UNIVERSITAT POLITÉCNICA DE CATALUNYA

Programa de Doctorat:
AUTOMÀTICA, ROBÒTICA I VISIÒ

Tesis Doctoral

DISTRIBUTED LARGE SCALE SYSTEMS: A MULTI-AGENT RL-MPC ARCHITECTURE

Valeria Javalera Rincón.

Directors: Vicenç Puig i Bernardo Morcego

Febrero 2016.

Distributed Large Scale Systems: a Multi-Agent RL-MPC Architecture

by Valeria Javalera Rincón

This thesis describes a methodology to deal with the interaction between MPC controllers in a distributed MPC architecture. This approach combines ideas from Distributed Artificial Intelligence (DAI) and Reinforcement Learning (RL) in order to provide a controller interaction based on cooperative agents and learning techniques. The aim of this methodology is to provide a general structure to perform optimal control in networked distributed environments, where multiple dependencies between subsystems are found. Those dependencies or connections often correspond to control variables. In that case, the distributed control has to be consistent in both subsystems. One of the main new concepts of this architecture is the negotiator agent. Negotiator agents interact with MPC agents to determine the optimal value of the shared control variables in a cooperative way using learning techniques (RL). The optimal value of those shared control variables has to accomplish a common goal, probably different from the specific goal of each agent sharing the variable. Two cases of study, in which the proposed architecture is applied and tested are considered, a small water distribution network and the Barcelona water network. The results suggest this approach is a promising strategy when centralized control is not a reasonable choice.

Table of Contents

List of figures

LIST OF TABLES

Chapter 1. Introduction

Large Scale Systems (LSS) are complex dynamical systems at service of everyone and in charge of industry, governments, and enterprises. The applications are wide. Examples of applications of LSS in continuous domains are: power networks, sewer networks, water networks, canal and river networks for agriculture, etc. Other examples of applications of LSS in discrete domain are traffic control, railway control, manufacturing industry, etc.

The quality of management and control of this kind of systems is crucial. Most of them are directly related with the quality of life of people in cities and have an impact on the environment preservation. As for example: sewer networks, metropolitan water networks, and canal and rivers networks for agriculture. If inefficient control strategies are used in these systems results might derive on: spills of contaminated water to the field, the sea or within the cities, floods, restrictions of water in the cities, bad quality of water, unsatisfied water demand needs in agriculture etc. In other types of LSS risks and consequences can be: pollution, traffic unsafety, blackouts, etc.

According to (Lunze, 1992), the notion of large-scale systems came into use when it became obvious that there are practical control problems that cannot be solved efficiently by the principles and methods of the classical control theory. The reason for this is that the systems to be controlled are too large and the problems to be solved too complex, in one sense or another, so that the amount of computation is too large to be manageable and even the basic assumptions of multivariable control are far from being satisfied.

In order to manage the complexity and the amount of computation required analyzing and controlling a large-scale system, designers are often forced to break down the whole problem into smaller subproblems, solve these subproblems separately, and then combine their solutions in order to get a global result for the original task. However, the subproblems are not independent. Some coordination or modification of the solutions of the subproblems is necessary in order to consider the interrelationships between them. The effort required to deal with these subproblems and their coordination can be allocated to various processors, which constitute a distributed computing system. Therefore, the concepts and techniques for reformulating a control problem as a set of interdependent subproblems and for solving these subproblems are often referred to as *distributed control.*

One of the leading control techniques used to deal with large-scale systems is *model predictive control* (MPC). The success of MPC is due to its ability to handle several dynamically coupled, manipulated and controlled variables (up to several hundreds) and constraints on them (Badwell, 2003). The latter are almost impossible to tackle by traditional frameworks. Since MPC directly embodies technical specifications (model, performance, limits) into the control algorithm, no a-posteriori patches are required to take into account limitations on system's variables. Being MPC a systematic design flow, independent of the model chosen and performance/constraint specifications, the major benefit is that design and implementation errors can be avoided at an early phase of the life cycle of the system, therefore reducing the cost and time to market compared to other more conventional control design methodologies.

Traditional MPC procedures assume that all available information is centralized. In fact, a global dynamical model of the system must be available for control design (off-line or a priori information). Moreover, all measurements must be collected in one location to estimate all states and compute all control actions (on-line or a posteriori information).When considering large-scale systems, the centrality assumption usually fails to hold, either because gathering all measurements in one location is not feasible, or because a centralized high-performance computing unit is not available.

