A HIERARCHICAL OPTIMIZATION APPROACH FOR COOPERATIVE VEHICLE NETWORKS

By

CARLO BRANCA

Laurea Degree

Università di Roma Tor Vergata

Rome, Italy

2003

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER of SCIENCE
December, 2005

A HIERARCHICAL OPTIMIZATION APPROACH FOR COOPERATIVE VEHICLE NETWORKS

Thesis Approved:

Dr. Rafael Fierro

Thesis Adviser

Dr. Carl Latino

Dr. Carlos Oliveira

Dr. A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGMENTS

My thanks to my thesis adviser, Dr. Rafael Fierro for allowing me to be a part of the MARHES laboratory, for his sincere encouragement, support, and advice and for taking time to have many discussions with me.

I would like to thank Dr. Carl Latino and Dr. Carlos Oliveira for serving on my committee.

Thanks to my friend Dr. John Wilson for patiently proofreading this work.

I like to thanks all the members of the MARHES laboratory for their help and sharing with me this learning experience.

A special thank to all the members of the Cultural Italian American Organization and of the Latin American Student Association since they have been my family during these two years.

This thesis is dedicated to my grandparents for being an example and an inspiration in my life and to my uncle and godfather Mario. I believe I would have made him proud of me.

Carlo Branca

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Table Page

Chapter 1

Introduction

1.1 Motivation

During the past few years the interest in developing new control algorithms and control tools for cooperative mobile robots has been constantly increasing. This is due to the desire of substituting men with mobile robots in hazardous environments both in the civilian and military domains.

It is clear that a team of mobile robots can accomplish several tasks in a more efficient way, some examples could be mine sweeping, search and rescue and environmental monitoring. Having a team, instead of a single agent, will permit differentiation of skills among the agents. In this way, it would be possible to have several smaller agents equipped with different sensors or actuators instead of a single multi equipped agent. Groups of smaller differentiate agents would have advantages in the mobility point of view. In fact, it is easier for a small agent to move in a hostile environment. Advantages will also come in the reliability point of view, for example, if an agent with a specific skill fails it may be substituted from another agent in the team with the same skill, ensuring the accomplishment of the task.

On the other hand, the presence of multiple unmanned agents brings up problems

regarding the interaction among them. It is desired, in fact, that they cooperate, exploiting their different capabilities to accomplish the common task, while avoiding to interfere within each other. All teammates are both a necessity, because they compensate for lack of skill, and a possible threat because their presence can lead to a collision or in general limit the range of maneuvers.

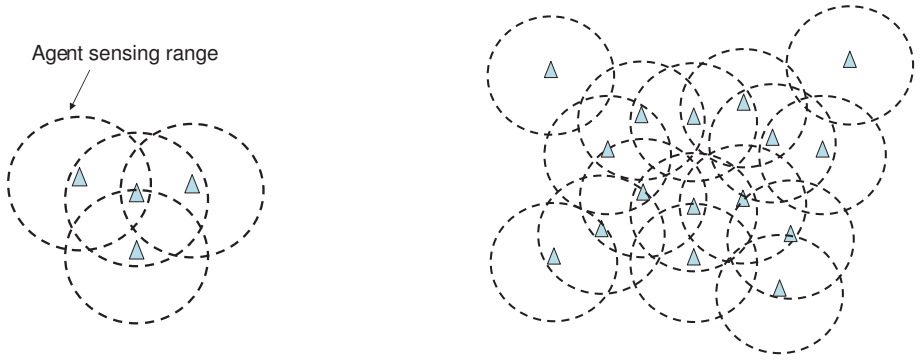
It is obvious, then, that there must be a coordinating algorithm able to deal with all these issues. The algorithm also needs to have others properties, like scalability over the number of robots (it is desirable to use the same algorithm for small and large groups) and the computation power requested. Since resources are limited, like limited amount of power supplies, communication distance and so on, it is necessary to design a control scheme that takes into account these constraints and gives an optimal solution.

(a) Geese flying in the V formation. (b) School of fish.

Figure 1.1: Example of cooperation in nature.

One of the first attempts to address the coordination problem was by borrowing from nature. There are several examples in nature of entities working together to accomplish a task in a more efficient way. For example, geese fly in the characteristic V formation because this formation minimizes the energy required to fly, or fishes move together as a school such that they appear bigger to possible predators, see Fig. 1.1.

By studying animal behavior, simple coordination rules have been proposed to solve the coordination problem [23]. Those rules apply to each agent and are based only on the local sensing information and they do not require any kind of communication. For these reasons, they are easy, and computationally inexpensive. They also scale really well on the number of agents because even though the number of agents in the team is increasing the only possible interaction that can happen is among agents in the sensing range, see Fig. 1.2. The application of the rules derived from nature lead to what is usually defined *flocking*. Flocking is an interesting emergent behavior that has been and it is widely studied to prove stability and convergence properties [23]. Unfortunately, flocking is not enough in most cases and more complex cooperation techniques are required. Moreover, flocking does not contain any optimization criteria.



(a) Team with few agents

(b) Team with many agents

Figure 1.2: When the number of agent increases the number of sensed agent does not increase too much.

A possible alternative to address the coordination problem is by enforcing some kind of formation. Several control algorithms for formation control can be found in the literature and this is a current topic. Although formation control has most of the proprieties of simplicity, scalability and optimality, this approach is not flexible enough when the task becomes complex.

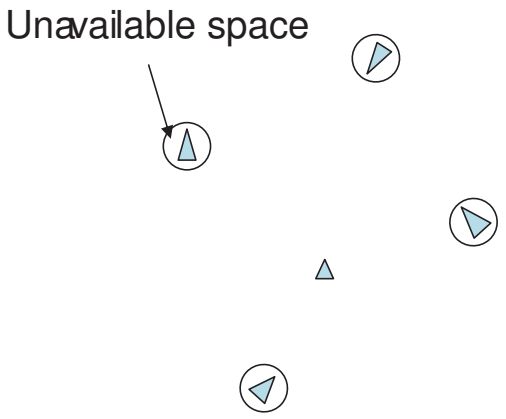


Figure 1.3: The space of the possible position available for the agent is made non-convex due to the presence of other agent.

The method used to address the cooperation problem in this work is the model predictive control algorithm (MPC) or receding horizon control (RHC). This chosen algorithm will be explained in more details in the next chapters. This algorithm contains the properties needed to solve the problem. In fact, the MPC is based on an optimization problem, so the control input will fulfill the optimality criteria. Moreover, since the control law is computed on-line it will be possible to consider dynamic changes in the environment. It also allows to include in the problem constraints on the inputs and the state variables. The problem of having a flexible algorithm, even thought the task gets complicated, is not solved yet. In fact, the use of classical convex optimization does not easily allow modeling the tasks when they involve logical statements or implications. An easy example is collision avoidance among teammates. This is a basic requirement for any cooperative algorithm. To avoid collisions each agent needs to stay away from the areas where others agent are, see Fig. 1.3. This makes the space of possible positions of each agent a non-convex set. For these reasons, convex optimization cannot be used.

For the mentioned reasons it has been decided to use, instead of classical convex optimization, mixed integer linear optimization (MILP). MILP offers a series of tools to model logical statements, as for example AND and OR, or implication statements, like IF-THEN or IF-THEN-ELSE. Moreover, the use of discrete variables makes possible to model a wide range of interesting problems.

1.2 Contributions

Mixed integer linear programming has been widely used for off-line trajectory planning for the flexibility it offers in modeling several different problems. Lately, the MILP has been used for the underlaying optimization problem in the MPC algorithms. The problem that has to be faced is the time constraint MPC brings out. In fact, when the MPC technique is used, the optimization problem needs to be solved as fast as possible. On the other hand, the MILP is an *NP-hard* optimization problem and so the solution time may grow exponentially.

The first attempt we made to solve the cooperative control problem was by using the MPC algorithm in which the underlying problem was formulated as an MILP. The problem with this approach is that it does not scale well on the number of robots in the team and on the complexity of the environment and of the task. This is due to the fact that the MILP becomes too hard to be solved efficiently.

The main contribution brought to this work is a different approach to the cooperation problem. Instead of formulating a single global flat problem to represent the cooperation problem, a hierarchy structure is introduced. In particular, two levels are defined. The higher-level makes decisions about tasks related to the whole formation, and is actually the one taking into account all the cooperating tasks. The

lower-level offers services related to the single agent. At this level will be taken into account problems like collision avoidance, obstacle avoidance, and all other tasks not requiring an explicit cooperation among agents.

Using this new structure, instead of solving a single more complicated problem, several easier problems will be solved. This will reduce the time needed to solve the optimization problem making the MPC/MILP implementable even when the number of agents increase and the environment and the tasks get complicated. The cost associated with the reduced computational time is the loss of global optimality. In fact, the solution obtained with this hierarchical approach may be worse than the one it would be obtained solving the single global problem.

1.3 Thesis outline

This thesis is organized in the following way. In chapter two, a literature review about coordination control is given. In this chapter all the mathematical tools that will be used in the analysis are also introduced. In particular, basic concepts about model predictive control algorithm and mixed integer linear programming will be covered.

In chapter three, the problem is formulated. In this chapter an overview of the task required from the team is given. The model of the robots and the environment in which they are supposed to move is also described. The chapter ends with a formal definition of hierarchical and decentralized optimization.

In chapter four, the model predictive control based on the global optimization problem is described. After defining the open-loop optimization problem, a brief description of the simulator used to test the algorithm is given. At the end of the chapter, the simulations obtained are reported and a discussion about the applicability

of the algorithm when the size of the team or the complexity of the environment increase is done.

Chapter five is dedicated to the hierarchical/decentralized approach. The first part of the chapter is dedicated to the formulation of the higher and lower level optimization problems. Some heuristics to improve the performance of the algorithm are also presented. Then, it is given a brief explanation of the simulator used to test the algorithm. The chapter is concluded with the simulation result and a comparison between the performance in term of optimality and computational time of the two algorithms presented.

The last chapter is dedicated to some conclusions and future work.

Chapter 2

Related work

In this chapter, after a review about the possible strategies for the coordination of team of mobile unmanned robots, the control theory and the mathematical tools used to design the proposed cooperative control algorithm will be presented and discussed.

2.1 Coordination control: literature review

It is necessary to deal with a coordination control problem any time there is more than one agent cooperating to accomplish a common task. There are several tasks in the technological world that have to be made or can be made better with multiple agents. Example are the search and rescue tasks or the perimeter surveillance. There are also many systems that are the composition of different subsystems dynamically connected and that work together to accomplish a common task. Some examples of these systems are a network for energy distribution or a manufacturing system.

The main objective of a multi-robot system is accomplishing a common goal by exploiting the capabilities of each member of the team.

There are at least two possible ways to solve the coordination problem :

- The behavioral approach: this means starting studying the general behavior of

the formation and then going down approaching step by step the behavior of the single agent. One of the basic works in computer science about the formation control is the one from Reynolds [23]. In this paper, he suggested a protocol, made of three easy rules, to describe the behavior of flocks, herds and schools. All the work is based on these rules, without the use of equations.

- The system theory approach: the problem is faced bottom-up. In other words, firstly it is studied the control of the single agent and then how the agents can collaborate with each other, trying to formally demonstrate the properties of stability and feasibility.

Our attention will be focused on the latter approach.

The kind of systems we will consider are those composed of agents spatially distributed and dynamically separated. For example, a team of mobile robots, the agents are dynamically independent but coupled by the common goal and the constraints on the state. Continuing with the example of the mobile robot, it is clear that each robot has its own dynamic that is not affected by the dynamics of the others robots and can be independently actuated. However the behavior of the robot will be decided based on the state of the others robots either to accomplish the common task or to avoid possible collisions or meet constraints due to communication problems.

One possible approach is to solve this control problem is in a centralized manner. This means to define a model for the system that takes into account each detail of the team, define the cost function that represent the problem, and finally solve the problem finding all the control actions. The control problem posed in this way can became very difficult even when the size of the system is small. In addition, it has to be considered that these kinds of problems occur in real time. This means that the control problem has to be solved while the system is working, so the solution

of the problem has to be found in the shortest time possible. Even exploiting the capacity of recent computers, this approach is almost unfeasible. For this reason it is necessary to define a modular approach to make the problem easier so that it can became computationally acceptable.

To reach this objective there are plenty of strategies, which may be divided into two categories. One possibility is to approach the problem in a decentralized fashion. This means that, instead of solving a central problem that involves each agent, a problem is solved for each agent taking into account only the state of its neighbors. In this way, the size of the problem is reduced and is easier to solve. The challenge with this approach is how to define the local cost function, how the state of neighbor agents may affect the state of the one considered, and how to prove global stability, feasibility, and optimality. A second possible way is using a hierarchical approach. In this case, different layers are defined, starting from the highest one that considers a simplified model going to the lowest one that has a detailed model of the system. The advantage with this technique is that each layer makes decisions only about a restricted set of variables, because some are given by the highest level and others are left for the lower level.

The literature about coordination control of robot teams is quite large. Several different approaches have been tried to address the problem. For instance, there are some researchers that have based their work on the rules proposed by Reynolds [23]. An interesting example can be found in the work of Saber and Murray [27]. In this paper the authors present a theoretical framework for representing a dynamic graph. This tool has been exploited to generate a model that includes all the three roles of Reynold's model. Interesting split and rejoin behavior can be obtained using this framework.

Another work based on the Reynolds' ideas is the one from Jadbabaie *et al.* ([19],[30]). In a first paper they give a proof of the stability of formation driven by the Reynolds' protocol. In a successive paper the author suggests a control approach that mixes properties from graph theory, classical non linear theory, and potential field theory that allows to prove stability property for the system in presence of switching in the interconnection topology. Similar results are obtained by Tanner [20], but considering a nonholonomic model for the robot. More interesting approaches are those considering graph theory and optimization.

A possible way to control a robot's formation is defining a control graph ([7], [8], [9]). The control graph consists of a triple $(g, r, \mathcal{H})$ where g is the trajectory of the reference that could be one of the robots in the formation or a virtual robot, r is a vector of shape variables that describe the position of a robot respect the reference, and $\mathcal{H}$ is a graph that describes the behavior of the robots in the formation. This approach is in the middle between a centralized approach and a decentralized one, in fact there is a central agent that defines the control graph and then there are local controllers (leader follower) that guarantee the convergence to desired formation. Different works in the literature are based on the control graph. Two examples are the work from Das *et al.* [6] and the one from Fierro *et al.* [16]. They develop a set of simple control strategies like the leader follower, the three robot shape control or the leader-obstacle control using the input-output feedback linearization, and based on these controls they suggest a coordination protocol to switch from one controller to another. The controllers based on the control graph have the problem that the graph $\mathcal{H}$ is assigned based on some heuristics and so it is not guaranteed to be an optimal choice.

Other works address the problem using optimization techniques. Two categories

can be highlighted: centralized approach and decentralized approach.

Even if the centralized approach could be too expensive in term of computation power required, there are several works in the literature that use this approach. Furthermore, the centralized control could be considered the basic step in solving the decentralized problem.

One of the most interesting works is the one from Dunbar and Murray [10]. In this paper the authors apply the model predictive control to the problem of formation control (MPC). The basic idea of the MPC is to define a cost function that measures the performance of the system on an interval T called prediction interval and then minimize this cost function constrained by the system's dynamic equations and other eventual constraints. By solving the minimization problem we obtain the optimal input to the system for all the prediction intervals. To have a closed-loop control only the first sample of the solution is used, then the problem is solved again with the new values of the state's variables. In another paper Saber, Dunbar and Murray [26] propose a systematic way to define the cost function for the MPC using a cost graph.

A similar approach has been used from Wesselowski and Fierro [32]. In their work the authors use a dual mode MPC for the formation control. The dual MPC differs from the standard MPC in the way that the MPC drives the system into a terminal set instead in the equilibrium point. When the system is in the terminal set the control is switched to a local control that drives the system to the equilibrium state. In their paper the authors use the input output feedback linearization as a local controller.

Several researchers have tried to find a decentralized approach to formation control in order to reduce the complexity of the problem. Unfortunately, when a decentralized

approach is used others issues arise. In fact, with the decentralized MPC the property of stability and feasibility are lost. To recover these properties, the literature proposes several strategies. Dunbar and Murray [11] propose a decentralized approach in which each agent defines and solves its own control problem and then transmit the solution to all its neighbors. The constraint requires that the open loop trajectory obtained at the sampling time k differs less than a certain constant from the one obtained at sampling time k-1. In this way, under certain conditions they prove the stability of the decentralized control.

Another possible solution has been formulated from Shim *et al.* [28]. They suggest a formulation of the distributed MPC that is mixed with a potential field approach.

The problem of dividing a decoupled system in several subsystems to apply a decentralized MPC control has been treated also by Keviczky, Borrelli and Balas [21].

The problem of coordination control has been faced into centralized way also using mixed integer linear programming (MILP). This approach allows conversion of logical rules in a mixed integer linear constraints. Having the possibility to use logical statements makes easier to model some formation behavior or tasks. In the paper of Bemporad and Morari [3] it is possible to find an extensive description about how to convert logical constraints into mixed integer linear programming and how to combine them with the model predictive control algorithm. The literature about the coordination problem using MILP is quite rich. A general idea about how mixed integer linear programming can be used in a control loop can be found in [25]. In that paper the authors, after giving a general overview about the structure of the algorithm, give some properties and insights about the modeling and solution techniques for MILP optimization problems. An example of the use of MILP for

cooperative problem can be found in [22]. In this paper the problem considered is coordinating a group of agents to accomplish a time dependant task. Instead, in [18], the authors try to consider the non-linear nature of UAV using the differential flatness property to model the non-linear system in a mixed integer linear formulation. Different kinds of tasks can be coded using MILP. For example, in [1] the problem of coordinating a team of aerial vehicles in an hostile environment is considered. The central problem is how to schedule the task for each UAV such that the risk of failure for the whole formation is minimal. Using MILP inside a model predictive control algorithm requires that a solution for the MILP problem has to be found quickly. For this reason, several works aim to propose different techniques to reduce the solution time. To achieve this goal, two main approaches are possible. The first is by improving the solver algorithm. Some examples pointing in this direction can be found in [29, 13]. In the first one the authors proposed a genetic algorithm for the solution of the MILP problem, while in the second the branch and bound search is specialized on the cooperative problem. The second possible way to reduce the solution time is by simplifying the model by using some alternative approach. An example can be found in [4], in which the author suggested a way to approximate the MILP problem into a simply linear programming problem that can be solved much faster. Another possible approach is the one suggested in [14]. In this work the author presents a way to sample the system based on an unequal distribution. In this way it is possible to have the optimal number of samples and with a higher concentration where it is most needed. In [24] a decentralized approach is suggested. An order among the agents is defined, then a local problem is solved starting from the first one going to the last one. At each stage the local problem is formulated based on the solutions of the previous stages.

In this work to address the computational problem associated with MILP, it will be presented a structure obtained from a combination of a hierarchical and decentralized approaches.

