning Robots, Brighton, UK.*

[15] Sarkar, N., Yun, X., and Kumar, V. (1993). Dynamic path following: a new control algorithm for mobile robots. In *Proceedings of the 32nd conference on decision and control, Sant'Antonio - Texas, USA.*

[16] Tanner, H., Jadbabaie, A., and Pappas, G. (2003). Stability of flocking motion. Technical Report MS-CIS-03-03. University of Pennsylvania.

[17] Theys, J. (2005). Joint spectral radius: theory and approximations. Doctorate thesis, Université Catholique de Louvain, Center for systems engineering and applied mechanics.