A way of circumventing these issues is to look into *decentralized* or *distributed* MPC (DMPC) algorithms, in which the original large-size optimization problem is replaced by a number of smaller and easily tractable ones that work iteratively and cooperatively towards achieving a common, system-wide control objective. The industrial success of traditional MPC drives now a new interest in this old area of distributed/decentralized control, and distributed/decentralized MPC has become one of the hottest topics in process control in the early 21st century, both in the US and in Europe. The new research concerns not only the issues related to the underlying optimization as feasibility, convergence and computational effort, but also the closed-loop issues of stability and robustness.

In decentralized MPC, the resulting subsystems are independent from each other. But the high level of connections and interdependence of LSS is the reason why, in most cases, they cannot be modeled as decentralized systems. In distributed systems, the resulting subsystems can have physical dependencies between them and therefore communication among them. One of the main problems of distributed control of LSS is how these dependence relations between subsystems are preserved. These relations could be, for example, pipes that connect two different control zones of a decentralized water transport network, or any other kind of connection between different control zones. When these connections represent control variables, the distributed control has to be consistent for both zones and the optimal value of these variables will have to accomplish a common goal. In order to do this, many negotiation techniques have been proposed (see for example, (Camponogara, 2002), (Negenborn, 2008), (Venkat, 2005), (El Fawal, 1998), (Gómez, 1998) and (Rawlings, 2008)). Calculation time, problems handling multiple restrictions and multiple objectives and the impossibility to ensure convergence are the main problems of these approaches. Although there have been successful results there is still a need of a methodology that can be used for all kind of continuous LSS.

The present thesis addresses this open problem in control theory by the combination of adequate control and computer science techniques, more precisely, the combination of Model Predictive control (MPC), Multi-Agent Systems (MAS), and Reinforcement Learning (RL). Considering that Distributed Control (DC) shares philosophic aspects with Distributed Artificial Intelligence (DAI), the idea is to apply MAS techniques and technology to DC problems as communication, coordination, need of adaptation (learning), autonomy and intelligence.

The use of MAS will allow to:

- Enjoy all the benefits of distributed systems like speed-up of the system, due to parallel computation, scalability and flexibility due to the modularity of the system, simplicity of design and maintenance of the system, robustness and reliability due to the possibility to implement failure tolerance.
- Perform an appropriate coordination and synchronization of the agents.
- Provide a management and communication platform for the MAS. This will allow allocating MPC Agents in different computers of a network.
- Use appropriate tools of development and standards.
- Use methods and tools of Analysis and Design in order to make an appropriate formalization and documentation of the system.

The use of RL in the negotiation process will allow to:

- Make the process of negotiation adaptive.
- Learn from its own experience.
- Explicitly consider the whole problem of two goal-oriented agents.
- Deal with a dynamical and uncertain environment.
- Connect the process of negotiation with the one of the MPC control, because of the compatibilities found between them.

1.1 Scope

The objective of using a real case of study in this work is to validate results and technical viability of the proposed architecture. Although the solution obtained by the proposed architecture and methodology has to be efficient for the considered case, it has to be general enough for being applied in any kind of continuous LSS.

This thesis has been developed in the context of the European Project Decentralized and Wireless Control of Large Scale Systems, WIDE - 224168 - FP7-ICT-2007-2. The case of study is the Barcelona water transport network. The network is managed by the company Aguas de Barcelona (AGBAR), a partner of the WIDE project. AGBAR not only supplies water to Barcelona city but also to the metropolitan area (see data in the table 1.1).

Territorial extension	425	km^2
Drinkable water net	4.470	km
Drinkable water production	237,7	hm^3
Population	2.828.235	

Table 1.1 Metropolitan Area Of Barcelona- Water Net (2006)

The sources of water are the rivers Ter and Llobregat. Since 1976, the network had a centralized tele-control system, organized in a two-level architecture (see Figure 1.1). At the upper level, a supervisory control system installed in the control center of AGBAR is in charge to optimally control the whole network by taking into account operational restrictions and consumer demands. This upper level provides the set-points for the lower-level control system. This optimizes the pressure profile to minimize losses caused by leakage and provide sufficient pressure, e.g. for high buildings. The system responds to changes in network topology (ruptures), typical daily/weekly profiles, as well as major changes in demand, etc.