2.2 Optimization

2.2.1 Model predictive control

Model predictive control (MPC) or receding horizon control (RHC) is a control algorithm in which the control input is obtained by solving, at each sampling time, an on-line finite horizon open-loop optimal control problem, using the current state of the plant as initial state. The solution of the optimization problems gives a sequence of control inputs for all the control horizon but only the first element of the sequence is applied to the plant. The reason that made MPC popular, specially in the process control field, is its ability to deal with constraints. Every control problem has to consider constraints, in fact, actuators can give limited forces, safety limits on state must be respected, but at the same time to obtain the best efficiency it is desirable to work close to those limits. Classical control algorithms do not take into account those constraints and often ad hoc methods are used to overcome this problem. Model predictive control, instead, is the perfect tool to combine optimality and feasibility of the control effort.

Consider the system described by the difference equation

$$\begin{aligned} x(k+1) &= f(x(k), u(k)) \\ y(k) &= h(x(k)) \end{aligned} \tag{2.1}$$

where the state x and the input u must satisfy

$$\begin{aligned} x(k) &\in \mathbb{X} \\ u(k) &\in \mathbb{U} \end{aligned} \tag{2.2}$$

where $\mathbb{X} \subset \mathbb{R}^n$ and $\mathbb{U} \subset \mathbb{R}^m$. The general control problem is to drive in the optimal way the state of the system to the origin while satisfying state and input constraints.

The cost function that measures the optimality of the solution is

$$J(x, \mathbf{u}, k) = \sum_{i=k}^{k+N-1} c(x(i), u(i)) + C(x(k+N)) \tag{2.3}$$

where $\mathbf{u} = \{u(k), u(k+1), \ldots, u(k+N-1)\}$, N is called the receding or control horizon, c is the cost that penalize the trajectory and the input and C is the final cost associated with the final state of the system. Under the assumption that $f(\cdot)$ and $c(\cdot)$ are time invariant the cost function is time invariant in the sense that the solution of the optimization problem with the system in the state x at the time k is the same as the solution of the optimization problem from the state x at the time 0 ($\mathbf{u}^o(x, k) = \mathbf{u}^o(x, 0)$), which means, that the solution of the optimization problem depends only on the state x of the system.

The general open-loop optimization problem can be defined as

$$\mathcal{P}_N(x) : J_N^o(x) = \min_{\mathbf{u}} \{J_N(x, \mathbf{u}) | \mathbf{u} \in \mathcal{U}\} \tag{2.4}$$

where, now,

$$J(x) = \sum_{i=0}^{N-1} c(x(i), u(i)) + C(x(N)) \tag{2.5}$$

and $\mathcal{U}$ is the set of admissible control inputs that satisfies all the constraints. The

model predictive control law is then

$$\kappa_N(x) := u^o(0, x) \tag{2.6}$$

The MPC algorithm can be summarized as reported in Algorithm 1.

Algorithm 1 Model Predictive Control algorithm

1: **for all** Sampling time k **do**
2: Sample the state of system to obtain $x(k)$
3: $x = x(k)$
4: Solve the on-line open-loop optimization problem $\mathcal{P}_N(x)$
5: Apply the control law $\kappa_N(x) := u^o(0, x)$
6: **end for**

The early versions of the MPC did not guarantee stability, and for this reason several modifications have been proposed during the years to achieve this goal. It has been demonstrated in [31] that the original MPC model guarantee stability for unconstrained linear systems and constrained stable systems, but for these systems there is an ample literature of well defined optimal control techniques. A first modification that ensured stability for time-varying, nonlinear, constrained discrete time systems was proposed by [17], where, to obtain stability a terminal equality constraint is added. This means the constraint

$$x(N) = 0 \tag{2.7}$$

is added in the problem $\mathcal{P}_N(x)$. This modification ensures stability but it brings up a feasibility problem. In fact, it is possible that there is no feasible solution able to guide the system to the origin if the horizon N is not long enough. From another point of view it can be stated that with this modification the region of attraction for the controller is the subset $X_N \subset \mathbb{R}^n$ of points from which a feasible solution exists.

Another possible modification is the dual-mode MPC. In this version a final constrain set is added to the problem $\mathcal{P}_N(x)$

$$x(N) \in X_f \tag{2.8}$$

with $0 \in X_f$. When the system enters the final set the controller is switched to a local controller that is in charge of driving the system to the origin. The final set can be as big as the region of attraction of the local controller, and the bigger the final set the bigger is the set of points X_N for which a feasible solution can be found. With this approach the attraction region of the controller can be made bigger. In the next sections it will be supposed that the control horizon is long enough such that a feasible solution always exists.

2.2.2 Mixed integer linear programming

There is a wide range of practical problems that can be modeled with discrete and continuous variables and linear constraints. It is clear the need for discrete variables in all optimization problems regarding discrete goods. For example, in the problem of optimizing the number of cars produced by a plant, an integer solution is desirable because a fractional solution would not make sense. Although this may seem the obvious application for discrete optimization, when there are problems involving discrete goods it is usual to neglect the integer constraint solving a classical linear programming (LP) problem and then rounding the continuous solution obtained from the LP problem.

Discrete optimization shows all its power and flexibility when decision variables are used. Decision variables are discrete variables that can take only the value zero

or one, and they are usually related to the logical TRUE (or yes) and FALSE (or no). For example, if a new plant needs to be built in one among several possible locations, a discrete optimization problem can be modeled using decision variables. In this case there will be a decision variable δ_i for every location that will be one if the new plant is assigned to location i and it will be zero if the plant is not assigned to that location.

Another useful kind of discrete variables are the indicator variables. These variables, like the decision variables, can take only the value zero or one, and they are usually used to indicate the state of certain continuous variable. For example, suppose it is necessary to know if $f(x) \leq 0$, an indicator variable δ will be used such that $f(x) \leq 0$ implies $\delta = 1$.

Now it will be shown how to convert logical statement involving decision and indicator variables into mixed integer linear constraints.

Decision variables are always associated with a statement. For example "$f(x) \leq 0$" or "the i-th task is scheduled in the j-th machine". It is common practice to represent statements with literals, X_i, that has a *truth value* of either "T" (True) or "F" (False). Statements can be combined using logical operators and boolean algebra. Suppose there are two statements represented with the two literal X_1 and X_2 and suppose that two decision variables δ_1 and δ_2 are associated to these literals such that $\delta_i = 1$ if the statement X_i is T and zero viceversa. For example, if in modeling a certain problem it is desired that at least one between the statements is true (a logical OR) the constraint

$$\delta_1 + \delta_2 \geq 0 \tag{2.9}$$

it is clear, that if both the statement are F both the decision variables are zero, but this would violate the constraint, then to satisfy the constraint at least one between δ_1 and δ_2 must be one. A list of the conversion of the most common logical operator

is reported below

$$\begin{array}{lll} X_1 \text{ AND } X_2 & \text{is equivalent to} & \delta_1 = 1,\ \delta_2 = 1 \\ \text{NOT}(X_1) & \text{is equivalent to} & \delta_1 = 0 \\ X_1 \rightarrow X_2 & \text{is equivalent to} & \delta_1 - \delta_2 \leq 0 \\ X_1 \leftrightarrow X_2 & \text{is equivalent to} & \delta_1 - \delta_2 = 0 \\ X_1 \oplus X_2 & \text{is equivalent to} & \delta_1 + \delta_2 = 1 \end{array} \tag{2.10}$$

These logical operators can be combined to model complex logical statements. For example, suppose in a plant it is necessary to open the valve V if the temperature T is high or the pressure p is high. Then, there will be a literal X_1 associated with the statement "valve V open", a literal X_2 associated with "temperature high", and a literal X_3 associated with "pressure high". The statement it is desired to convert in a constraint is

$$X_2 \text{ OR } X_3 \rightarrow X_3 \tag{2.11}$$

this logical statement can be translated in the linear constraint

$$\delta_1 + \delta_2 - 2\delta_3 \leq 0 \tag{2.12}$$

in fact, if $\delta_1 = 1$ (the temperature is high) the only way to satisfy the constraint is by having $\delta_3 = 1$ (the valve is open), and in the same way if $\delta_2 = 1$ or both are 1 the only way to satisfy the constraint is by having $\delta_3 = 1$. On the other hand, if both δ_1 and δ_2 are zero the constraint is satisfied by $\delta_3 = 0$ (the valve is not open).

An important tool in mixed integer linear programming is the **big M** technique. This technique allows to connect indicator variables to the continuous variables. The

basic building block for this technique is the translation of the implication

$$x > 0 \quad \rightarrow \quad \delta = 1 \tag{2.13}$$

that means $x > 0$ implies $\delta = 1$. The implication in (2.13) can be converted into the following mixed integer linear constraint

$$x - M\delta \leq 0 \tag{2.14}$$

where M is a big positive number. To verify the validity of the constraint in (2.14) it is useful to rewrite the inequality in the following way

$$x \leq M\delta \tag{2.15}$$

in this second form it is easy to see that if $x > 0$, the only way to satisfy the constraint is by having $\delta = 1$, that it is what the implication in (2.13) required, while if $x \leq 0$ the constraint is satisfied for every value of δ. Another important implication is the reverse implication respect the one in (2.13), that is

$$\delta = 1 \quad \rightarrow \quad x > 0 \tag{2.16}$$

however the implication in (2.16), it must be slightly modified to be converted into a mixed integer linear constraint. The modified implication is

$$\delta = 1 \quad \rightarrow \quad x \geq \epsilon \tag{2.17}$$

where ϵ is the smallest number for which x is considered to be not zero (it is usually the

machine precision). The implication in (2.17) is translated in the following constraint

$$x - \epsilon\delta \geq 0 \tag{2.18}$$

in this case, it can be easily seen as if $\delta = 1$ the constraint in (2.18) force x to be greater equal then ϵ that is what requested in (2.17), while if $\delta = 0$ no constraint is forced on x (note that it is supposed the linear problem is in standard for so all the continuous variables are greater equal then zero).

It is also useful to know how to translate the following kind of implications

$$\delta = 1 \qquad \rightarrow \quad \sum_j a_j x_j \leq b \tag{2.19}$$

In a similar way it was done before the implication can be translated in the following linear constraint

$$\sum_j a_j x_j \leq b + M(1 - \delta) \tag{2.20}$$

where, as before, M is an upper bound on the expression $\sum_j a_j x_j - b$. From (2.20), it is possible to see as, when $\delta = 1$, the constraint enforce $\sum_j a_j x_j \leq b$. Instead, when $\delta = 0$ the constraint is reduced to $\sum_j a_j x_j \leq M + b$ that is satisfied for all possible values of x_j.

The last basic implication needed for modeling MILP problem is the reversed implication respect the one in (2.19), that is

$$\sum_j a_j x_j \leq b \quad \rightarrow \qquad \delta = 1 \tag{2.21}$$

that is equivalent to

$$\delta = 0 \qquad \rightarrow \quad \sum_j a_j x_j > b \tag{2.22}$$

the mixed integer linear constraint that translates the implication is

$$\sum_j a_j x_j > b + m\delta \tag{2.23}$$

where m is a lower bound on the expression $\sum_j a_j x_j - b$. It can be seen that if δ is equal to zero the constraint ensure $\sum_j a_j x_j > b$ while, if $\delta = 1$ the constraint $\sum_j a_j x_j > b + m$ is automatically satisfied by the definition of m. A problem occurs with the constraint in (2.23). In fact, for the theorem of existence of the optimum the feasible region must be limited and close, but the constraint in (2.23) would define an open set. For this reason the starting implication must be modified in the following

$$\delta = 0 \qquad \rightarrow \quad \sum_j a_j x_j \geq b + \epsilon \tag{2.24}$$

where ϵ is chosen as in (2.17). The modified version of the constraint is

$$\sum_j a_j x_j \geq b + \epsilon + (m - \epsilon)\delta \tag{2.25}$$

In a similar way a linear constraint can be written for the "$\geq$" case, so the implications

$$\begin{aligned} \sum_j a_j x_j \geq b \quad &\rightarrow \qquad \delta = 1 \\ \delta = 1 \qquad &\rightarrow \quad \sum_j a_j x_j \geq b \end{aligned} \tag{2.26}$$

are respectively converted in the constraints

$$\begin{aligned} \sum_j a_j x_j &\leq b + +m(1-\delta) \\ \sum_j a_j x_j &\leq b - \epsilon + (M+\epsilon)\delta \end{aligned} \tag{2.27}$$

Combining all the previous implications a wide range of logical statements and conditions can be modeled in a mixed integer linear problem. For example, suppose it is desired to model the following implication

$$\sum_j a_j x_j = b \quad \rightarrow \qquad \delta = 1 \tag{2.28}$$

but this is equivalent to ask that $\delta = 1$ if both the inequalities "$\geq$" and "$\leq$" are hold simultaneously. It is necessary, then, to use two auxiliary indicator variables δ' and δ'', one for each of the inequality and then require that $\delta = 1$ if $\delta' = 1$ AND $\delta'' = 1$. This is obtained with the set of equations

$$\begin{aligned} \sum_j a_j x_j &\geq b + \epsilon + (m-\epsilon)\delta' \\ \sum_j a_j x_j &\leq b - \epsilon + (M+\epsilon)\delta'' \\ \delta' + \delta'' - \delta &\leq 1 \end{aligned} \tag{2.29}$$

the first two constraints in (2.29) would drive the indicator variables δ' and δ'' to one if the related inequality is satisfied. The last equation drive δ to one if both δ' and δ'' are one.

Another interesting example of the use of the **big M** technique is the conversion

in linear constraints of the following non linear expression

$$y = \delta f(x) \tag{2.30}$$

where δ is a binary variable and $f(x)$ is a linear function. The expression is converted into linear constraints using the following set of equations

$$\begin{aligned} y &\leq M\delta \\ y &\geq m\delta \\ y &\leq f(x) - m(1-\delta) \\ y &\geq f(x) - M(1-\delta) \end{aligned} \tag{2.31}$$

where, as before, M and m are respectively the upper and lower bounds on the function $f(x)$. It can be verified that if $\delta = 0$ the first two equations force y to zero, while the second two are satisfied for every value of y. Instead, if $\delta = 1$, the first two equation are satisfied for every value of y, while the combination of the second two force y to be equal to $f(x)$.

All the logical operators among decision variables and the implications to relate the indicator and the continuous variables will be used in the next chapters to model the cooperative problem. To better understand why the complexity of a MILP problem is exponentially increasing with the number of binary variables a brief overview of the solution techniques for mixed integer linear programming will be presented in the next subsection.

2.2.3 Solving mixed integer linear programming model

While it is possible to efficiently solve linear programming model using the *simplex* algorithm, no algorithm has shown to be efficient in solving mixed integer linear programming, in particular MILP belongs to the class of *NP-hard* problem. All the proposed algorithms to solve MILP model relay on solving the LP problem obtained by neglecting the integrality constraint. Among them, the most commonly used algorithms are the branch and cut algorithm and the branch and bound algorithm. Most of the commercial solvers use a combination of these two algorithms to efficiently solve medium-large sized problems.

The idea behind the branch and cut algorithm is to solve the LP problem obtained by dropping the integrality requirement, and, if the solution obtained is an integer feasible solution the solution of the LP problem is also the solution of the MILP model. If not, a new constraint, called cutting plane, is added such that the previous LP optimal solution is cut but all the integer feasible solutions are hold in the LP feasible region. The branch and cut has shown good performances over a wide range of problem especially because for certain problem *ad-hoc* cutting plane can be used. When a problem does not show any particular structure, the branch and bound algorithm is preferred over the branch and cut.

As for the branch and cut algorithm, the starting point for the branch and bound is the solution of the LP model P_0 obtained by neglecting the integrality constraint. Note that the solution of the relaxed version represents an upper bound on the integer solution. If the LP optimum is integer feasible (all the integer variable are integer) the LP optimum is the optimum for MILP. If the solution is not integer feasible one of the discrete variables δ_i that is not integer in the solution is chosen and two new problems are formulated one P_1 with the additional constraint $\delta_i = 0$ and the other

one P_2 with the constraint $\delta_i = 1$. The process of choosing a variable and add the related constraints is called the branching phase. Note that, when a constraint is added the solution get worst than the original problem. After the branching phase, the problem P_1 and P_2 are added to the stuck of the open problems $\mathfrak{P} = \{P_1, P_2\}$. The first problem in $\mathfrak{P}$ is taken and solved. If the solution is integer feasible the branch is closed, and the solution and the value of the solution are saved. If the solution is not integer feasible another branching phase is taken, and the two problems generated are added on the top of $\mathfrak{P}$. The branching proceeds until an integer feasible solution is found. The branching phase will generate a search tree, as the one reported in Fig. 2.1. An integer feasible solution is a lower bound on the integer optimal solution. For this reason all the branches that lead to problems with solution worse than the best integer feasible solution can be closed and removed from the $\mathfrak{P}$. This phase is called bounding phase. The branching and bounding phase are repeated until the stack $\mathfrak{P}$ is empty. Once $\mathfrak{P}$ is empty, the best integer feasible solution is the MILP optimum.

The branch and bound algorithm is reported in Alg. 2.

An example is reported to clarify the algorithm. Suppose to have a minimization problem

$$\begin{aligned} &\min_{y} \sum c_i y_i \\ &\text{s.t.} \\ &Ay \leq b \end{aligned} \tag{2.32}$$

with y_i binary variables. At the first step solving the LP problem P_0 a non integer solution is found with a value of 10 (see Fig. 2.1 as reference). To generate the problem P_1 and P_2 the variable y_1 is chosen. At the next step the problem P_2 is

Algorithm 2 Branch and Bound algorithm

1: $\mathfrak{P} = \{P_0\}$
2: $opt = \infty$
3: **while** $\mathfrak{P} \neq \emptyset$ **do**
4: Take off the first problem P in the stack $\mathfrak{P}$
5: Find the solution x' whose value is f'
6: **if** x' is integer feasible **then**
7: **if** f' is better than opt **then**
8: $opt = f'$
9: remove all the problems in $\mathfrak{P}$ that lead to a leaf with optimal solution worst than opt (bounding phase)
10: **end if**
11: **else**
12: Choose a branching variable δ
13: Generate the two new problem P^i and P^{ii} having respectively the constraints $\delta = 0$ and $\delta = 1$
14: AddP^i and $P^i i$ at the top of the stack $\mathfrak{P} = \{P^i, P^{ii}\} \bigcup \mathfrak{P}$
15: **end if**
16: **end while**

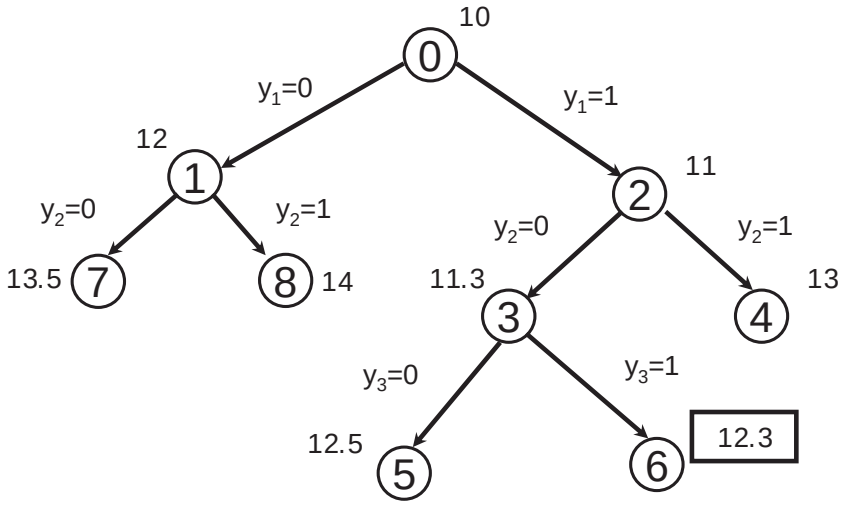


Figure 2.1: Search tree generated from the Branch and Bound algorithm.

chosen and solved. The solution is non-integer with value of 11. The next branching phase is made over the variable y_2 so that the problem P_3 and P_4 are generated. The next problem solved is P_3. Its solution is non-integer with value 11.3. The branching phase generates problem P_5 and P_6. Solving P_5 brings to the first integer feasible solution with value 12.5. The actual solution is saved. When problem P_6 is solved another integer feasible solution is found with value 12.3 that is a better solution respect the previous one, so the saved solution is updated. At this point all the branches generated from P_3 are visited, then the problem P_4 is solved. Its solution is not integer and its value is 13. Since all the branches that will be generated from P_4 will have value higher then 13, there is no sense in proceeding the branching phase and the branch can be closed. Going back to problem P_1, its solution gives a non integer solution of value 12. In this case the branch cannot be closed since the upper bound represented from the linear solution is not higher then the actual solution. Two new problems are generating by branching the variable y_2. Solving the two problems brings to non-integer solution with values 13.5 and 14 that both are higher than the actual feasible solution and then they can be both closed. Since there are no more branches to visit, the algorithm ends and the optimum is 12.3.