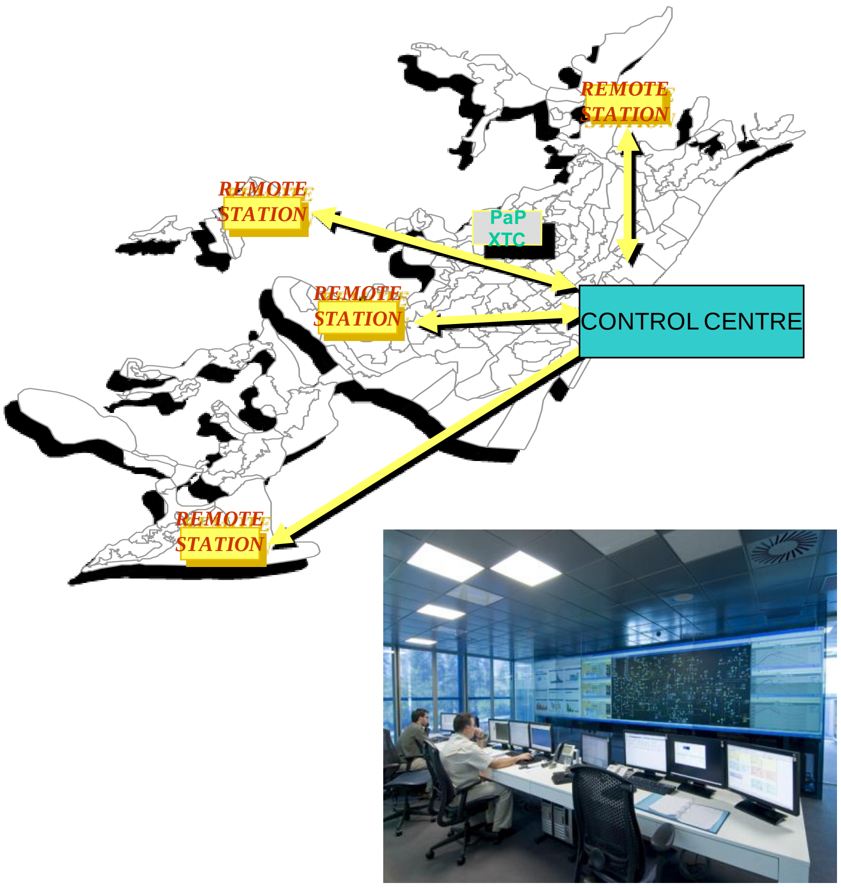


Figure 1.1 Agbar´s Centralized Tele-Control System.

The Barcelona water network is comprised of 200 sectors with approximately 400 control points. At present, the Barcelona information system receives, in real time, data from 200 control points, mainly through flow meters and a few pressure sensors. Sensors measurements are sent to the operational data base of the telecontrol information system via telephone XTC network or GSM radio using the ModBus communication protocol. This water network, as any other, is composed by nodes, valves, pumps, tubes, and sources. Figure 1.2 depicts the diagram of the Barcelona water transport network.

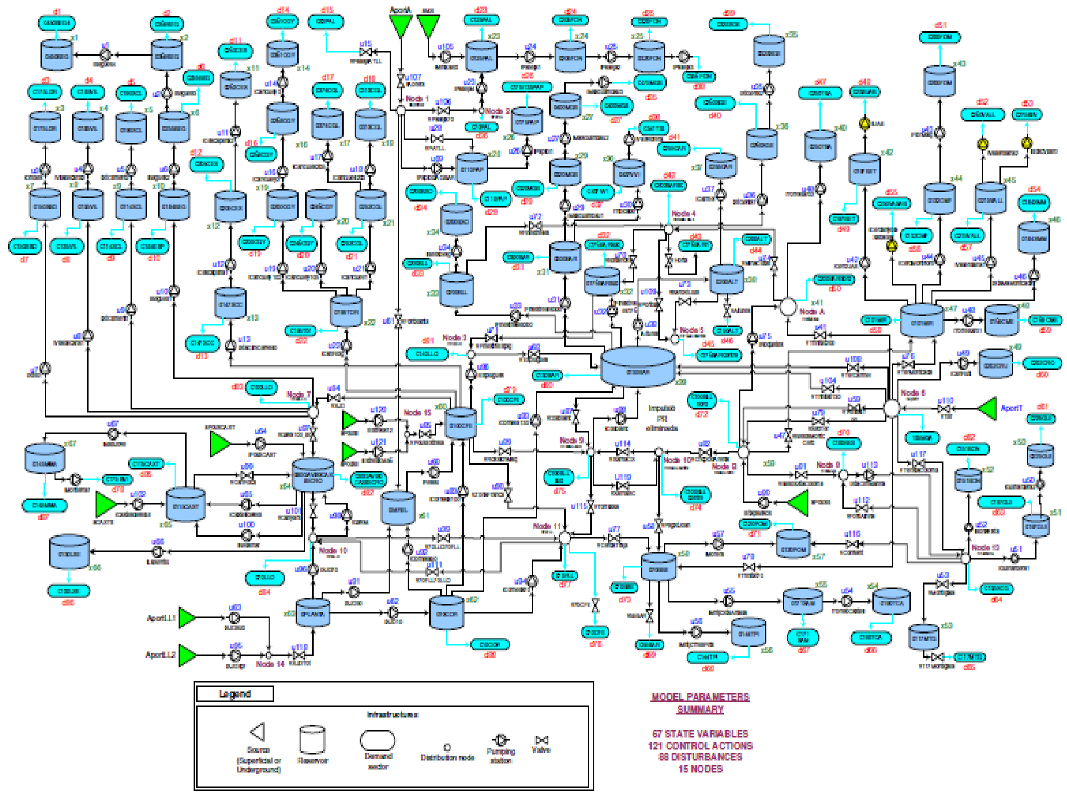


Figure 1.2 Diagram of the Barcelona Water Transport Network.

1.2 Objectives

The specific objectives of this thesis are:

- To develop a distributed control architecture for LSS based on three main concepts: Negotiation-Cooperation-Learning.
- To combine Model Predictive Control (MPC), Reinforcement Learning (RL) and the Agent Oriented Paradigm (AOP) as the basis of the proposed approach.
- To prove technical feasibility of the proposed approach.
- To provide a general methodology for the application of the proposed architecture.
- To validate the proposed architecture applying it to the Barcelona water transportation network.
- To compare results against the centralized MPC and the decentralized approach presented in this thesis, applied to the same case of study.

1.3 Expected contributions

As it will be discussed in the state of art (presented in Chapter 2), MPC, RL and AOP are powerful tools widely studied and applied each one in their own area. Works have been made relating MPC and RL but not in cooperative environments. There is a very short intersection between AOP and control and no intersection at all of these three areas. This work proposes two main learning techniques obtained as a result of the combination of elements in this intersection and a methodology that is expected to support future applications in LSS.

Another expected contribution is to introduce the term agent in the control language as a basic element of the AOP and to combine suitable solutions between distributed control and Distributed Artificial Intelligence (DAI).

A step by step application of the proposed methodology in a case of study is presented using two different learning algorithms proposed (Chapter 6 and Chapter 7). The one that presents the best result is applied to the Barcelona case of study (Chapter 8). It is expected that these examples motivate other practical applications in LSS using the techniques presented in this thesis.

Organization of the thesis is as follows: Chapter 2 presents the state of the art of the related areas. Chapter 3 introduces the problem to be solved and describes the formalization of the proposed framework. Chapter 4 describes the learning techniques used. Chapter 5 describes the proposed MA-MPC methodology. Chapter 6 and 7 are devoted to present the proposed approach using learning by teacher and by exploration using selective reward and their illustration using an application example. Chapter 8 presents results using the Barcelona water network case. And finally, conclusions and further research are shown in Chapter 9.

Some of the publications related with this thesis are:

Conference proceedings:

- Javalera, V.; Morcego, B.; Puig, V. Distributed MPC for Large Scale Systems using agent-based reinforcement learning., 12th IFAC Symposium on Large-Scale Systems: Theory and Applications, 2010, Villeneuve d'Ascq, France, p. 1-6.

- Javalera, V.; Morcego, B.; Puig, V. A multi-agent MPC architecture for distributed large scale systems. A: 2nd International Conference on Agents and Artificial Intelligence. "2nd International Conference on Agents and Artificial Intelligence". Valencia: INSTICC Press. Institute for Systems and Technologies of Information, Control and Communication, 2010, p. 544-551.