It is easy to see that the number of possible linear problems that could be necessary to solve increases exponentially with the number of binary variables. That is the reason why, when the number of binary variables increases, the computation time required to solve the problem may blow up. The second part of the thesis will be presenting a method to keep the number of binary variables in each MILP problem low.

Chapter 3

Problem formulation

This chapter introduces the coordination problem. First, the task that the team is required to accomplish is introduced. Then, the model considered for the mobil robot is presented. In the third section, the environment in which the team is supposed to operate is described. Finally, an introduction about the concepts of hierarchical, distributed and decentralized optimization are given.

3.1 The coordination problem

This work will focus on the problem of coordinating a team of mobile robots working together to accomplish a common task. Since the central point of the thesis is not modeling tasks using MILP but in presenting a different structure to state the problem, it will consider as common task the assignment problem. In particular it is supposed to have a set of targets N_T that must be reached by the robots N_R. It is also required that each robot is assigned to a single target and each target has one assigned robot. These two requirements imply that $N_R = N_T$.

It is also required that during the accomplishment of the task the team members avoid collision with each other and avoid the obstacles present in the environment.

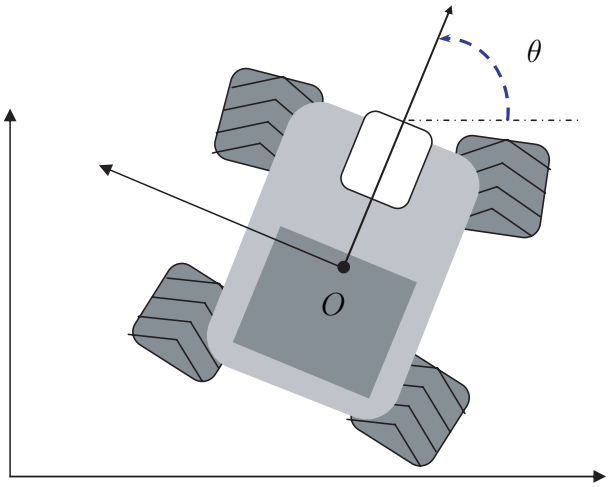


Figure 3.1: Robot's set of coordinates

3.2 Robot's model

The control algorithm developed is intended to be used on a car like platform. In particular the test bed available in the MARHES laboratory is a team of ten 1/10 scale monster trucks. The platform is used as a mobile sensor.

The model used to describe a car like robot is the unicycle model

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \omega \end{aligned} \tag{3.1}$$

where x and y are the coordinates of the origin of a system of coordinates fixed to the robot with respect to a global system of coordinates and θ is the orientation of the system of coordinates of the robot with respect to the global one. Finally, v and ω are respectively the linear and angular velocity of the robot in the global set of coordinates. A drawing of the robot with the associated set of coordinates is shown in Fig. 3.1.

To be able to pose the coordination problem as a mixed integer linear programming problem it is necessary to have a linear model for the robot. For this reason, as extensively done in the literature, it is supposed there is a lower-level controller in charge of the nonlinear behaviors of the robot and that makes the robot to behave as a double integrator with a dumping factor. The model considered for the robots will be

$$\begin{aligned} \ddot{x} &= -b\dot{x} + u_x \\ \ddot{y} &= -b\dot{y} + u_y. \end{aligned} \tag{3.2}$$

To formulate the mixed integer linear programming problem it is also necessary to have a discrete system so the sampled version of the previous model will be considered. The discrete model is obtained considering the following approximation for the derivative:

$$\begin{aligned} \ddot{x} &\simeq \frac{\dot{x}(k+1) - \dot{x}(k)}{\Delta T} = -b\dot{x}(k) + u_x(k) \\ \dot{x} &\simeq \frac{x(k+1) - x(k)}{\Delta T} = \dot{x}(k) \end{aligned}$$

considering the same approximation for the y component and substituting $\dot{x}$ and $\dot{y}$ with respectively v_x and v_y the equation can be rewritten as

$$\begin{aligned} x(k+1) &= x(k) + \Delta T v_x(k) \\ v_x(k+1) &= (1 - b\Delta T)v_x(k) + u_x(k) \\ y(k+1) &= y(k) + \Delta T v_y(k) \\ v_y(k+1) &= (1 - b\Delta T)v_y(k) + u_y(k) \end{aligned}$$

that in matrix form becomes

$$\begin{pmatrix} x(k+1) \\ v_x(k+1) \\ y(k+1) \\ v_y(k+1) \end{pmatrix} = \begin{pmatrix} 1 & \Delta T & 0 & 0 \\ 0 & 1-\Delta T\, b & 0 & 0 \\ 0 & 0 & 1 & \Delta T \\ 0 & 0 & 0 & 1-\Delta T\, b \end{pmatrix} \begin{pmatrix} x(k) \\ v_x(k) \\ y(k) \\ v_y(k) \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ \Delta T & 0 \\ 0 & 0 \\ 0 & \Delta T \end{pmatrix} \begin{pmatrix} u_x \\ u_y \end{pmatrix}.$$

In a more compact way, this can be written as

$$X(k+1) = A\, X(k) + B\, U(k) \tag{3.3}$$

with the obvious meaning of the symbols. This is the model considered in the rest of the thesis. It is supposed that the robot has constraints on the maximum acceleration and on the maximum velocity in particular:

$$u_{x|y} \leq U_{MAX} \qquad v_{x|y} \leq V_{MAX}. \tag{3.4}$$

Some considerations about the choice of the sampling period ΔT are in order. As for all sampled systems, it is desirable to have the shortest sampling period so that the sampled model is as close as possible to the continuous one. Having a short sampling period would improve the process of avoiding obstacles and collisions. In fact, since between two samples no control action can be made to repair for an unexpected situation, the shorter is the sampling period the sooner the repairing action can be taken. On the other hand, since a model predictive control algorithm is used some others aspects need to be considered in the choice. First of all, to make the optimization problem feasible the control horizon must to be held long enough. Holding the same control horizon and reducing the sampling periods would bring

more samples in each optimization problem. More samples imply more variables and then a more complex optimization problem that would require more time to be solved. At the same time, it is required that the optimization problem is solved in the time between two sampling instants. It should be obvious, then, that the sampling period can be reduced as far as the resulting optimization problem can still be solved in no more than a ΔT.

3.3 Environment

The robots move in an environment populated with obstacles or unsafe zones that can be described by a convex linear set, that means they can be represented by set of linear inequalities of the form

$$OX \leq r \qquad X = \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.5}$$

where O is an $R \times 2$ matrix with R the number of linear constraints needed to define the obstacle. Even though it may seem restrictive the use of this obstacle representation, it is possible to describe a wide range of different situations. For example, it is possible to model linear nonconvex obstacles by composing convex sets or nonlinear obstacles using a linear approximation of the nonlinear obstacle. Examples are reported in Fig. 3.2.

Since the robot is considered to be a point mass, all the obstacles must be enlarged to consider the actual dimensions of the robot.

The obstacles must also be enlarged due to the discretization. In fact obstacle avoidance will be guaranteed only at the sampling time, so it would be possible to

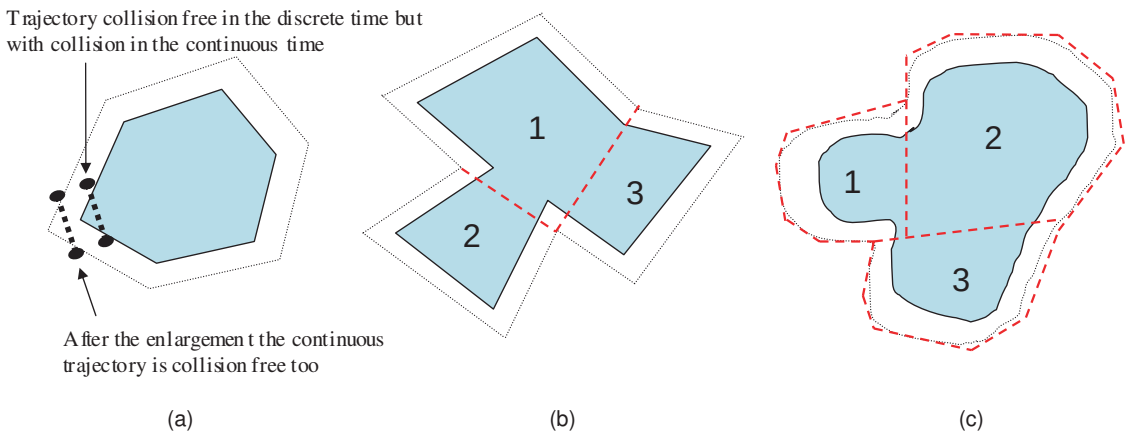


Figure 3.2: Obstacle representations: (a) A convex linear obstacle enlarged to consider the robot's dimensions; (b) A nonconvex linear obstacle obtained by composition of convex linear sets; (c) A nonconvex nonlinear obstacle represented using linear approximation and composition of convex linear sets.

have a trajectory that is collision free at the sampling time but that would end up in a collision during the time between the samplings. This is most likely to happen in the vicinity of a corner of an obstacle as shown in Fig. 3.2(a). The enlargement depends on the maximum possible distance D_{pp} between two points a robot can travel between samplings. D_{pp} is function of the maximum velocity of the robots V_{max}, and of the sampling period ΔT, in particular $D_{pp} = V_{max}\Delta T$. Once the obstacle is enlarged, the following implication needs to be true

$$\begin{aligned} &\forall x_1, x_2 \in \partial O_e \text{ such that } ||x_1 - x_2|| \leq D_{pp}, \\ &x = \lambda x_1 + (1-\lambda)x_2, \lambda \in [0,1], \quad x \notin O \end{aligned} \tag{3.6}$$

where O_e is the convex set describing the enlarged obstacle, ∂O_e is its boundary, while O is the convex set describing the original obstacle. As discussed before a shorter sampling period would imply a smaller enlargement and so a better performance.

3.4 Hierarchical control

Complex systems are often described by a wide number of variables that could either be continuous or discrete. In these situations it is often hard to solve an optimization problem taking into account all the decision variables. For this reason complex system are often split in hierarchical levels such that each level take decision only on a restricted set of variables, neglecting part of the decision variables and receiving the value of the others from the higher level in the hierarchy. An example of this hierarchical approach can be found in the manufacturing system control. In this case, several decisions need to be taken like for example the kind of products ordered, the quantity of products, the path that each piece needs to follow inside the manufacturing system, the scheduling on each machine and the management of the stock. It is clear that all this decision variables are somehow related and to find the global optimal solution for the production problem they should be considered in one big optimization problem. However, formulating the problem in this way would produce many problems. For example, the dimension of the formulation will become too big, and the time required to solve to problem too long. This makes the problem not solvable in a single stage. Instead of solving one single problem the decision variables are divided in different layer following some criteria. In the case of the manufacturing system is the rate of change of the input that affects the division of the decision variables. The hierarchical approach offers the possibility of obtaining smaller problems that can be easily solved and often the solution obtained it is not far from the optimal one.

In a more formal way, supposing to have an optimization problem that involves a

set of variables $X = \{x_1, x_2, \ldots, x_n\}$:

$$\begin{aligned} &\min J(x_1, \ldots, x_n) \\ &\text{s.t.} \\ &f(x_1, \ldots, x_n) = 0 \\ &g(x_1, \ldots, x_n) \leq 0 \end{aligned} \tag{3.7}$$

After defining a hierarchical criteria the set of variables is divided in several subsets:

$$X_1 = \{x_{11}, x_{12}, \ldots, x_{1n}\}, \ldots, X_m = \{x_{m1}, x_{m2}, \ldots, x_{mn}\}$$

Every subset defines a layer of the hierarchy, at each layer an optimization problem is defined and the others variables explicitly not considered in the problem are either neglected or considered constant and equal to the value found solving higher level problems. The optimization problem at each level is constrained by the constraints that involve only variables of that level. Some of the constraints may involve variables of higher levels, so it may be necessary, to satisfy the constraints, to include in the problem variables already considered at an higher level. The optimization problem in a general level could be formulated as follows:

$$\begin{aligned} &\min J_i(x_{i1}, \ldots, x_{in}, x_1^H, \ldots, x_k^H) \\ &\text{s.t} \\ &f(x_{i1}, \ldots, x_{in}, x_1^H, \ldots, x_k^H) = 0 \\ &g(x_{i1}, \ldots, x_{in}, x_1^H, \ldots, x_k^H) \leq 0 \\ &\{x_1^H, \ldots, x_k^H\} \subset \{X_1, \ldots, X_{i-1}\} \end{aligned} \tag{3.8}$$

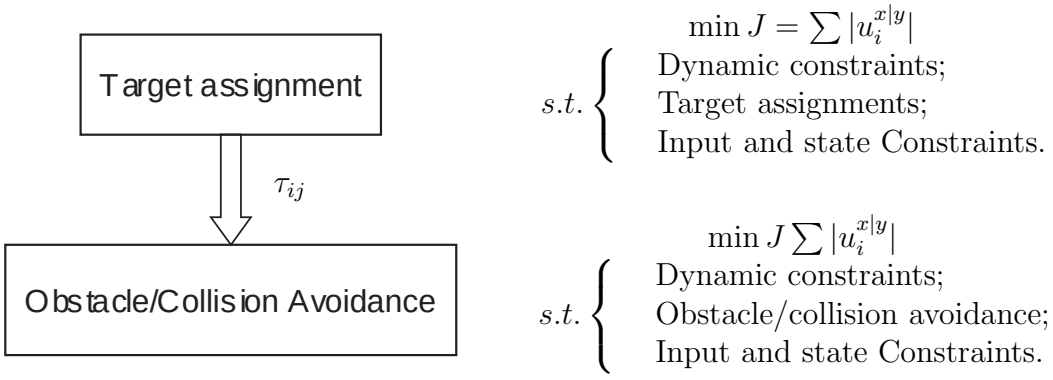


Figure 3.3: An example of hierarchical control

where x_i^H are the variables of the higher level that need to be reconsidered in the problem to meet the constraints neglected at the higher levels. It is clear that the set of variables x_i^H is as smaller as the hierarchy criteria used to divide the original set of variable is coherent with the constraints in the problem.

Applying this concept to the coordination control it is possible to generate a hierarchy dividing the variables into two groups: one group with the variables related to task involving the whole formation, like for example the target assignment or keeping a formation, and the second group of variable related to task involving the single robot, like the obstacle or collision avoidance. Fig. 3.3 indicates the scheme of the controller for the higher level solving the target assignment problem and the lower level solving the single robot tasks (collision/obstacle avoidance). In this case, the higher level takes decision only on the assignment neglecting the obstacle and collision avoidance. The lower level takes the variable τ_{ij} obtained from the higher level and solves an optimization problem taking into account obstacle and collision avoidance

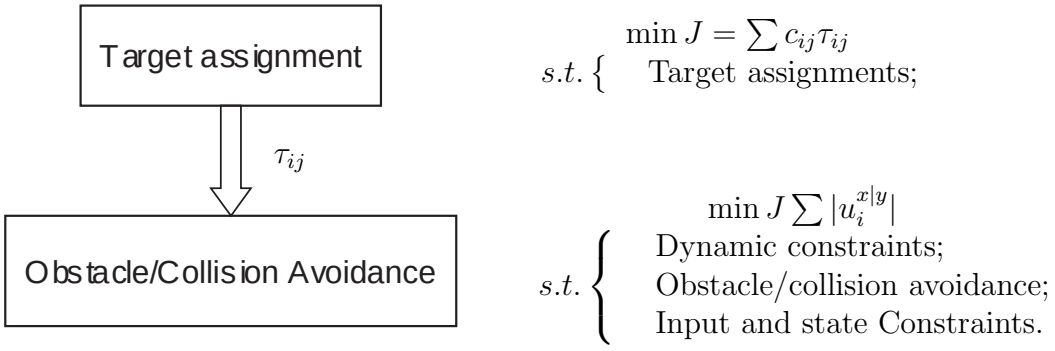


Figure 3.4: An example of hierarchical control

among the members of the formation. Notice that all the state variables and the input variables need to be recomputed to meet the obstacle and collision avoidance constraints that were neglected in the higher level. The convenience of such approach is that the binary variables in original problem are divided in two separate problems such that the computational power required to solve the two combined problems is less than the computational power originally required.

It has to be noted that when a hierarchical approach is used the global optimality of the solution is lost for having a groups of easier solvable problems. The same problem can be addressed using the scheme in Fig. 3.4. In this second case there is a complete division among the decision variable so that the set of variables x_i^H is empty. For this second approach becomes crucial the choice of the cost associated with the target assignment. An easy solution can be the distance between the robot and the target, while another, more complex but that could bring to a better optimum, is by using as a cost for τ_{ij} the total input effort it would be required to bring the robot i

to the target j. All these parameters make the problem easier or harder to solve but at the same time they bring the solution closer or farther from the global optimum.

3.5 Distributed and decentralized control

When the optimization problem involves different autonomous systems a possible way to simplify is to define one optimization problem for each system. Instead of solving a big problem that takes into account all the systems, several smaller problems are solved. For each system a subset of the agents, called neighbors is defined. The optimization problem will take into account only the neighbor systems so that the optimization problem is smaller, moreover the variables regarding the neighbors are considered constant. That means only decisions on the local system are taken in the optimization problem.

In a more formal way, suppose to have the following optimization problem involving n different agents:

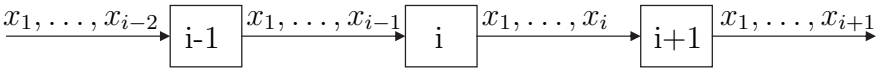
$$\begin{aligned} &\min J(x_{11}, \dots x_{1n_1}, \dots, x_{n1}, \dots, x_{nn_n}) \\ &\text{s.t.} \\ &f(x_{11}, \dots x_{1n_1}, \dots, x_{n1}, \dots, x_{nn_n}) = 0; \\ &g(x_{11}, \dots x_{1n_1}, \dots, x_{n1}, \dots, x_{nn_n}) \leq 0; \end{aligned} \tag{3.9}$$

and supposing that the i-th agent as the neighbors described by the following set of variable $\mathcal{N} = \{\mathbf{x}_1^N, \dots, \mathbf{x}_p^N\}$ where $\mathbf{x}_i^N$ is the vector containing the state variable of the i-th neighbor of the agent is taken into account. The optimization problem for

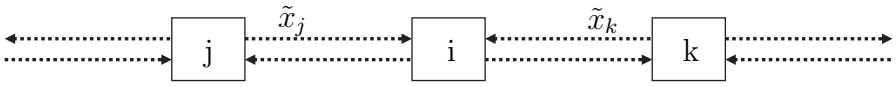
the local agent will be:

$$\begin{aligned} &\min J(x_{i1}, \dots x_{in_i}) \\ &\text{s.t.} \\ &f(x_{i1}, \dots x_{in_1}, \tilde{\mathbf{x}}_{\mathbf{1}}^{\mathbf{N}}, \dots, \tilde{\mathbf{x}}_{\mathbf{p}}^{\mathbf{N}}) = 0; \\ &g(x_{i1}, \dots x_{in_1}, \tilde{\mathbf{x}}_{\mathbf{1}}^{\mathbf{N}}, \dots, \tilde{\mathbf{x}}_{\mathbf{p}}^{\mathbf{N}}) \leq 0; \end{aligned} \tag{3.10}$$

where $\tilde{\mathbf{x}}_{\mathbf{i}}^{\mathbf{N}}$ can be either an estimation of the state of the i-th neighbor or it can be sensed or communicated.



(a) Distributed

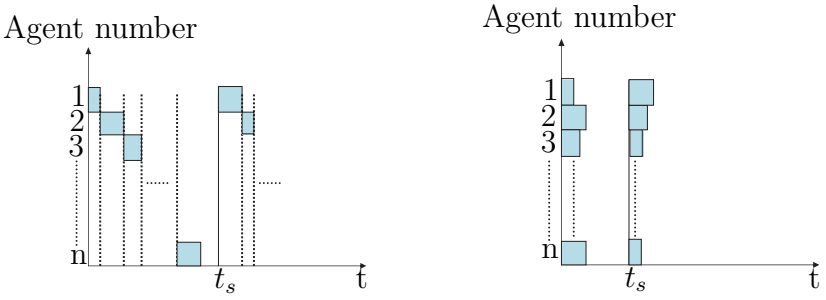


(b) Decentralized

Figure 3.5: Interaction among agents in distributed and decentralized control.

Depending on how the value for the $\tilde{\mathbf{x}}_{\mathbf{i}}^{\mathbf{N}}$ is fixed there are two possible different scenarios: distributed control or decentralized control. If the local agent solves its optimization problem and then communicates the solution to the others the control

algorithm is said to be distributed. Otherwise, if each agent simply senses or estimates the state of the neighbors then the control algorithm is said to be decentralized. The two algorithms require a different kind of synchronization among the agents. In case of distributed control it is necessary to define an order among the systems in order to obtain synchronization. In this way, the first agent could solve the optimization problem ignoring the others agents, then, once the optimization problem is solved it can communicate his solution to the second agent such that it can use the information in its optimization problem, and so on. Following this approach a chain is built in which each agent solves an optimization problem knowing the solution of all the agents that precede it in the chain. In the case of decentralized control instead, all the optimization problems are started at the same moment and the information about the neighbors variables are either sensed or estimated. In Fig. 3.5 is shown how the generic agent interact with its neighbors. In Fig. 3.5(a) is shown the communication chain among consecutive agents, while in Fig. 3.5(b) is shown the sensing among the agents.



(a) Distributed (b) Decentralized

Figure 3.6: Time comparison between the distributed and decentralized controller algorithm.

In both cases the global optimality is lost, but on the other hand the time required to solve the small problem is reduced compared to the time required for solving the

global problem. The difference between the two approaches, as shown in Fig 3.6, is that in the distributed approach before solving the $i-th$ problem it is necessary to wait that all the previous $i-1$ problem are solved. Instead, for the decentralized control, all the problem are solved at the same time. When the optimization problem is used in an online application it is clear that the optimization needs to end in the time constraint imposed by the sampling time. It is then clear that based on the problem the distributed approach could be not applicable. On the other side, the distributed approach could give better results in terms of the optimal value because the optimization problem are based on the true values for the state variables for the preceding agents while in the decentralized approach the state variables are only sensed or estimated. In both cases there could be problem of feasibility. In fact, in the distributed case a decision taken at some point in the chain ignoring the subsequent agent could make the problem for the following agents unfeasible. Instead, in the decentralized case, if the actual value for a variable is far away from the sensed or estimated value, feasible problem can arise. To avoid these situations more attention have to be placed on the constraints. A common drawback is that more conservative constraints bring the solution farther from the global one.

In the case of coordination control, for the distributed control algorithm there will be the first robot solving its optimization problem while ignoring the others robots, then it will communicate the solution to the second robot. The second robot will solve the optimization knowing where the first robot will be. Then it will transmit its and the first robot's solution to the third robot an so on down the chain. It is clear how proceeding down in the chain it is possible that some infeasibility problem may arise. In fact some robots may get blocked due to the decision taken from the previous one. In the same way in the decentralized algorithm, each robot solves its own optimization

problem sensing some information about its neighbor, and estimating others. If the sensed or estimated value is wrong it could cause an infeasibility problem.

A combination of hierarchical and decentralized optimization will be used to address the cooperative problem.

Chapter 4

Centralized control

4.1 Introduction

In this chapter the centralized approach to coordination control problem is presented. In this first formulation the open-loop optimization problem solved in the MPC algorithm takes into account all the constraints related to the coordination control in a single problem. This formulation is presented as a basic step to the hierarchical decentralized approach and for motivating the necessity for a different control algorithm.

4.2 Open-loop optimization problem

The coordination problem that will be solved consists of finding the control input for a formation of robot that has the task of reaching a set of targets while minimizing the input effort and avoiding obstacles and collisions among the member of the team.

Firstly, a logical formulation of the problem will be presented and then it will be shown how, using the **big M** technique, these constraints can be translated in mixed integer linear programming constraints.

The minimization problem that will be solved can be summarized as follow

$$\min \sum_{k=0}^{(T-1)} \sum_{i=0}^{(N_R)} |u_i^x(k)| + |u_i^y(k)|$$

subject to:

1. Dynamic equation constraint;
2. Obstacle avoidance constraint;
3. Collision avoidance constraint;
4. Target assignment constraint.

Among these constraints only the dynamic equations are already in linear form

$$X(k+1) = A\,X(k) + B\,U(k) \qquad \forall k = 0, \ldots, T-1 \tag{4.1}$$

for all the others it will be necessary to express them in a linear way. It is also necessary to write the cost function in a linear fashion. In the following subsections it will be shown how to treat each of the constraints, putting them in a mixed integer linear form.

4.2.1 The absolute value

The absolute value it is not a linear function so it cannot be included in a linear programming problem, but it is a composition of linear functions so it is possible to formulate the problem in a linear way using auxiliary variables.

Let define the auxiliary variable z_i^x as follows

$$\begin{aligned} z_{x_j}(k) &\geq u_{x_j}(k) \\ z_{x_j}(k) &\geq -u_{x_j}(k) \end{aligned} \tag{4.2}$$

the two inequalities in (4.2) model the absolute value of u_i^x. In fact, as it possible to see in Fig. 4.1, the constraints (4.2) force z_{x_j} in the region R and when the minimization over the sum of the z_{x_j} is done, it is guaranteed that the value obtained for z_{x_j} is on the region's R boundary, that is by construction the absolute value of u_{x_j}. The cost function will then become

$$\min \sum_{k=0}^{(T-1)} \sum_{i=0}^{(N_R)} z_i^x(k) + z_i^y(k)$$

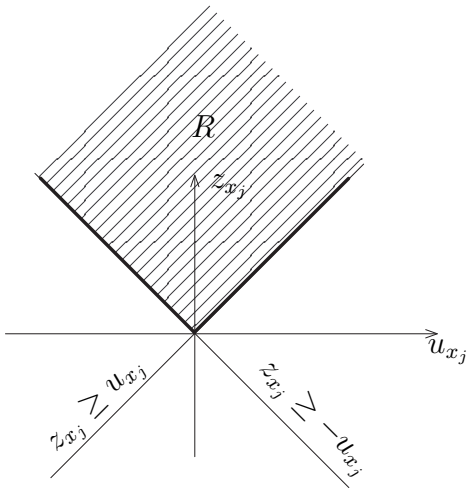


Figure 4.1: Admissible zone for z_{x_j} due to constraints (4.2)

4.2.2 Obstacle avoidance

If an obstacle has a convex shape and its borders are linear, all the points inside the obstacle could be represented as the set of points that satisfy a set of linear inequalities, in a general form this could be written as:

$$O \begin{pmatrix} x \\ y \end{pmatrix} \leq r \tag{4.3}$$

where O is a $2 \times P$ matrix and r is a $P \times 1$ vector with P the number of constraints necessary to define the obstacle.

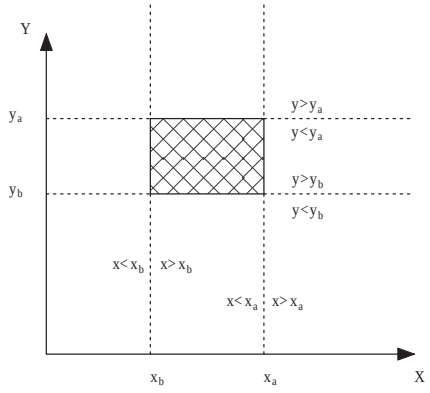


Figure 4.2: Obstacle's representation

Suppose to have an obstacle like the one shown in Fig. 4.2. The matrix describing the obstacle would be

$$\begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \leq \begin{pmatrix} x_a \\ -x_b \\ y_a \\ -y_b \end{pmatrix}$$

A binary auxiliary variable ω_{pi}^{k} is associated with each constraint that defines the

obstacle at every sampling time k such that the binary variable is one when the constraint is satisfied by the position of the robot $(x(k), y(k))$. All the points inside the obstacle are characterized by having all the P constraints that characterize the obstacle satisfied. In contrast, the points outside the obstacle are characterized by having at least one of the constraints violated. Then, considering the i-th robot at the k-th sample time, it is possible to write

$$o_{p1}x_i(k) + o_{p2}y_i(k) \leq r_p \Rightarrow \omega_{pi}^k = 1 \tag{4.4a}$$

$$\sum_{p=1}^{P} \omega_{pj}^k \leq P - 1 \tag{4.4b}$$

Equation (4.4a) drives the binary auxiliary variable ω_{pi}^k to 1 if the p-th constraint that describes the obstacle is satisfied by the position coordinates of the robot i at time k. The robot is inside the obstacle if all the P constraints that describe the obstacle are satisfied. The constraint in (4.4b) imposes that the coordinates of the robot violate at least one of the constraints that describe the obstacle. This ensures that the robot stays outside the obstacle. It is clearly necessary to have the same set of equations for all the obstacles, all the robots, and for all the sampling instants.

The implications in (4.4a) are translated in mixed integer linear inequalities using the **big M** technique, they can be written as:

$$o_{p1}x_i(k) + o_{p2}y_i(k) - r_p \geq \epsilon + (m - \epsilon)\omega_{pi}^k \tag{4.5}$$

while the inequalities in (4.4b) are already in linear form.

4.2.3 Collision avoidance

Collision avoidance could be seen as a special case of obstacle avoidance, in fact the others robots in the team can be seen as obstacles to be avoided. To implement the collision avoidance a safety zone around the robot is defined and the constraint that no robot can enter in the safety zone is added. In Fig. 4.3 it is reported a draw of the safety zone around the robot.

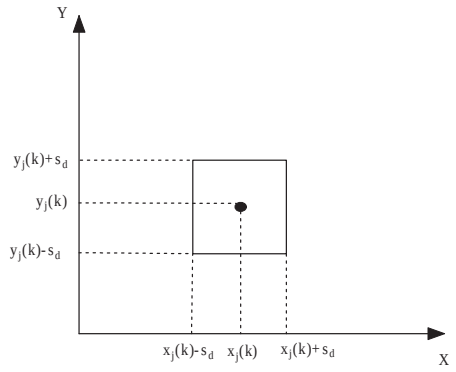


Figure 4.3: Robot's safe zone

If we select the j-th robot, its safe zone can be represented by the following set of inequalities

$$x \geq x_j(k) - s_d$$

$$x \leq x_j(k) + s_d$$

$$y \geq y_j(k) - s_d$$

$$y \leq y_j(k) + s_d$$

that could be written in the following matrix form

$$C_A \begin{pmatrix} x \\ y \end{pmatrix} \leq C_A \begin{pmatrix} x_j(k) \\ y_j(k) \end{pmatrix} + \mathbf{s}_d \tag{4.6}$$

where

$$C_A = \begin{pmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{pmatrix} \qquad \mathbf{s}_d = \begin{pmatrix} s_d \\ s_d \\ s_d \\ s_d \end{pmatrix}$$

In the same way it has been done for the obstacle avoidance, there will be an auxiliary binary variable $\tau_{ij}^p(k)$ for each inequalities that define the safety zone and the variable will be one if the inequality is satisfied. The collision avoidance will be obtained enforcing that the sum of the auxiliary variables $\tau_{ij}^p(k)$ is less then 3 (number of constraints that define the safety zone minus one). More formally the collision avoidance is model with the following implication:

$$\begin{gathered} C_A \begin{pmatrix} x_i(k) - x_j(k) \\ y_i(k) - y_j(k) \end{pmatrix} \leq \mathbf{s}_d \Rightarrow \tau_{ij}^p(k) = 1 \\ \sum_{p=1}^{4} \tau_{ij}^p(k) \leq 3 \end{gathered} \tag{4.7}$$

where $(x_i(k), y_i(k))$ and $(x_j(k), y_j(k))$ are the position of robot i and j at time k, τ_{ij}^p is an auxiliary binary variable which is 1, if the p-th inequality in the first equation in (4.7) is satisfied. The implications in (4.7) are converted in the same way shown for the obstacle avoidance. Note that imposing these constraint is equal to requesting that the distance between two robots is at least s_d in the x or y coordinate.

4.2.4 Target assignment

The target assignment consists of assigning one robot to one of the possible targets. The auxiliary variable γ_{ij} is one if the robot i is assigned to the target j. It is clear

that every robot has to be assigned to one target and each target needs to be assigned to one robot. From these two observations it is possible to write the first two sets of equations

$$\sum_{j=1}^{N_R} \gamma_{jl} = 1 \quad \forall l = 1, \ldots, N_T \tag{4.8}$$

$$\sum_{l=1}^{N_T} \gamma_{jl} = 1 \quad \forall j = 1, \ldots, N_R \tag{4.9}$$

where N_R and N_T are the number of robots and the number of targets.

It is now necessary to have an implication that drives the robot to the coordinates of the target that has the auxiliary variable equal to one. The implication can be written in the following way

$$\gamma_{jl} = 1 \quad \Rightarrow \quad \mathbf{X}_j(T) - \mathbf{X}_l^t = 0 \tag{4.10}$$

where $X_j(T)$ are the coordinates of the robot j at the end of the control horizon and X_l^t are the coordinates of the target l. The conversion of the implication in (4.10) is not straight forward as the one seen before. First of all, the equation is a vectorial equation, so it is necessary to split its x and y components. The new implications can be written in the following way

$$\begin{aligned} \gamma_{jl} = 1 \Rightarrow x_j(T) - x_l^t = 0 \\ \gamma_{jl} = 1 \Rightarrow y_j(T) - y_l^t = 0 \end{aligned} \tag{4.11}$$

To transform the first two implications in (4.11) it is necessary to add other auxiliary variables because there is no way to use the **big M** technique for implication that have the equality constraint. For this reason the implications needs to be divided in

two implications. The implications for the x component are

$$\begin{aligned} \gamma_{jl} = 1 \Rightarrow x_j(T) - x_l^t \geq 0 \\ \gamma_{jl} = 1 \Rightarrow x_j(T) - x_l^t \leq 0 \end{aligned} \tag{4.12}$$

A similar set of implications need to be written for the y component. At this point all the implications can be translated using the **big M** technique. The resulting set of inequalities are respectively

$$\begin{aligned} x_j(T) - x_l^t \geq m(1 - \gamma_{jl}) \\ x_j(T) - x_l^t \leq M(1 - \gamma_{jl}) \end{aligned} \tag{4.13}$$

A similar set of inequalities for the y component needs to be enforced as well

$$\begin{aligned} y_j(T) - y_l^t \geq m(1 - \gamma_{jl}) \\ y_j(T) - y_l^t \leq M(1 - \gamma_{jl}) \end{aligned} \tag{4.14}$$

4.2.5 The complete optimization problem

Putting together all the constraints, the optimization problem that will be solved is

$$\min \sum_{k=0}^{(T-1)} \sum_{i=1}^{(N_R)} z_i^x(k) + z_i^y(k) \tag{4.15a}$$

subject to the dynamic equations of the system

$$X_i(k+1) = A_i\, X_i(k) + B_i\, U_i(k) \qquad \forall k = 0, \ldots, T-1; i = 1, \ldots, N_R \tag{4.15b}$$

the equations that define $z_i^{x|y}$

$$\begin{aligned} & z_i^{x|y}(k) \geq u_i^{x|y}(k) \\ & z_i^{x|y}(k) \geq -u_i^{x|y}(k) \qquad \forall k = 0, \ldots, T-1; i = 1, \ldots, N_R \end{aligned} \tag{4.15c}$$

the obstacle avoidance constraints

$$\begin{aligned} & o_{p1} x_i(k) + o_{p2} y_i(k) - r_p \geq \epsilon + (m - \epsilon)\omega_{pi}^k \\ & \sum_{p=1}^{P} \omega_{pi}^k \leq P - 1 \qquad \forall k = 0, \ldots, T-1; i = 1, \ldots, N_R \end{aligned} \tag{4.15d}$$

the collision avoidance constraints

$$\begin{aligned} & ca_{p1}(x_i(k) - x_j(k)) + ca_{p2}(y_i(k) - y_j(k)) - s_d \geq \epsilon + (m - \epsilon)\tau_{pij}^k \\ & \sum_{p=1}^{4} \tau_{pij}^k \leq 3 \qquad \forall k = 0, \ldots, T-1; i = 1, \ldots, N_R; j = i+1, \ldots, N_R \end{aligned} \tag{4.15e}$$

the target assignment constraints

$$\begin{aligned} & \sum_{j=1}^{N_R} \gamma_{jl} = 1 && \forall l = 1, \ldots, N_T \\ & \sum_{l=1}^{N_T} \gamma_{jl} = 1 && \forall j = 1, \ldots, N_R \\ & x_i(T) - x_l^t \geq m(1 - \gamma_{jl}) \\ & x_i(T) - x_l^t \leq M(1 - \gamma_{jl}) \\ & y_i(T) - y_l^t \geq m(1 - \gamma_{jl}) \\ & y_i(T) - y_l^t \leq M(1 - \gamma_{jl}) \qquad \forall i = 1, \ldots, N_R; l = 1, \ldots, N_T \end{aligned} \tag{4.15f}$$

and the bound on the input amplitude and on the velocity

$$u_i^{x|y}(k) \leq U_{MAX} \qquad v_i^{x|y}(k) \leq V_{MAX} \qquad \forall i = 1, \ldots, N_R \tag{4.15g}$$

The problem as formulated requires $4N_RT$ continuous variables to describe the stare each robot during the all control horizon, $2N_RT$ variables for the input for each robot for each instant in the control horizon. The auxiliary variables $z_i^{x|y}$ will require other $2N_RT$. For the obstacle avoidance $N_RT\sum_{i+1}^{N_O} P_i$ binary variable are needed, and for collision avoidance $4T\sum_{i=1}^{N_R} i$ binary variable are requested. Finally, for the target assignment more N_RN_T are used.