- Javalera, V.; Morcego, B.; Puig, V. Negotiation and learning in distributed MPC of large scale systems. A: American Control Conference. "Proceedings of the 2010 American Control Conference". Baltimore: IEEE Press. Institute of Electrical and Electronics Engineers, 2010, p. 3168-3173.

Book chapter:

- Morcego, B.; Javalera, V.; Puig, V., Vito, R. Distributed MPC using reinforcement learning based negotiation: Applications to large scale systems. En Maestre, J. (eds.), *Distributed Model Predictive Control made easy*. Dordrech: Springer Science+Business Media. 2014, p. 517-534.

Chapter 2. State of Art.

2.1 Model predictive control philosophy

As it was mentioned before, MPC is a recognized powerful approach with proven capability to handle a large number of industrial control problems. The philosophy of MPC is well resumed in (Scattolini, 2009) when it emphasizes that the main characteristic of MPC is to transform the control problem into an optimization one, so that at any sampling time instant, a sequence of future control values is computed by solving a finite horizon optimal control problem. Then, only the first element of the computed control sequence is effectively used and the overall procedure is repeated at the next sampling time according to the so-called receding horizon principle. For a more detailed explanation about MPC see the text book (Camacho, 2007).

The main characteristics of centralized MPC are (Negenborn D. S., 2004):

- The centralized system model is given by a (possibly time-varying) dynamic system of difference or differential equations and constraints on inputs, states, and outputs.
- The goal of the control problem is to minimize a cost function. The control problem is stated as a multiple-objective optimization problem that is transformed to a single objective one using a weighted approach.
- The problem is solved by a single centralized agent, the information set of which consists of measurements of the physical system, and the control action set of which consists of all possible control actions.

A controller based on MPC solves the problem with a three-step procedure (see Figure 2.1):

- It reformulates the problem of controlling the time-varying dynamic system using a time-invariant approximation of the system, with a control and a prediction horizon to make the solution computation tractable and a rolling horizon for robustness.
- It solves the reformulated control problems, often using general, numerical solutions techniques, while taking into account constraints on control actions and states.
- It combines the solutions to the approximations to obtain a solution of the overall problem. This typically involves implementing the control actions calculated from the beginning of the time horizon of the current approximation, until the beginning of the next approximation.

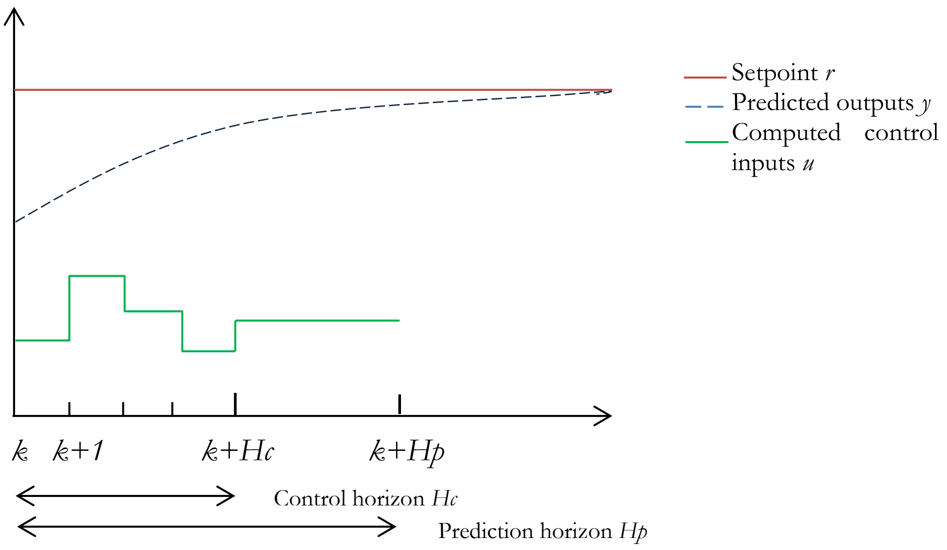


Figure 2.1 Example of conventional MPC.