On the other hand, from the point of view of the number of constraints present in the problem, the dynamic equations require $4N_RT$, the definition of $z_i^{x|y}$ need $2N_RT$ constraints, the obstacles avoidance requires $N_RT\sum_{i+1}^{N_O}(P_i + 1)$, the collision avoidance brings $5T\sum_{i=1}^{N_R} i$ constraints, the target assignment gives $4N_RT + N_R + N_T$ constraints, and finally the bounds on the input and velocity require the last $4N_RT$ constraints.

This analysis should give an idea of the dimensions of the optimization problem. In fact, already for small teams and easy environment (few obstacles) the number of variables and constraints can total thousands, making the time required to solve the optimization problem too extended. This is the principal motivation that has forced us to investigate the development of a different approach.

4.3 Implementation

To test the centralized algorithm a simulator has been developed. The simulator can be divided in three sections, the Matlab part, the interface Matlab/CPLEX and

CPLEX. The Matlab section is in charge of the simulation of the system and, moreover, the translation of the problem into the form

$$\begin{aligned} &\min f = c'X \\ &\text{s.t.} \\ &A_1 X = b_1 \\ &A_2 X \leq b_2 \end{aligned} \quad . \tag{4.16}$$

When the problem is defined, the number of team members and the number of obstacles is known. This can be used to compute the number of variables and constraints in the problem, which are equivalent respectively to the number of columns and rows of the matrices A_1 and A_2. The interface initially generate the two matrices just with zeros. The next step is to define the order of the optimization variable in the vector X. Based on that order, the matrices A_1 and A_2 have to be populated inserting the right values in the right position to represent the constraints defined in the problem. For example, defining the vector X as

$$\begin{aligned} X = [&x_1(1), v_1^x(1), y(1), v_1^y(1), u_1^x(0), u_1^y(0), \dots \\ &x_1(T), v_1^x(T), y(T), v_1^y(T), u_1^x(T-1), u_1^y(T-1), \dots]. \end{aligned} \tag{4.17}$$

Now let us consider the constraints

$$\begin{pmatrix} x_1(2) \\ v_1^x(2) \\ y_1(2) \\ v_1^y(2) \end{pmatrix} = A \begin{pmatrix} x_1(1) \\ v_1^x(1) \\ y_1(1) \\ v_1^y(1) \end{pmatrix} + B \begin{pmatrix} u_1^x(1) \\ u_1^y(1) \end{pmatrix}$$

that written in the standard form are

$$I_{4\times4}\begin{pmatrix} x_1(2) \\ v_1^x(2) \\ y_1(2) \\ v_1^y(2) \end{pmatrix} - A\begin{pmatrix} x_1(1) \\ v_1^x(1) \\ y_1(1) \\ v_1^y(1) \end{pmatrix} - B\begin{pmatrix} u_1^x(1) \\ u_1^y(1) \end{pmatrix} = 0_{4\times1},$$

which in terms of the vector X can be written as

$$[-A \;\; 0_{4\times2} \;\; I_{4\times4} \;\; -B \;\; 0_{4\times1} \;\; \dots \;\; 0_{4\times1}]X = 0_{4\times1} \tag{4.18}$$

the equation in (4.18) represent 4 rows of the matrix A_1 and the correspondent 4 rows of the vector b_1. A similar procedure is done for all the constraints in the problem. All this process is done by the first Matlab interface.

Once the problem is formulated as in (4.16) the Matlab interface for CPLEX [2] can be called. The interface take charge of all the procedures necessary to open the CPLEX environment populate the problem, start the optimization and retrieve the results. Based on the result given from CPLEX the status is updated and a new problem is formulated, and so on. A scheme of the algorithm is reported in Fig. 4.4.

4.4 Simulation results

To verify the algorithm several simulations have been run.

In Fig. 4.5 an example with three robots that try to reach three targets while avoiding three obstacles is presented.

Due to the complexity of the global optimization problem, the computational time required to solve each problem grows very fast and even for small team and

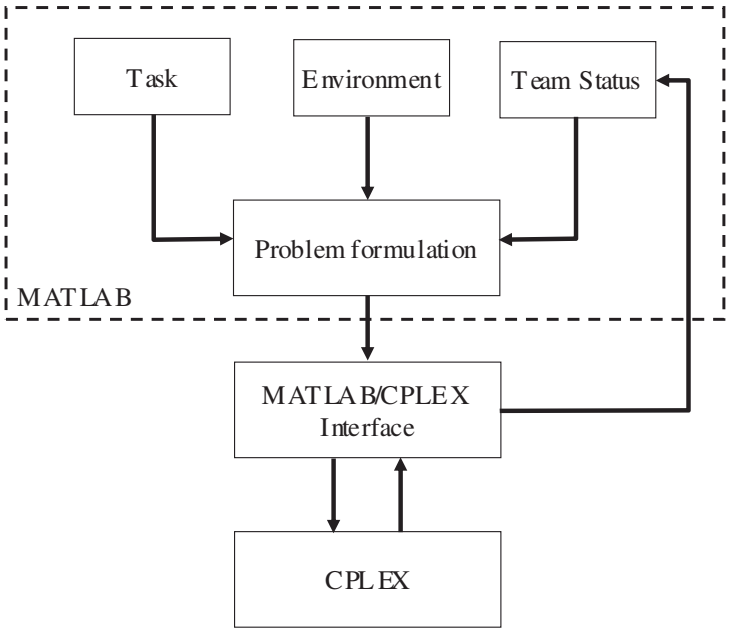


Figure 4.4: Algorithm scheme for the centralized controller.

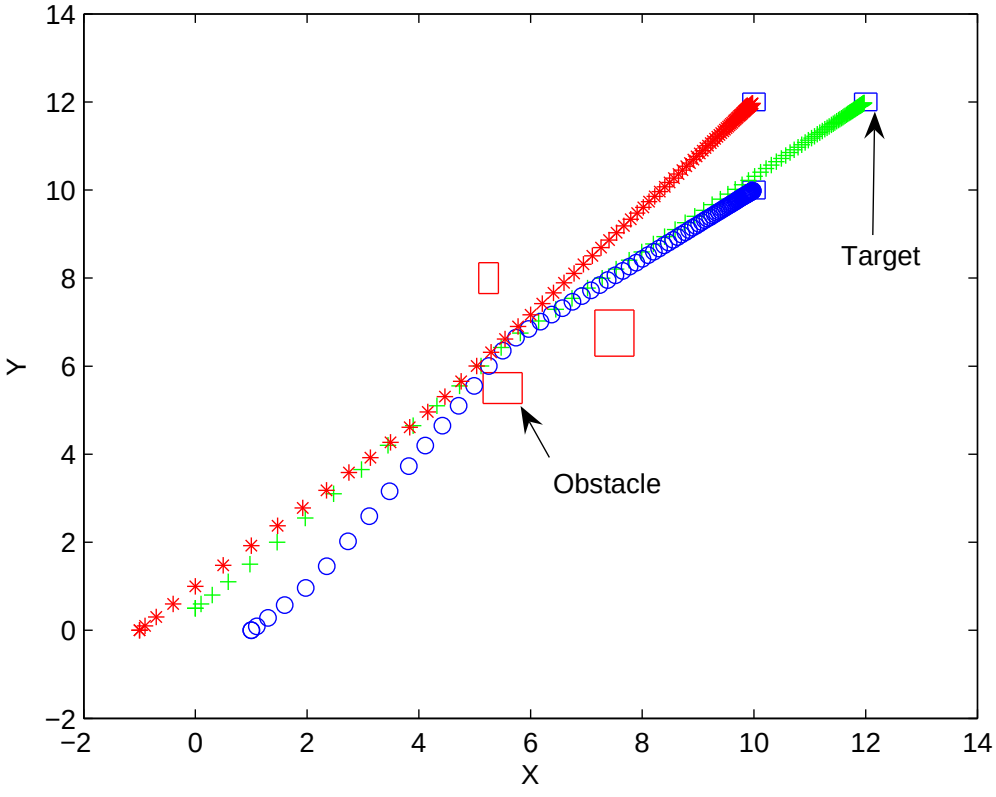


Figure 4.5: Three robots moving towards given targets while avoiding obstacles.

		# of robots				
		2	3	4	5	6
# of obstacles	1	0.125	0.484	3.938	18.31	290.1
	2	0.281	0.891	7.953	27.61	566.2
	3	0.406	2.016	20.05	203.9	1433

Table 4.1: Mean time to solve one optimization problem in the global formulation, function of the number of robots and the number of obstacles

easy environment like the one reported in Fig. 4.5 it may becomes difficult to meet the time constraint on the solution of the optimization problem imposed from the sampling time of the MPC algorithm. In Tab. 4.1 are reported the average time required to solve one single optimization problem in function of the number of robots and the numbers of obstacles. The problems refer to an environment such as that described in Fig. 4.5 with obstacles or robots added or deleted. It can be seen as moving at the bottom right side of the table, that correspond to larger team and more complex environment, the time required to solve the optimization problem becomes unacceptable. In particular considering a sampling time of 1 second the MPC could be implemented only at most for a formation of three robot moving in an environment with two obstacles. Obviously, this result is not satisfactory, because an algorithm able to deal with more complex situations is required. That is the motivation to investigate a different structures to address the coordination control, the structure presented in the next chapter aims to improve the scalability of the global algorithm such that the MPC algorithm with the underneath MILP optimization problem could be used for big formation moving in complex environment.

4.4.1 Alternative scenarios

To show the flexibility of the combination of the model predictive control some different scenarios will be presented. The first example comes from the idea of applying

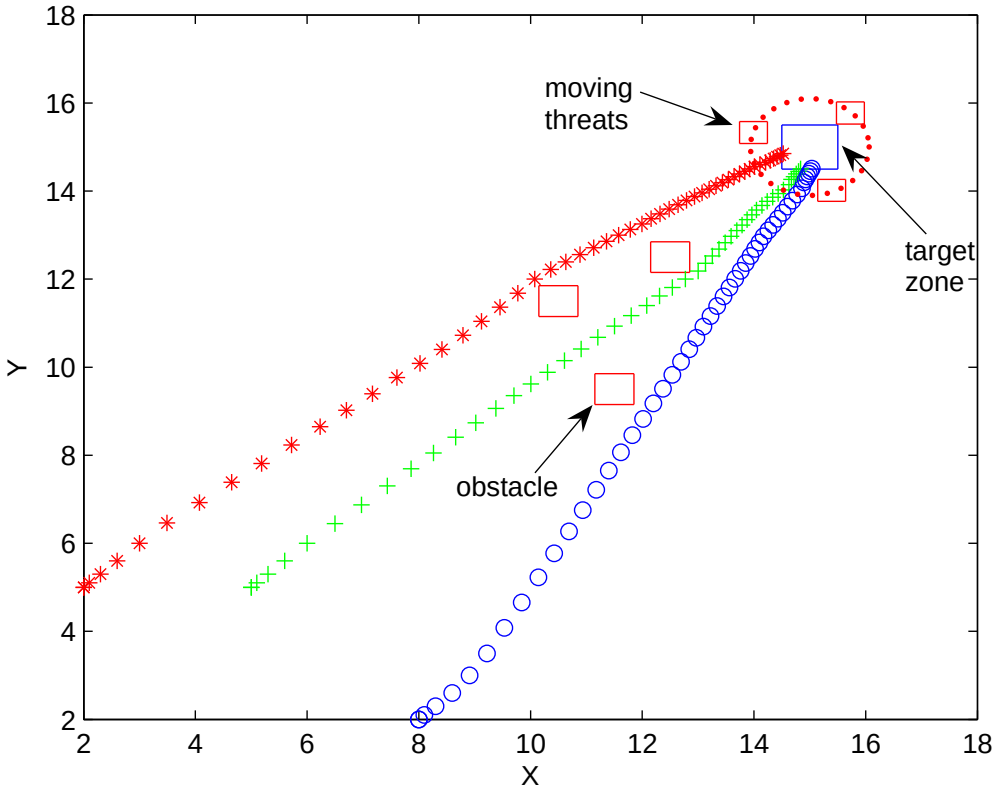
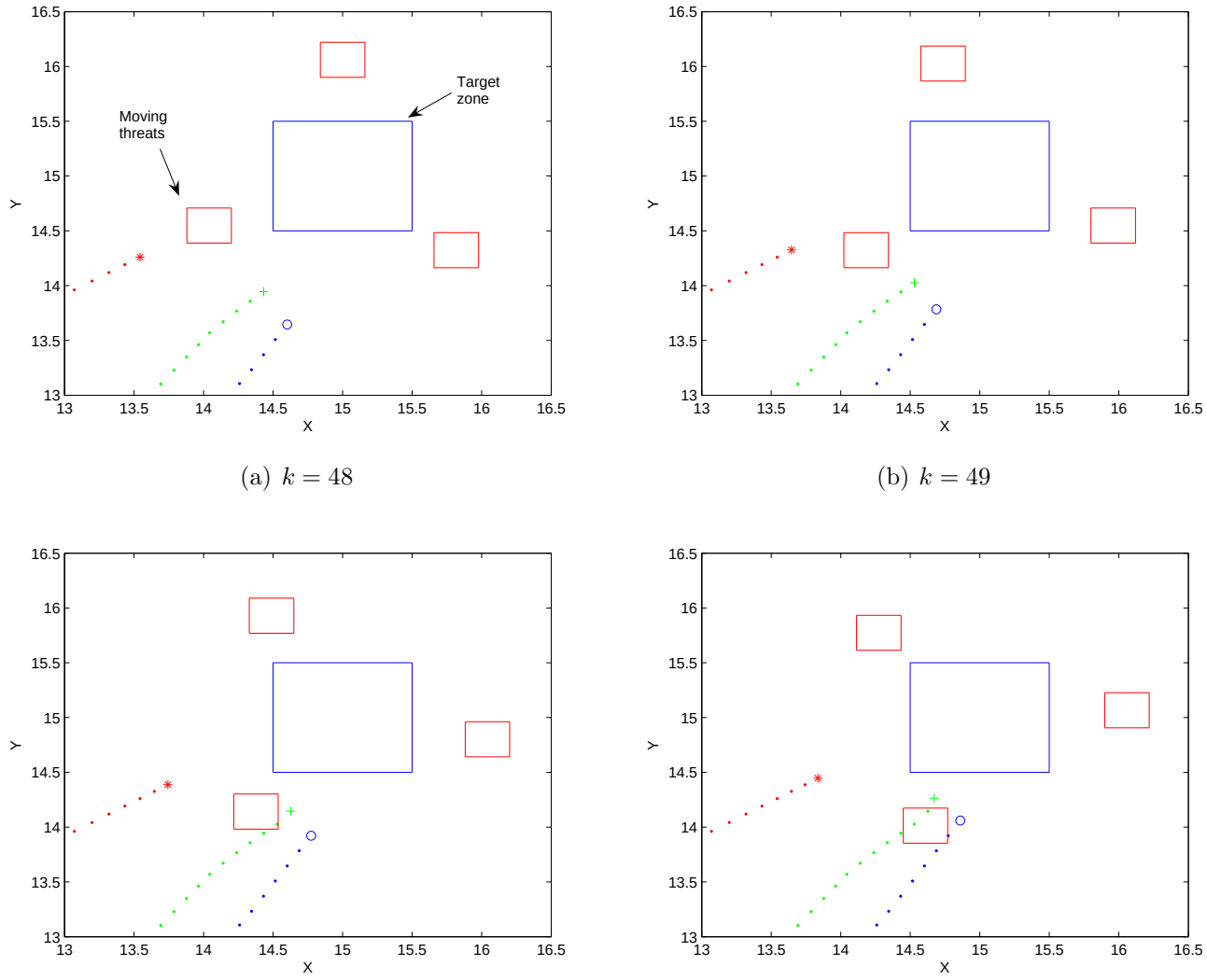


Figure 4.6: Three robots moving towards a target region while avoiding obstacles and moving threats.

the MPC/MILP to the RoboFlag [12]. The RoboFlag is a game in which two teams aim to steal the opponent's flag and bring it back to its own home zone. A robot trying to reach the flag can be tagged by opponents, in which case it must go back to the home zone before it can start aiming for the flag again. Part of the game consists, then, in trying to reach a zone defended by opponents avoiding them. Fig. 4.6 indicates the result of a simulation in which three robots are aiming to reach a zone defended by some moving obstacles. In the simulation it is supposed that the defenders rotate around the defended zone at constant angular speed. The result was obtained by some slight modification of the formulation presented. In particular the target assignment is substituted with a final constraint that impose to robots to be

in the defended zone, and the moving obstacles as treated as normal obstacle with the difference that the matrices $\mathbf{O}_t$ and the vectors r_t in (4.3) are known function of the time. As it can be seen in Fig. 4.7, robots are able to avoid the moving threats and successfully reach the target zone.



(a) $k = 48$ (b) $k = 49$

(c) $k = 50$ (d) $k = 51$

Figure 4.7: Robots are able to avoid moving obstacles and reach the goal zone.

Another interesting example is the case of moving targets. This can be thought as using the target assignment to keep a formation. Since the the optimization problem in the MPC is formulated again at every sampling time it is easy to introduce moving

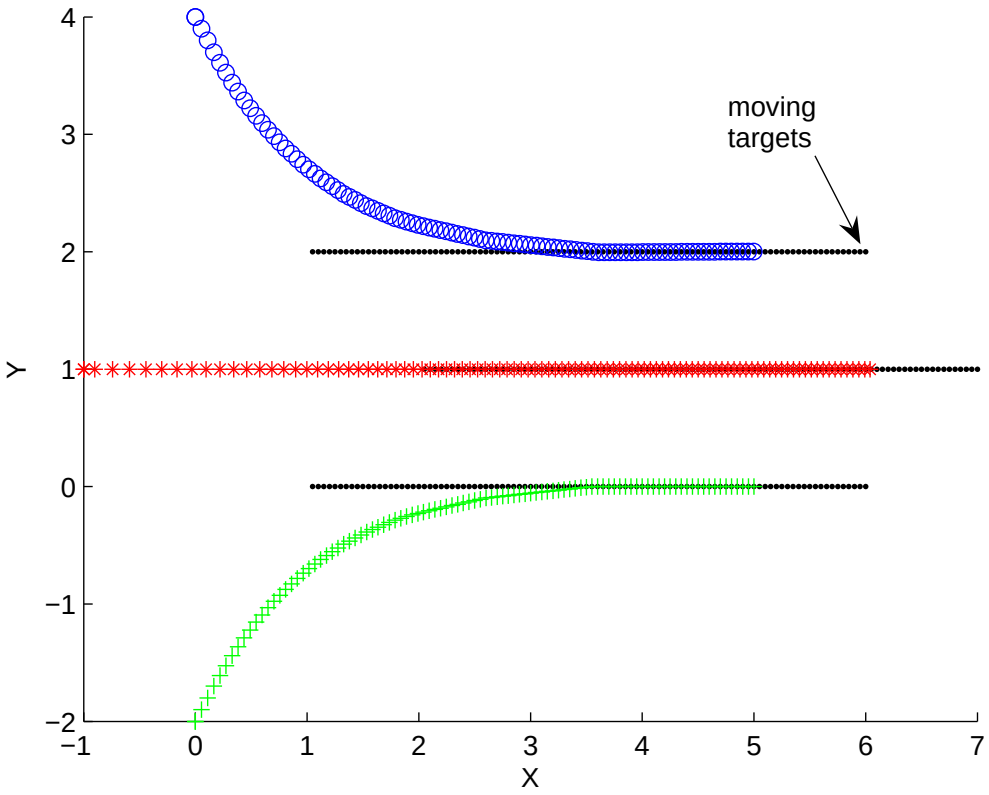


Figure 4.8: Moving targets.

targets to represent the desired movement of the formation. In Fig. 4.8 an example with three robots trying to reach a triangle shape formation is reported. An extension of the pervious example is obtaining by considering an environment with obstacles like the one in Fig. 4.9. It is interesting to see how, at the beginning, the team is trying to converge in the formation with the blue robot in the top position the red in the center position and the green in the bottom position, after avoiding the obstacle to optimize the team performance the MPC algorithm finds a new assignment with the green robot in the top position and the blue in the bottom. Note also the deviation of the red robot to avoid the green. For this example some observations are in order. It can be seen as there are some sampling times in which the desired target is inside the obstacle. Clearly, there can not be a feasible solution that guarantees at the same

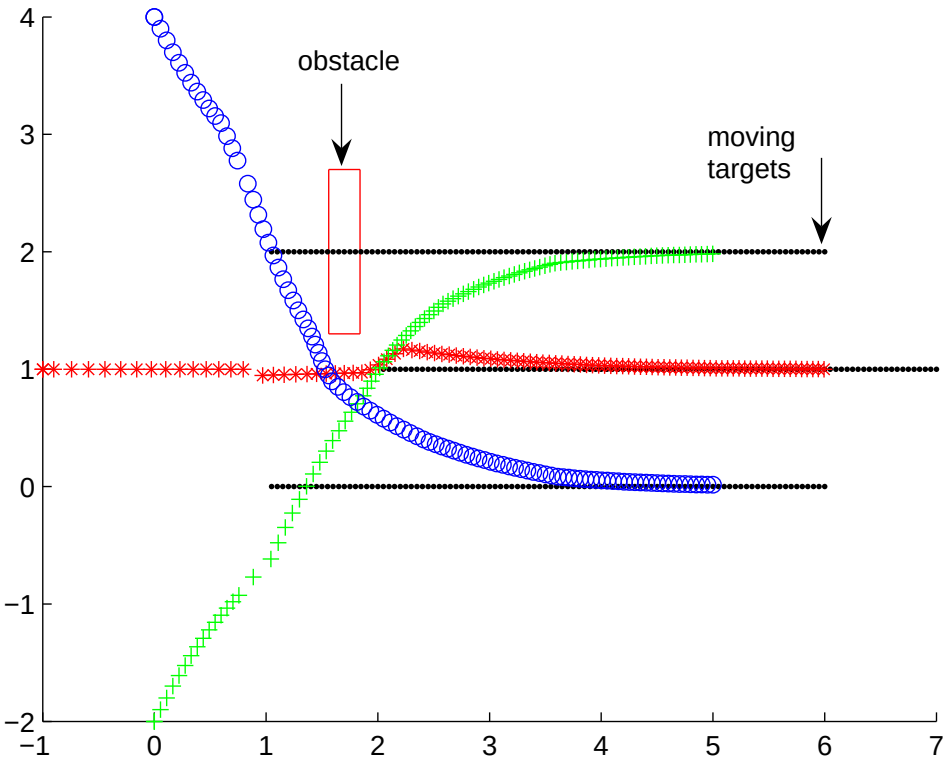


Figure 4.9: Moving targets with obstacles and dynamic reassignment.

time the satisfaction of obstacle avoidance and the final constraints since the target is inside the obstacle. For this reason the previous formulation must be corrected. In order to ensure the existence of a solution, the target assignment must be made a soft constraint. Soft constraints are a very useful tool for model predictive control algorithms. The soft constraints, contrarily to hard constraints that must be always satisfied by the solution, are constraints that can be violated by the solution but this violation comes with a penalty in the cost function.

In particular to make the target assignment a soft constraint, in the equations (4.15f) the equations that force to have each robot assigned to a target and each

target assigned to a robot must be modified in the following way

$$\begin{aligned} \sum_{j=1}^{N_R} \tau_{jl} + \zeta_l = 1 \quad l = 1, \ldots, N_T \\ \sum_{l=1}^{N_T} \tau_{jl} + \xi_j = 1 \quad j = 1, \ldots, N_R \end{aligned} \tag{4.19}$$

where ζ_l and ξ_j are auxiliary continuous variables, and the cost function must become

$$\min \sum_{k=1}^{T} \sum_{j=1}^{N_R} [z_j^x(k) + z_j^y(k)] + W_1 \sum_{i=1}^{N_T} \zeta_i + W_2 \sum_{j=1}^{N_R} \xi_j \tag{4.20}$$

where W_1 and W_2 are the cost penalty associated with the violation of the assignment constraint. To minimize the cost function the solver would try to keep the auxiliary variable zero, such that nothing is changed from the previous formulation. In case no feasible solution with a valid assignment is possible, the presence of the auxiliary variable allows to have a robot assigned to no target. Suppose no assignment is possible for the robot j then all the variables τ_{jl} will be zero and the auxiliary variable ξ_j will become 1, causing a penalty W_2 in the cost function.

These examples were reported to show the flexibility of the MPC/MILP combination in representing different kind of tasks/scenarios a property that has not been found in others approaches. In the next chapter will be described a new structure for the coordination problem that will add at the model flexibility offered from the MILP the scalability over the team size and the environment complexity that is at this point missing.

Chapter 5

Hierarchical decentralized control

The previous chapter has presented the coordination control problem in a global flat fashion. In other words it was formulated, and solved, a single problem including all the available information about the environment and the agents comprising the team. Solving this problem finds the best possible solution but, as shown before, this approach does not scale well on the number of agents in the team and on the complexity of the environment (number of obstacles). In fact, the formulation proposed has a number of binary variables that grows fast with the number of robots and obstacles and, as it is well known, the computation time required to solve a mixed integer linear problem grows exponentially with the number of binary variables. To obtain an algorithm that could be used for big formations and complex environments, it is necessary to divide the problem into several smaller problems in order to keep the number of binary variables in each problem from getting too large. In this way the global optimality of the solution is lost, but instead, a good and fast solution can be found.

5.1 Introduction

The multi-vehicle cooperative problem is divided both in a hierarchical way and a decentralized way. Because it is desired to accomplish tasks related to the whole team, a higher-level controller that knows the states of all the agents in the team is still present. The higher-level controller, considering the state of all the agents, makes decisions related to the whole team, for example the target assignment or keeping a formation, while will leave tasks that are related to each single robot to the lower-level controller, such as obstacle and collision avoidance. Instead of planning an off-line trajectory, the higher-level controller periodically reformulates and resolves the optimization problem to compensate for the unconsidered elements in the environment. In this way, if a robot, after avoiding an obstacle using the lower level controller, is closer to a different target from the one originally assigned, it is possible that the robot could be reassigned to this new target if this reassignment is more convenient for the formation.

On the other hand, decentralization is applied on the lower-level services. Every robot would solve its own optimization problem to find a safe trajectory. The optimization problem will consider only the closest obstacles and teammates. In fact, when a robot is navigating, the real danger comes only from the close obstacles and teammates, since all the others are out of the reachable space.

Considering only the closest dangers brings several advantages. First of all, because the number of binary variable is related to the number of obstacles and teammates considered, reducing the number of threats considered in the optimization problem would reduce these variables, making the problem easier to solve. Second, a sensing device (like an omnidirectional camera [5]) can be used instead of communication among teammates to estimate their state. Although, as a drawback the

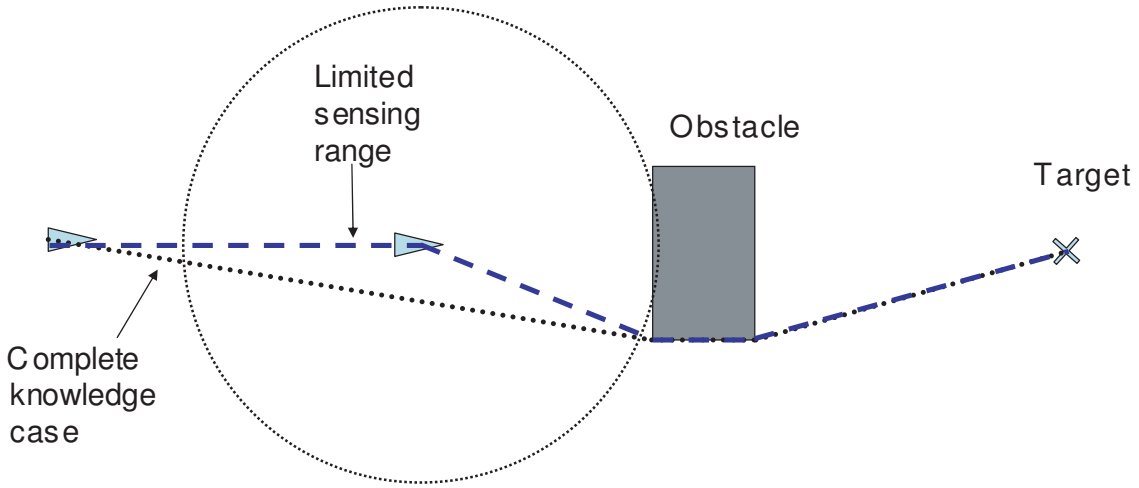


Figure 5.1: The difference in the trajectories in the case of complete knowledge and partial knowledge is one of the reasons the solution found with the decentralized method is worse than the one found with the global optimization.

solution obtained using a limited range of sense will be worse than the one found using complete knowledge, see Fig. 5.1.

In the following sections the higher-level and lower-level control problems are formalized. Then, some heuristics for improving the performance are introduced. The final part of the chapter is dedicated to an overview of the simulator and some simulation results.

5.2 Hierarchical decentralized algorithm

A schematic diagram of the algorithm is reported in Fig. 5.2. Both the levels are based on a model predictive control type of algorithm, with the difference that the higher-level controller has a slower sampling time. Roughly speaking, the higher-level controller solves the open-loop optimization problem to find the best assignment robot/target for all the robots in the formation. The task of the lower-level controller is to drive the robot to the assigned target while avoiding collisions with obstacles and teammates.

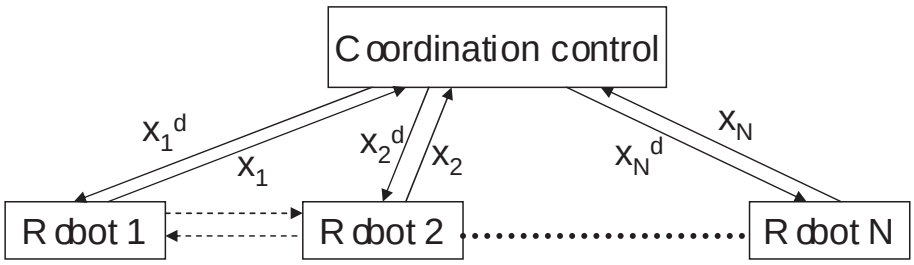


Figure 5.2: Control algorithm scheme.

5.2.1 Higher-level optimization problem

The higher-level model predictive control is in charge to find the best assignment robot/target based on the current status of the team. Thus, the higher-level control sends the assigned target to each robot.

The open-loop optimization problem solved by the higher-level MPC is formulated in the following way

$$\min \sum_{k=0}^{(T-1)} \sum_{i=1}^{(N_R)} z_i^x(k) + z_i^y(k) \tag{5.1a}$$

subject to:

$$\begin{aligned} & z_i^{x|y}(k) \geq u_i^{x|y}(k) \\ & z_i^{x|y}(k) \geq -u_i^{x|y}(k) \qquad \forall k = 0, \ldots, T-1; i = 1, \ldots, N_R \end{aligned} \tag{5.1b}$$

$$X_i(k+1) = A_i X_i(k) + B_i U_i(k) \qquad X_i(k) = \begin{bmatrix} x_i(k) \\ y_i(k) \\ v_i^x(k) \\ v_i^y(k) \end{bmatrix} \quad U_i(k) = \begin{bmatrix} u_i^x(k) \\ u_i^y(k) \end{bmatrix} \tag{5.1c}$$

$$\forall i = 1, \ldots, N_R; k = 1, \ldots, T;$$

$$\sum_{i=1}^{N_R} \gamma_{ij} = 1 \qquad \forall j = 1, \ldots, N_T \tag{5.1d}$$

$$\sum_{j=1}^{N_T} \gamma_{ij} = 1 \qquad \forall i = 1, \ldots, N_R \tag{5.1e}$$

$$\begin{aligned} x_i(T) - x_l^t &\geq m(1 - \gamma_{jl}) \\ x_i(T) - x_l^t &\leq M(1 - \gamma_{jl}) \\ y_i(T) - y_l^t &\geq m(1 - \gamma_{jl}) \\ y_i(T) - y_l^t &\leq M(1 - \gamma_{jl}) \qquad \forall i = 1, \ldots, N_R; l = 1, \ldots, N_T \end{aligned} \tag{5.1f}$$

$$u_i^{x|y}(k) \leq U_{MAX} \qquad v_i^{x|y}(k) \leq V_{MAX} \qquad \forall i = 1, \ldots, N_R; \forall k = 1, \ldots, T \tag{5.1g}$$

where the variable γ_{ij} is a binary variable that is one if the i-th robot is assigned to the j-th target and zero otherwise. As for the global case, equations (5.1b) define the variables $z_i^{x|y}(k)$ such that they are the absolute value of the variables $u_i^{x|y}(k)$. Equations (5.1d) ensure that each target is assigned to one robot. Equations (5.1e) ensure that one target is assigned to each robot. Equations (5.1f) force the position of the i-th to be equal to the position of the j-th target if $\gamma_{ij} = 1$. At the same time these equations act as a final set constraint and so, they also ensure stability.

This problem would then have $2N_RT$ continuous variables and $4N_RT$ constraints to represent the absolute value of $u_i^{x|y}$ (5.1b), $6N_RT$ continuous variables and $4N_RT$ constraint due to the dynamic equations (5.1c), and finally N_RN_T binary variables and N_T (5.1d) N_R (5.1e) and N_RN_T (5.1f) constraints for the target assignment. Since the problem so defined involves fewer binary variables, it will be easier to solve as shown in the simulation results.

5.2.2 Lower-level optimization problem

The lower-level optimization problem is in charge for finding the control inputs to drive the robot to the assigned target with a collision free trajectory.

Algorithm 3 Higher-level algorithm

1: **for all** Sampling time T_s^H **do**
2: Receive the state of each agent in the formation $X_i(k)$
3: Solve the optimization problem $\mathcal{P}^H(X(k))$ with $X(k)$ the state of the whole formation at k-th sampling time
4: Send to each agent the assigned target
5: **end for**

The optimization problem for the robot i can be formulated as

$$\mathcal{P}_i^L = \min \sum_{k=0}^{(T-1)} z_i^x(k) + z_i^y(k) \tag{5.2a}$$

subject to the equations that define $z_i^{x|y}(k)$

$$\begin{aligned} & z_i^{x|y}(k) \geq u_i^{x|y}(k) \\ & z_i^{x|y}(k) \geq -u_i^{x|y}(k) \qquad \forall k = 0, \ldots, T-1; \end{aligned} \tag{5.2b}$$

the dynamic equations of the robot

$$X_i(k+1) = A_i X_i(k) + B_i U_i(k) \qquad X_i(k) = \begin{bmatrix} x_i(k) \\ y_i(k) \\ v_i^x(k) \\ v_i^y(k) \end{bmatrix} \quad U_i(k) = \begin{bmatrix} u_i^x(k) \\ u_i^y(k) \end{bmatrix} \tag{5.2c}$$

$$\forall k = 0, \ldots, T-1;$$

the constraints for the obstacle avoidance

$$\begin{aligned} & o_{p1} x_i(k) + o_{p2} y_i(k) - r_p \geq \epsilon + (m - \epsilon)\omega_{pi}^k \\ & \sum_{p=1}^{P} \omega_{pi}^k \leq P - 1 \qquad \forall k = 0, \ldots, T-1 \end{aligned} \tag{5.2d}$$

the collision avoidance constraints

$$\begin{aligned} & ca_{p1}(x_i(k) - \hat{x}_j(k)) + ca_{p2}(y_i(k) - \hat{y}_j(k)) - s_d \geq \epsilon + (m - \epsilon)\tau_{pij}^k \\ & \sum_{p=1}^{4} \tau_{pij}^k \leq 3 \qquad \forall k = 0, \ldots, T-1; j = 1, \ldots, N_N \end{aligned} \tag{5.2e}$$

where N_N is the number of neighbors.

The terminal set constraints

$$x_i(T) = x_j^T, \quad y_i(T) = y_j^T \tag{5.2f}$$

and the bounds on the states and inputs

$$u_i^{x|y}(k) \leq U_{MAX} \qquad v_i^{x|y}(k) \leq V_{MAX} \qquad \forall k = 1, \ldots, T. \tag{5.2g}$$

Note that the constraints in (5.2d) cannot be written in the same way as it was written for the global case. In fact in the global case, the variables $x_j(k)$ and $y_j(k)$ were part of the optimization problem, instead in this case there is no knowledge about these variables. For this reason an estimation is used in place of the true value. Several ways are possible to obtain the estimation. One could be by direct communication, in this case every robot have to broadcast its plan to all the neighbors. This solution would require a complex structure for the synchronization among the teammates. A second way is by using a sensing device, like an omnidirectional camera. It is then supposed that each robot has the ability to sense the position and the velocity of its neighbor teammates. Using this knowledge, the estimation on $x_j(k)$ and $y_j(k)$ is made supposing that the other teammates will not change their velocity. The estimation

will be then

$$\hat{x}_j(k) = x_j(0) + kv_{xj}(0) \tag{5.3}$$

where the same estimation is made on the y direction. Since most likely the robot j would change its velocity to ensure a collision free solution, all the reachable points from robot j will be made unaccessible to robot i. This is equivalent to enlarge the safety zone around the robot j to $s_d = V_{max}T_s^L$, where T_s^L is the lower-level sampling time. This is a very conservative approach but it is necessary to ensure collision avoidance.

The algorithm executed by the lower-level controller is described in Algorithm 4.