In Figure 2.1, it can be noticed that the solution of the MPC problem is to find actions u_k … $u_k + H_c$, such that after H_p steps, the system behavior y approaches to the desired behavior r. In this example, y indeed reaches the desired set point r (Negenborn D. S., 2004). One can find some advantage and disadvantages of the MPC framework as discussed in the following:

Advantages

- The MPC framework handles input, state, and output constraints explicitly in a systematic way. This is due to the control problem formulation is based on the system model which includes the constraints.
- It can operate without intervention for long periods. This is due to the receding horizon principle, which enables the controller to looks ahead to prevent the system from going in the wrong direction.
- It adapts easily to new contexts because of the receding horizon use.

Disadvantages

- When the prediction horizon becomes large, the number of variables of which the agent has to find the value increases quickly.

- The resources needed for computation and memory may be high, increasing more when the time horizon increases. The amount of required resources also grows with increasing system complexity.

2.1.1 MPC Taxonomy

New approaches of MPC are continuously arising. Although it is a well-known and accepted control strategy, it is still an open field of research. Three main types of structures in which MPC is applied can be found in the literature (See Figure 2.2).

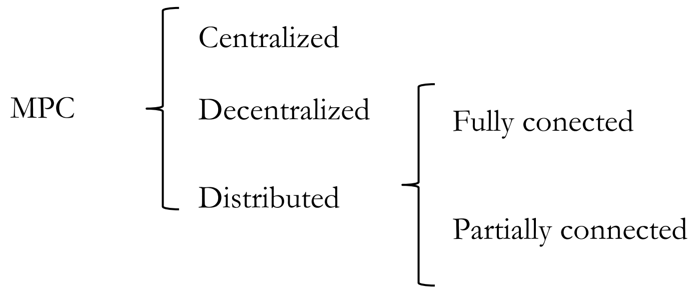


Figure 2.2 MPC Taxonomy

Centralized MPC is the classical way of implementing the MPC strategy. In (Rawlings, 2008) the authors state that the move from distributed PID to MPC of small systems was essentially a move towards centralized decision making. This technology gained support because the performance benefits were large.

But, as it was mentioned before, there are strong reasons that lead MPC to decentralized or distributed implementations. According to (Scattolini, 2009), decentralized control (See for example (Ocampo, 2014) is based on considering the control input (*u*) and the controlled output (*y*) variables are grouped into disjoints sets. These sets are then coupled to produce non-overlapping pairs from which local regulators can be single-input single-output or multivariable (locally centralized) depending on the cardinality of the selected input and output groups.

Many proposals have been suggested to define these sets disjoint where they are not naturally in that way (Camponogara, 2002) (El Fawal, 1998) (Van Breemen, 2001) (Barcelli, 2008)

(Rawlings, 2008). Another line of research is to communicate overlapping sets. This is called distributed control.

In distributed control structures, it is assumed that some information is transmitted among the local regulators, so that each one of them has some knowledge about the behavior of the others. In the next section distributed control implemented with the MPC approach is discussed.

2.1.2 Distributed MPC

When the local regulators of a distributed control structure are designed with MPC, the information typically transmitted consists of the future predicted control or state variables computed locally. In this way, any local regulator can predict the interaction effects over the considered prediction horizon. If the information exchange among the local regulators concerns the predicted evolution of the systems states, any local regulator needs only to know the dynamics of the subsystem directly controlled. On the contrary, if the predictive control actions are transmitted, the local regulators must know the model of all subsystems. In any case, it is apparent that the transmission and the synchronization protocols have major impact on the achievable performance (Scattolini, 2009).

Fully connected algorithms (see Figure 2.3) are the ones in which every regulator has bi-directional communication with all the other regulators. If any local regulator has communication just with a subset of the others, then it is partially connected.

In (Scattolini, 2009), a taxonomy of MPC approaches based in the protocol used for exchanging information among local regulators is provided. This taxonomy is summarized in the following diagram:

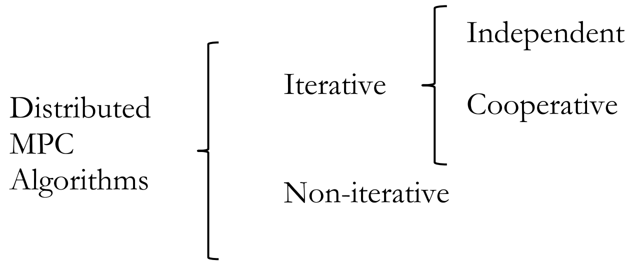


Figure 2.3 Distributed MPC Taxonomy.