Algorithm 4 Lower-level algorithm

1: **for all** Sampling time T_s^L **do**
2: Sample the state of the agent to obtain $X_i(k)$
3: Build the set of local constraints $\mathcal{C}_i^L$, and local variables $\mathcal{X}_i^L$
4: Solve the optimization problem $\mathcal{P}_i^L(X(k))$
5: Apply the first input from the input sequence obtained from the solution of the optimization problem
6: **end for**

The variables and constraints needed in the problem are: $2T$ continuous variables and $4T$ constraints for the definition of $z_i^{x|y}(k)$ (5.2b); $6T$ continuous variables and $4T$ constraints for the dynamic equations (5.2c); $\sum_i^{N_o} P_i T$ binary variables and $\sum_i^{N_o}(P_i + 1)T$ constraints due to the obstacle avoidance (5.2d), note that N_o is the number of obstacles considered; $4TN_N$ binary variables and $5TN_N$ constraints for the collision avoidance (5.2e). The dimension of the problem is much smaller than the global one, and the number of binary variables depends on the number of obstacles considered N_o and the number of neighbors N_N. Next section will described how some heuristic can be used to define the optimization problem in a more efficient way.

5.3 Heuristic for defining the lower-level optimization problem

The optimization problem $\mathcal{P}_i^L$ is defined through the set of its optimization variables $\mathcal{X}_i^L$ and the set of constraints $\mathcal{C}_i^L$. The main objective is to reduce the size of the two sets as much as possible in order to reduce the computation time required to solve the problem. At the same time, it is desirable to keep the solution collision free and as close as possible to the global optimum.

Among all the variables and constraints in the problem, there is a subset of variables $\mathcal{X}_i^n \subset \mathcal{X}_i^L$ and constraints $\mathcal{C}_i^n \subset \mathcal{C}_i^L$ that are necessary. These are the variables and the constraints due to the definition of $z_i^{x|y}(k)$ (5.2b), the dynamic equations of the robot (5.2c) and the bounds on the state variables and on the inputs (5.1g).

On the other hand, some decisions can be taken on the sets of variables and constraints related to the obstacles and collisions avoidance. Let us define the set of variables and constraints needed to include an obstacle $\mathbf{O}_t$ respectively

$$\mathcal{X}_t^o \qquad \text{and} \qquad \mathcal{C}_t^o \tag{5.4}$$

In the same way, the variables and constraints for including the robot R_j will be called

$$\mathcal{X}_j^r \qquad \text{and} \qquad \mathcal{C}_j^r \tag{5.5}$$

One way to build the two sets $\mathcal{X}_i^L$ and $\mathcal{C}_i^L$ is by including all the obstacles and all

the robots. This means the two sets are given by

$$\begin{aligned} \mathcal{X}_i^L &= \mathcal{X}_i^n \bigcup_{t=1}^{N_o} \mathcal{X}_t^o \bigcup_{j=1}^{N_R} \mathcal{C}_j^r \\ \mathcal{C}_i^L &= \mathcal{C}_i^n \bigcup_{t=1}^{N_o} \mathcal{C}_t^o \bigcup_{j=1}^{N_R} \mathcal{C}_j^r. \end{aligned} \tag{5.6}$$

Note that, this is equivalent to have complete knowledge of the environment and the state of all the robots in the team. Even though, the optimization problem defined in this way would have a solution close to the global optimum, it also has several drawbacks. First of all, the two sets $\mathcal{X}_i^L$ and $\mathcal{C}_i^L$ are the largest possible and, therefore, the solution to the problem would require much time. Moreover, it is not always possible to have complete knowledge of the environment where the robots are going to operate in. At the same time, this solution would require the knowledge about position and velocity of all the members in the team. This requirement would need communication among teammates but this communication may be undesirable.

An alternative is to consider in the sets $\mathcal{X}_i^L$ and $\mathcal{C}_i^L$ only the robots and the obstacles that can be sensed, see Fig 5.3. Following this idea the variables and the constraints for the obstacle $\mathbf{O}_t$ will be included if

$$\mathcal{X}_t^o \in \mathcal{X}_i^L;\ \mathcal{C}_t^o \in \mathcal{C}_i^L \Leftrightarrow \min_{(x,y)\in\mathbf{O}_t} d([x_i(k), y_i(k)], [x, y]) \leq d_{sensed}. \tag{5.7}$$

In the same way the variables and the constraints for the collision avoidance with the robot $\mathbf{R}_j$ will be included if

$$\mathcal{X}_j^r \in \mathcal{X}_i^L;\ \mathcal{C}_j^r \in \mathcal{C}_i^L \Leftrightarrow d([x_i(k), y_i(k)], [x_j(k), y_j(k)]) \leq d_{sensed}. \tag{5.8}$$

An assumption on the sensed range is in order. To guarantee the existence of a

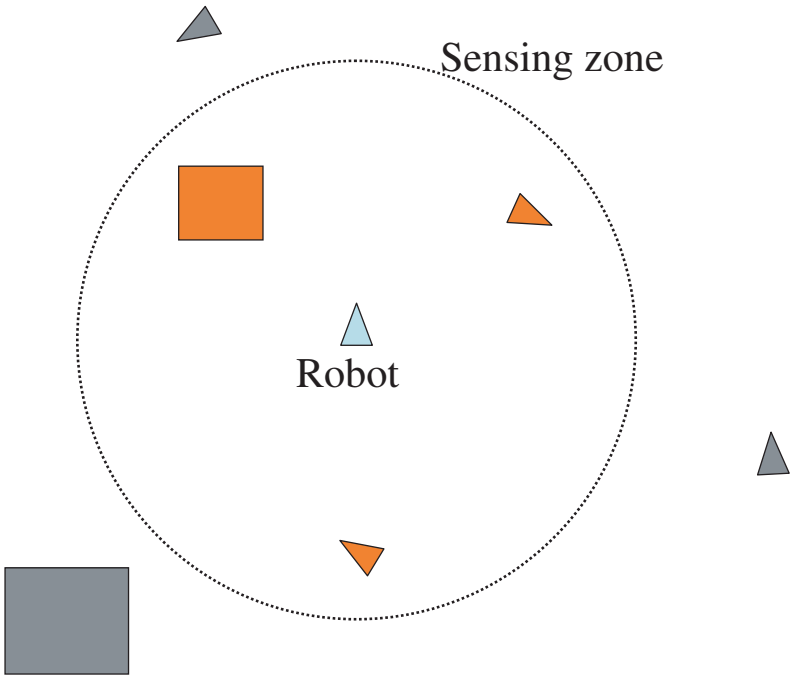


Figure 5.3: The blue robot can sense only the obstacles and the robots inside its sensing zone (in red), everything outside this zone is unknown (in gray).

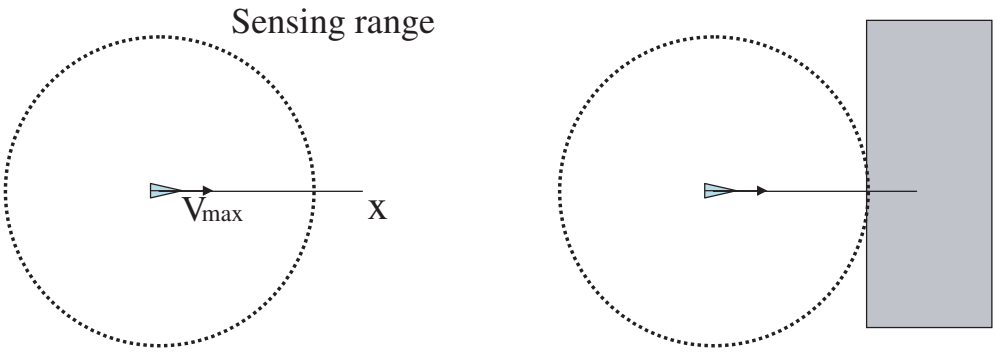
collision free trajectory, the sensing range must be large enough to allow the robot to stop before hitting the obstacles, once the obstacle is sensed and while moving at the maximum speed. This means that in the time t_s required to the robot to stop

$$\begin{aligned} V_{MAX} - t_s U_{MAX} &= 0 \\ t_s &= \frac{V_{MAX}}{U_{MAX}} \end{aligned} \tag{5.9}$$

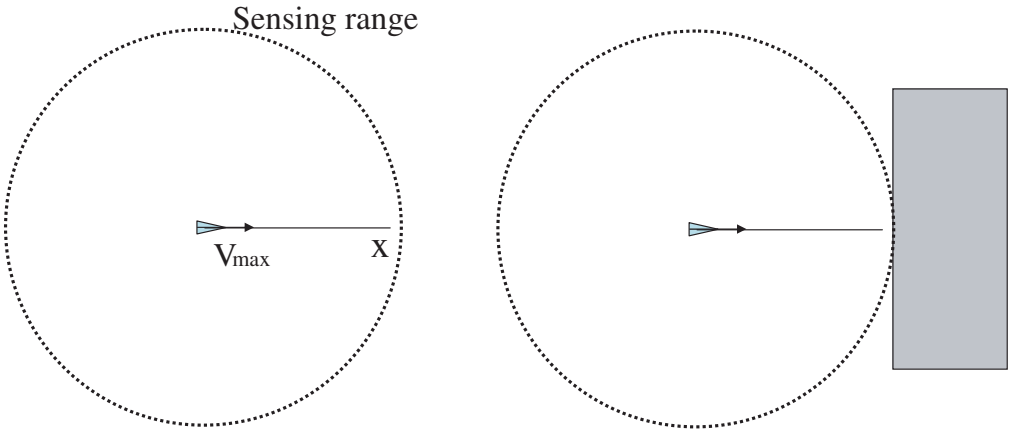
the distance covered x should be less the d_{sensed}

$$\begin{aligned} x &= V_{MAX} t_s - U_{MAX} \frac{t_s^2}{2} \leq d_{sensed} \\ x &= \frac{1}{2} \frac{V_{MAX}^2}{U_{MAX}} \leq d_{sensed} \end{aligned} \tag{5.10}$$

Using this heuristic the number of obstacles and teammates considered in each



(a) The sensing range is not big enough to guarantee obstacle avoidance.



(b) The robot can stop before hitting the obstacle.

Figure 5.4: Two situations in which depending on the sensing range the collision can be avoided or not. Note x is the minimum distance required to stop the robot while moving at the maximum speed.

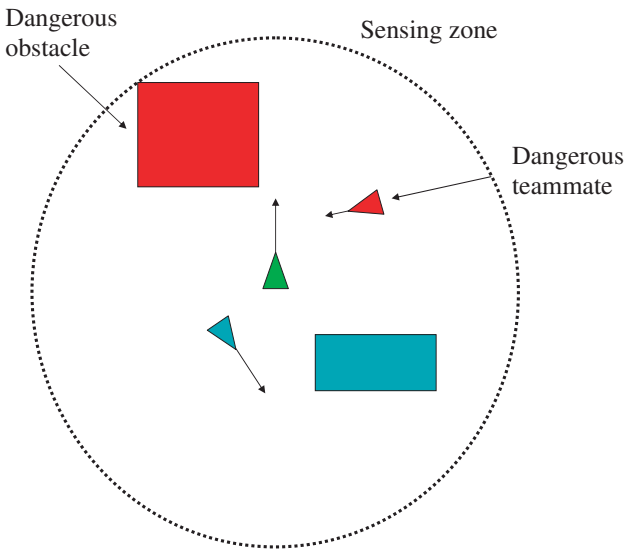


Figure 5.5: For the green robot only the red obstacle and robot represent a possible collision.

lower-level optimization problem are clearly reduced. Since the neglected obstacles and teammates are far, there is no risk of collision.

The result obtained with this first heuristic can be further improved. In fact, even though obstacles or teammates are in the sensing range they could not represent a risk. Let us consider for example a robot j in the robot i sensing zone, if j is behind i and moving in the opposite direction it is not a threat for i. In the same way, if and obstacle is in the i sensing range but it is behind the robot, with the robot moving away from the obstacle, there is no need to include the obstacle in the optimization problem. See Fig. 5.5. Considering the actual velocity and the bound on the acceleration, it is possible to build the cone of possible points in which every robot could be in the next sampling times, before actually computing the input from the optimization problem. The cone can be used as an estimation of where the robot will be in the next sampling time. The idea is then to use this cone to decide which robot and which obstacle are effectively dangerous and need to be included in the

optimization problem.

Given the velocity of the i-th robot at the k-th sampling time $(V_x(k), V_y(k))$ the maximum variation possible in the sampling time is $\pm U_{max} T_s$ for each component. The following four vectors can be obtained

$$\begin{aligned} v_1 &= \begin{pmatrix} V_x + U_{max} T_s \\ V_y + U_{max} T_s \end{pmatrix} \quad v_2 = \begin{pmatrix} V_x + U_{max} T_s \\ V_y - U_{max} T_s \end{pmatrix} \\ v_3 &= \begin{pmatrix} V_x - U_{max} T_s \\ V_y + U_{max} T_s \end{pmatrix} \quad v_4 = \begin{pmatrix} V_x - U_{max} T_s \\ V_y - U_{max} T_s \end{pmatrix} \end{aligned} \tag{5.11}$$

Among these, two of them v_i and v_j can be obtained as cone combination of the other two v_k and v_p, the latter two will be taken to generate the cone that will be used as estimation of the future position of the robot. More specifically the cone is defined as

$$\mathbf{C}_i = \{(x, y) \in \mathbb{R}^2 : (x, y) = \lambda_k v_k + \lambda_p v_p \quad \lambda_k, \lambda_p \geq 0\} \tag{5.12}$$

Once the cone $\mathbf{C}$ is obtained the intersection between the cone and the obstacle $\mathbf{O}_t$ can be taken

$$\mathbf{I}_o = \mathbf{O}_t \bigcap \mathbf{C}_i \tag{5.13}$$

if the intersection is empty the obstacle is not on the way then it can be omitted. If it is not empty, the obstacle will be included if the distance between the closest intersection point and the position of the robot i is less than a threshold t. This can

be summarized in the following implication

$$\begin{gathered} \mathcal{C}_t^o \in \mathcal{C}_i^L \, and \, \mathcal{X}_t^o \in \mathcal{X}_i^L \\ \Updownarrow \\ \mathbf{I}_o \neq \emptyset \bigwedge \min_{(x,y)\in \mathbf{I}_o} d([x_i(k), y_i(k)], [x, y]) \leq t \end{gathered} \tag{5.14}$$

In a similar way to decide if the robot j must to be included in the optimization problem of the robot i the intersection $\mathbf{I}_r$ between the cone $\mathbf{C}_i$ and $\mathbf{C}_j$ is computed

$$\mathbf{I}_r = \mathbf{C}_i \bigcap \mathbf{C}_j \tag{5.15}$$

if the intersection is empty the robot j is not on the way then it can be omitted. If it is not empty, the robot will be included if the distance between the closest intersection point and the position of the robot i is less then a threshold t. In summary

$$\begin{gathered} \mathcal{C}_j^r \in \mathcal{C}_i^L \, and \, \mathcal{X}_j^r \in \mathcal{X}_i^L \\ \Updownarrow \\ \mathbf{I}_r \neq \emptyset \bigwedge \min_{(x,y)\in \mathbf{I}_r} d([x_i(k), y_i(k)], [x, y]) \leq t \end{gathered} \tag{5.16}$$

In Fig. 5.6 is reported an example about the use of the cones.

5.4 Implementation

To implement the algorithm in real time, it would be necessary to have one solver for the higher-level controller and one solver for each lower-level controller. To verify the algorithm, a Matlab simulator has been developed. Since we had only one solver available the solution of the optimization problem is done sequentially instead of

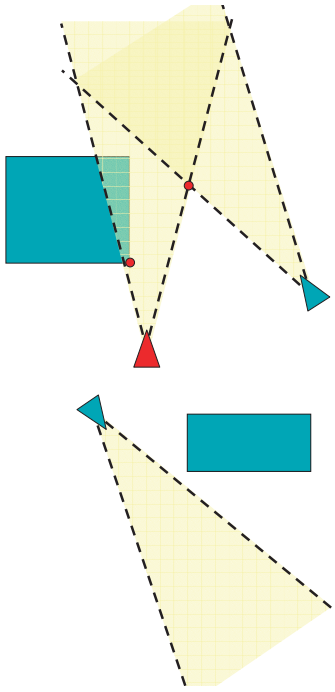


Figure 5.6: Among the sensed obstacles and teammates by the red robot only a subset has non empty intersection with its cone. Note that the red dots are the closer point of the intersections.

synchronously. In particular, firstly, the higher-level optimization problem is solved based on the initial condition of the team. Then based on the solution obtained from the higher-level, the lower-level problem are formulated and solved for all the robot. To recover the synchronization the inputs obtained by solving the lower-level problem are applied at the same time once all the problem are solved. The lower-level optimization is solved T_s^H/T_s^L times (T_s^H and T_s^L are chosen such that the division is an integer number). Then the higher-level optimization problem is formulated again based on the new state of the team. The simulator works following the Algorithm 5.

Going into the details of the Algorithm 5, after sampling the state of the team, the function high_level_problem is called. The function takes as input the position of the target and the state of the team and gives back the optimal assignment. The

Algorithm 5 Simulator algorithm

```
1: for all Sampling time t do
2:    Sample the state of the team to obtain X(t)
3:    Build the problem P^H(X(t))
4:    Find the optimal assignment
5:    for k = 1 to T_s^H/T_s^L do
6:       for i = 1 to N_R do
7:          Determine the robots and the obstacles in the sensing range of R_i
8:          Find the dangerous teammates in the sensing zone
9:          Find the dangerous obstacles in the sensing zone
10:         Build the problem P_i^L(X(t + kT_s^L))
11:         Find and store the solution of the optimization problem u_i^{x|y}(t + kT_s^L)
12:      end for
13:      Apply the input U^{x|y}(t + kT_s^L) to all the robots
14:   end for
15: end for
```

function is in charge of translating the constraints in the standard form

$$\begin{aligned} &\min f = c'X \\ &\text{s.t.} \\ &A_1 X = b_1 \\ &A_2 X \leq b_2 \end{aligned} \tag{5.17}$$

after the conversion the function will call CPLEX through the interface, and it will return the optimum assignment.

In the inner loop of the simulation, for each robot, a function that emulates the sensor will be called. This function is in charge of determining which robots among the team and obstacles are in the sensing range. To distinguish which robots and obstacles represent a possible collision the functions neighbors_on_the_way and obstacles_on_the_way are called. These two functions use the heuristic presented in the last section to reduce the number of robots and obstacles included in the optimization problem. The two functions follow the procedure described in the previous section,

in particular they will build the cone $\mathbf{C}_i$ and find the intersection with the cone of the others robots $\mathbf{C}_j$ and obstacles $\mathbf{O}_t$.

Once the problem is defined the function lower_level_problem is called to perform the translation in the standard form (5.17), and to call CPLEX.

5.5 Simulation results

To test the algorithm some simulations have been run.

In Fig. 5.7 is reported a simulation with six robots moving to reach six targets in an environment with several obstacles. Several aspects can be pointed out from the simulation. Obstacles with different shapes are included, and also a non-convex obstacle obtained from the combination of two convex obstacles is considered. Note also the trajectory of the yellow robot, it initially moves straight toward the target since it has no knowledge of the presence of the obstacle. When the obstacle is sensed it deviates from the trajectory to avoid the obstacle and then it moves back to reach the target when the obstacle has been passed.