In iterative algorithms, information is bi-directionally transmitted among local regulator many times within the sampling time. In non-iterative algorithms information is bi-directionally transmitted among the local regulators only once within each sampling time.

In iterative algorithms (see for example (Liu, 2014)) there is a sub-classification. When each local regulator minimizes a local performance index, it is said to be an independent algorithm, and when they minimize a global cost function it is called cooperative algorithm.

Independent (non-cooperative) algorithms are widely studied in game theory (much more widely used in comparison with cooperative algorithms) and also applied in MPC distributed control strategies. An example of application of min-max algorithm can be found in (Jia, 2002). Other application of non-cooperative algorithms examples are (Valencia, 2014), (Betti, 2014). Negotiation algorithms are also taken as non-cooperative according to many authors, an application example of a negotiation algorithm can be found in (Maestre, 2014).

As discussed in (Venkat, 2005), it is apparent that in iterative and independent algorithms each local regulator tends to move towards a Nash equilibrium, while iterative and cooperating methods seek to achieve the Pareto optimal solution provided by an ideal centralized control structure. However, Nash equilibrium can be even unstable and far from the Pareto optimal solution. So, specific constraints have to be included in the MPC problem formulation to guarantee closed-loop stability (Scattolini, 2009). As for the MPC algorithms published in the literature, the state feedback method described in (Camponogara, 2002) for discrete-time linear systems belongs to the set of independent noniterative algorithms. A stability constraint is included in the problem formulation, although stability can be verified only a-posteriori with an analysis of the resulting closed-loop dynamics. Nash equilibrium solutions are searched in the independent, iterative and fully connected methods developed in (Du, 2001) for discrete-time unconstrained linear systems represented by input–output models (Scattolini, 2009).

There is an analogous classification of distributed MPC taxonomy in game theory. Distributed algorithms correspond to algorithms where there is an exchange of information between players. MPC iterative algorithms correspond to dynamic algorithms, whereas MPC non-iterative algorithms correspond to static algorithms. MPC independent algorithms correspond to non-cooperative algorithms and cooperative algorithms are called cooperative as well.

Cooperative algorithms are important in distributed control systems because they seek the global optimum of the system, besides local optimums. There are many cooperative algorithms for DMPC in literature. In (Richards, 2014), cooperation is taken to mean the improvement of system-wide performance through the avoidance of greedy behaviors by individual agents. Coupled constraint satisfaction is, however, maintained without the need for inter-agent negotiation or bargaining. In (Jurado, 2014), cooperative MPC agents communicate with their neighbors in a flexible control architecture by adapting it to the possible changes in the network conditions. Other applications of cooperative algorithms can be found in (Pannocchia, 2014), (Ferramosca L. G., 2014), (Ferramosca, 2014).

An interesting approach is presented in (Venkat, 2005), where an iterative, cooperating method for linear discrete-time systems is presented. In particular, the proposed approach guarantees the attainment of the global (Pareto) optimum when the iterative procedure converges, but still ensures closed-loop stability and feasibility if the procedure is stopped at any intermediate iteration (Scattolini, 2009).

In (Rawlings, 2008), an alternative approach to solve the same problem was discussed. The novelty involves maintaining the distributed structure of all the local controllers, but changing the objective functions so that the local agents cooperate.

The seminal Tamura coordination method was discussed in the book (Brdys, 1994) even before MPC was first introduced. This method is based on using augmented Lagrangian to negotiate values on overlapping sub-networks in distributed large scale systems. Other works have applied this method (El Fawal, 1998) (Gómez, 1998) (Negenborn, 2008).

2.3 Reinforcement Learning.

Learning is the incorporation of knowledge and skills by an agent, leading to an improvement in the agent's performance (Busonui L., 2005). Learning techniques are powerful tools used mainly in large and complex systems in dynamical environments. Reinforcement learning is based on past experience, which, in this work, is used to reduce the need of iterative methods, facilitating that the system behaves almost like a reactive system with a very short time of response. RL is a well-known and formally studied family of learning techniques.

Moreover, depending on the formulation of the problem and the richness of experience data, the chances of convergence are high. Another key feature of reinforcement learning is that it explicitly considers the whole problem of a goal-directed agent interacting with an uncertain environment. This is in contrast with many approaches that consider subproblems without addressing how they might fit into a larger picture (Sutton, 1998).