This example could not be simulated with the centralized algorithm due to the high number of variables needed, it is instead easily handled from the hierarchical/decentralized algorithm. To have a quantitative comparison the algorithm is run over the same problem used to obtain Tab. 4.1. Running the same examples with the hierarchical decentralized algorithm the Tab. 5.1.

From the table it is possible to see how the time required to solve the problem does not increase too much when moving in the bottom right part of the table that correspond to the hardest problems. In particular it can be noted that all the scenarios in the table could be implemented with a sampling time of 0.5 s. Comparing the

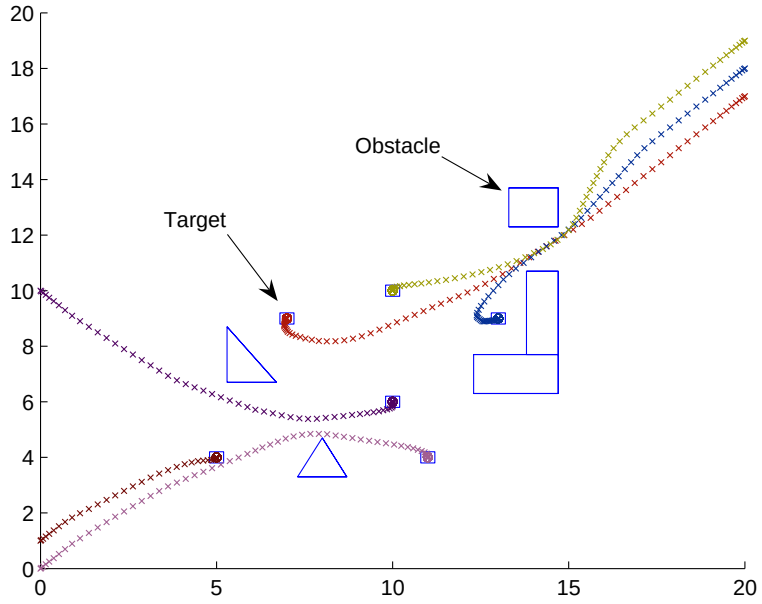


Figure 5.7: Six robots going to six targets while avoiding obstacles.

		# of robots				
		2	3	4	5	6
# of obstacles	1	.011	.011	.021	.025	.028
	2	.023	.027	.027	.028	.028
	3	.024	.028	.032	.032	.036

Table 5.1: Mean time to solve one optimization problem in the hierarchical/decentralized formulation, function of the number of robots and the number of obstacles.

	# of robots			
	2	3	4	5
global	402.1	575.4	768.4	992.2
Hierarchical/decentralized	881	1415	1804	2318

Table 5.2: Comparison between the cost of the global solution and the cost of the hierarchical/decentralized solution.

scenario with six robots and three obstacles in the centralized case 1433 s and the hierarchical/decentralized case 0.36 s it is possible to have an idea about the improvement in term of computational time required to solve the problem.

As stated before, the improvement in the required computational time comes with a deterioration of the solution, in the sense that the solution obtained with the hierarchical/decentralized algorithm will give a solution that is not as good as the one given from the centralized algorithm. In Table 5.2 is reported the comparison of the cost of the solution for the centralized and the hierarchical/decentralized algorithm. The environment considered is the one indicated in Fig. 4.5. As expected, the centralized algorithm is constantly better than the hierarchical/decentralized one. The solution obtained with the hierarchical/decentralizde algorithm is about 2.2-2.5 times the centralized solution that could be considered a satisfactory result.

The simulation in Fig.5.8 is reported to show the scalability of the algorithm over big team and complex environment. In the simulation 31 robots and 10 obstacles are considered.

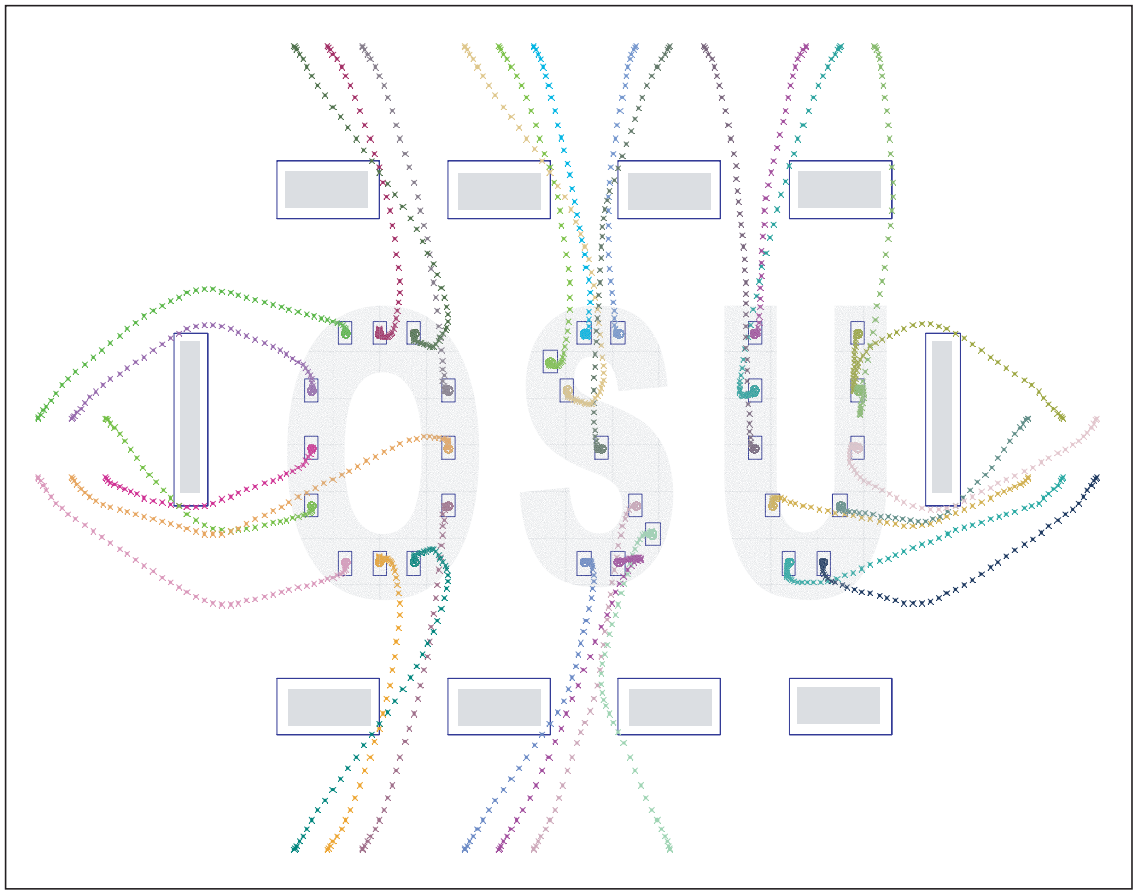


Figure 5.8: 31 robots going to 31 targets while navigating in a complex environment.

Chapter 6

Conclusion and future work

6.1 Conclusion

The problem of coordinating a team of mobile unmanned vehicles cooperating to accomplish a common task has been considered. In designing the control algorithm for the coordination of the team, the goal was to obtain an algorithm which would deal with both small and large groups. The algorithm must show flexibility such that it can be used for complex environments and tasks. Finally, last but not least it must give an optimal solution. Several algorithms addressing this problem can be found in the literature, but they do not seem to satisfy all the requirements. For example the ones scaling well on the number of members in the team may not show flexibility over the task that can be modeled or they may not consider optimization and so on. This work aimed to propose a control algorithm that satisfies all the properties mentioned before.

After reviewing the literature of the works on the cooperative control, it seemed that model predictive control (MPC) was the most promising control technique because its algorithm is based on the solution of an open-loop optimal control problem. Moreover and thanks to the on-line implementation, the model predictive control

offers a framework that addresses dynamic changes in the environment and in the constraints. These characteristics make model predictive control the basic building block for the analysis. However, model predictive control algorithms based on classical convex optimization techniques showed themselves not flexible enough to adapt to different kind of tasks and environments, since the requirement of having convex constraints was quite restrictive. Motivated by the need of having a more efficient tool, the attention was directed to mixed integer linear programming (MILP).

Mixed integer linear programming provides a broad set of tools for modeling non-convex constraints. In fact, it offers the possibility of including in the problem decision variables that can be associated to logical statements. Then, decision variables can be combined to model complex logical structures. Mixed integer linear programming also permits to define discrete variables to monitor the state of continuous variables. And finally, it gives a way to model some classes of non-linear functions.

The first attempt to address the coordination of multi-vehicle systems, then, was done by using a mixed integer linear program as the underneath optimal open-loop problem in the model predictive control algorithm. Even though, the approach was promising due to the combination of the properties of the MPC and MILP, the algorithm showed poor performance already for quite small teams and simple environments. The problem was in the NP-*hard* nature of the MILP problem. In fact, the computational time required to solve the MILP problem grows exponentially with the number of binary variables, and this number quickly becomes large when modeling a large team or complex environment. Then, although the algorithm showed some good properties, it didn't scale as adequately as required.

The desire to exploit the properties from the MPC and the MILP pushed us to look in a different structure that could improve the scalability of the algorithm. The

basic idea was to divide the services regarding the whole team, like the cooperation tasks, from the ones regarding the single member, for example the obstacle avoidance. Applying this concept, we proposed a hierarchical/decentralized structure. When a problem is too complicated to be solved in a single step it is a common practice to use a hierarchical approach and/or a decentralized approach. This causes us to divide the original problem into smaller problems that are easier to solve. The cost that must be paid in using these techniques is in terms of the value of the solution. In fact, the solution obtained using the hierarchical and decentralized model is usually worse than the one found solving the original problem.

Applying these concepts to the algorithm initially proposed, brought us to a new algorithm that keeps the original properties shown from the MPC and MILP adding at the same time good scalability over the size of the team and the complexity of the tasks and environments, while keeping the cost of the solution reasonably close to the one obtained solving the global problem. In conclusion it can be claimed that the proposed algorithm satisfies all the properties we were seeking for the solution of the problem of coordinating a team of mobile robots cooperating to accomplish a common task.

6.2 Future work

To improve the performance of the algorithm two methods are possible. The fist one is by improving the model. It is always possible to find smarter models that reduce the number of variables and constraints, and, at the same time present a solution that is closer to the global optimum. The second way is by improving the solver trying to reducing the computational time such that more complex problems could be solved

quickly.

We are also planning to develop a graphical user interface to allow a user to define the mission objective using high-level instructions. In this way the simulator can become a tool to test the high-level algorithm without entering too much into the details of the translation of the mission in terms of mixed integer linear programming.

Part of the future work will be trying to implement the algorithm on a real testbed under development in the MARHES laboratory. In order to make this vision a reality it will be necessary to build all the communication architecture between the central station and the agents. It will also be necessary to find a way to guarantee the synchronization among the agents. In addition to the informational architecture, more work will be necessary to develop the sensors needed to estimate the position of each robot which must be transmitted to the central station. At the same time each robot must be able to estimate the position of its neighbors and their velocity [15]. Due to the presence of delay, uncertainties and noise from sensors, it may be necessary to adjust the model to better fit these situations that were not considered in this work.

Bibliography

[1] M. Alighanbari and J. P. How. Cooperative task assignment of unmanned aerial vehicles in adversarial environments. In *Proc. American Control Conference*, pages 4661–4666, Portland, Oregon, June 8-10 2005.

[2] M. Baotic. *Matlab interface to CPLEX.* Available from http://control.ee.ethz.ch/ hybrid/cplexint.php.

[3] A. Bemporad and M. Morari. Control of system integrating logic, dynamics, and constraint. *Automatica*, 35:407–427, 1999.

[4] G. C. Chasparis and J. S. Shamma. Linear-programming-based multi-vehicle path planning with adversaries. In *Proc. American Control Conference*, pages 1072–1077, Portland, Oregon, June 8-10 2005.

[5] A. K. Das, R. Fierro, V. Kumar, J. P. Ostrowski, J. Spletzer, and C. J. Taylor. A vision-based formation control framework. *IEEE Trans. on Robotics and Automation*, 18(5):813–825, October 2002.

[6] A. K. Das, R. Fierro, V. Kumar, J.P. Ostrowski, J. Spletzer, and C.J. Taylor. A vision-based formation control frameworks. *IEEE Transactions on robotics and automation*, 18(5):813–825, October 2002.

[7] J.P. Desai. A graph theoretic approach for modelling mobile robot team formation. *Journal of robotic system*, 19(11):511–525, 2002.

[8] J.P. Desai, V. Kumar, and J.P. Ostrowski. Control of change in formation for a team of mobile robots. In *Proceeding of the 1999 IEEE international conference on robotics and automation*, volume 2, pages 1556–1561, Detroid, Michigan, May 10-15 1999.

[9] J.P. Desai, V. Kumar, and J.P. Ostrowski. Modelling and control of formations of nonholonomic mobile robots. *IEEE Transactions on robotics and automation*, 17(6):905–908, December 2001.

[10] W.B. Dunbar and R.M. Murray. Model predictive control of coordinated multi-vehicle formations. In *Proceeding of the 41th IEEE conference on decision and control*, volume 4, pages 4631–4636, Las Vegas, Nevada, USA, December 10-13 2002.

[11] W.B. Dunbar and R.M. Murray. Receding horizon control of multiple vehicle formations: a distributed implementation. In *Proc. IEEE Conf. on Decision and Control*, volume 2, pages 1995–2002, Atlantis, Paradise Island, Bahamas, December 14-17 2004.

[12] M. G. Earl and R. D'Andrea. Modeling and control of a multi-agent system using mixed integer linear programming. In *Proc. IEEE Conf. on Decision and Control*, volume 1, pages 107–111, Las Vegas, NV, December 10-13 2002.

[13] M. G. Earl and R. D'andrea. A decomposition approach to multi-vehicle cooperative control. Technical report, Department of Mechanical and Aerospace Engineering Cornell University, 2004. Available at http://control.mae.cornell.edu/earl/.

[14] M. G. Earl and R. D'andrea. Iterative milp methods for vehicle control problems. In *Proc. IEEE Conf. on Decision and Control*, volume 4, pages 4369–4374, Atlantis, Paradise Island, Bahamas, December 14-17 2004.

[15] D. Cruz C. Flesher B. Perteet J. McClintock R. Fierro. An experimental testbed for swarming and cooperative robotic networks. *Submitted to 2006 IEEE International Conference on Robotic and Automation.*

[16] R. Fierro, A. Das, V. Kumar, and J. P. Ostrowski. Hybrid control of formations of robots. In *Proc. IEEE Int. Conf. Robot. Automat.*, pages 157–162, Seoul, Korea, May 2001.

[17] S. S. Keerthi E.G. Gilbert. Optimal, infinite horizon feedback law for a general class of constrained discrete system: Stability and moving-horizon approximations. *Journal of Optimization Theory and Application*, 57:265–293, 1988.

[18] Y. Hao, A. Davari, and A. Manesh. Differential flatness-based trajectory planning for multiple unmanned aerial vehicle using mixed-integer linear programming. In *Proc. American Control Conference*, pages 104–109, Portland, Oregon, June 8-10 2005.

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

[20] H. G. Tanner A. Jadbabaie and G. J. Pappas. *Flocking in teams of nonholonomic agents*, volume 309, pages 229–239. Springer, lecture notes in control and information science edition, 2004.

[21] T. Keviczky, F. Borrelli, and G. J. Balas. A study on decentralized reciding horizon control for decoupled system. volume 6, pages 4921–4926, Boston, Massachusset, June 30 - July 2 2004.

[22] D. B. Kingston and C. J. Schumacher. Time-dependent cooperative assignment. In *Proc. American Control Conference*, pages 4084–4089, Portland, Oregon, June 8-10 2005.

[23] C.W. Reynolds. Flocks, herds, and schools: a distributed behaivor model. *Computer graphics*, 21(4):25–34, July 1987.

[24] A. Richards and J. P. How. A decentralized algorithm for robust constrained model predictive control. In *Proc. American Control Conference*, pages 4261–4266, Boston, Massachusetts, June 30-July 2 2004.

[25] A. Richards and J. P. How. Mixed-integer programming for control. In *Proc. American Control Conference*, pages 2676–2683, Portland, Oregon, June 8-10 2005.

[26] R. O. Saber, W. B. Dunbar, and R. M. Murray. Cooperative control of multi-vehicle systems using cost graphs and optimization. In *Proc. American Control Conference*, volume 3, pages 2217–2222, Denver, Colorado, June 4-6 2003.

[27] R. O. Saber and R. M. Murray. Floking with obstacle avoidance: cooperation with limited communication in mobile networks. In *Proc. IEEE Conf. on Decision and Control*, volume 2, pages 2022–2028, Maui, Hawaii, December 9-12 2003.

[28] D. H. Shim, H. J. Kim, and S. Sastry. Decentralized nonlinear model predictive control of multiple flying robots. In *Proc. IEEE Conf. on Decision and Control*, volume 4, pages 3621–3626, Maui, Hawaii, December 9-12 2003.

[29] T. Shima, S. J. Rasmussen, and A.G. Sparks. Uav cooperative multiple task assignment using genetic algorithms. In *Proc. American Control Conference*, pages 2989–2994, Portland, Oregon, June 8-10 2005.

[30] H. Tanner, A. Jadbabaie, and G.J. Pappas. Flocking in fixed and switching networks. *Submitted to IEEE Transactions on Automatic Control*, April 2005.

[31] R. R. Bitmead M. Gevers V. Wertz. *Adaptive optimal control - The thinking man's GPC*. Prentice-Hall, Englewood Cliffs, NJ, 1990.

[32] K. Wesselowski and R. Fierro. A dual-mode model predictive controller for robot formations. In *Proc. IEEE Conf. on Decision and Control*, pages 3615–3620, Maui, Hawaii, December 9-12 2003.

VITA

Carlo Branca

Candidate for the Degree of

Master of Science

Thesis: A HIERARCHICAL OPTIMIZATION APPROACH FOR COOPERATIVE VEHICLE NETWORKS

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Campobasso, Italy, on May 25th, 1978, son of Michele and Maria Branca.

Education: Graduated from Istituto Tecnico Industriale G. Marconi, Campobasso, Italy, in July 1996; received a laurea degree in Computer Engineering from Università di Roma Tor Vergata, Rome, Italy in May 2003. Completed the requirements for the Master of Science degree with a major in Electrical Engineering at Oklahoma State University in December, 2005.

Experience: Employed by Oklahoma State University, Department of Electrical Engineering as a graduate research assistant, 2003 to 2005.

Professional Memberships: Phi Kappa Phi Honor Society.

Name: Carlo Branca Date of Degree: December, 2005

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: A HIERARCHICAL OPTIMIZATION APPROACH FOR COOPERATIVE VEHICLE NETWORKS

Pages in Study: 99 Candidate for the Degree of Master of Science

Major Field: Electrical Engineering

This research presents a control algorithm for the cooperative control of unmanned mobile robots. The algorithm relies on continuously solving an open loop mixed integer linear programming optimization problem. Since the model can become quite complex when the number of robots and the complexity of the environment increase, computational problems can arise. To overcome these problems an approach involving a hierarchical decentralized formulation of the optimization problem is proposed.

ADVISOR'S APPROVAL: Dr. Rafael Fierro