Due to the difficulties in dealing with open and time-varying environments, most multiagent learning algorithms are designed for unchanging environments. They typically involve some fixed learning structures that are updated by a set of rules involving some fixed or scheduled parameters. This kind of learning is called "static" learning (Busonui L., 2005).

By allowing the learning parameters or structures of the static algorithms to adapt, the learning processes of the agents should be able to regain their ability of handling open and time-varying environments (Busonui L., 2005).

Note that adaptive learning is not a radically different process from learning. It can be viewed as a kind of "meta-learning" – that is, a special case of "learning how to learn" (Busonui L., 2005).In the book (Sutton, 1998), Reinforcement Learning (RL) is defined as: learning what to do, how to map situations to actions, so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the maximum reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics (trial-and-error search and delayed reward) are the two most important distinguishing features of reinforcement learning.

Reinforcement learning is defined by characterizing a learning problem, rather than by characterizing learning methods. Any method that is well suited to solving that problem is considered to be a reinforcement learning method. The basic idea is to capture the most important aspects of the real problem faced by a learning agent by interacting with its environment to achieve a goal. Clearly, such an agent must be able to sense the state of the environment to some extent and must be able to take actions that affect the state and return a reward (Figure 2.4). The agent also must have a goal or goals relating to the state of the environment. The formulation is intended to include just these three aspects (sensation, action, and goal) in their simplest possible forms without trivializing any of them (Sutton, 1998).

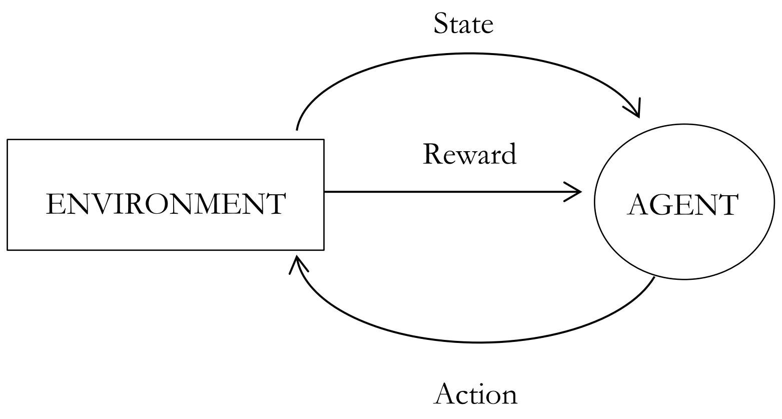


Figure 2.4 RL agent interactions (Alpaydin, 2014).

Although the applications of RL are typically static, many control applications have been developed for dynamical environments (Agostini, 2005), (Martinez E., 2003), (Tesauro, 2003) recently (Gatti, 2015) show many examples of experiments of RL applied in dynamical environments. In (Hester, 2013) an algorithm to be both data-efficient and computation-efficient enough to work on real robots in the real world is shown. Even more, there are some works that relate MPC and RL. In (Ernst, 2007) a comparison between both approaches is made, and in (Ernst D., 2006) they are seen as complementary frameworks.

An interesting paper about cooperative learning applying RL in control is (Bakhtiari, 2007). In the area of Distributed Artificial Intelligence, papers about learning in cooperative Multi-Agent systems with RL are (Lauer, 2000) (Claus, 1998) (Kapentanakis, 2002). The last one considers also coordination. Another application of RL for coordination in Multi-agents Systems is (Boutilier, 1999). In all those papers, the term Multi-Agent is referring to agents in Distributed Artificial Intelligence terminology. In the next section a short description of these terms will be introduced.

The theoretical foundations of the approach proposed in this thesis are related to Markov Decision Processes (MDP) (Alpaydin, 2014). MDPs are the formal description of discrete time stochastic control problems. Its main components are: the state transition probability function, which describes the behavior of the process; the controller policy, which is the sought function that assigns a control action to a state; and the reward function, which evaluates the quality of state transitions. The two areas that have mainly dealt with the algorithms that search solutions of problems formulated as MDPs are Dynamic Programming (DP) and Reinforcement Learning (RL). DP needs an explicit model of the state transition probability function and the reward function while RL does not. Also previously discussed in this chapter, RL is a well-established theory and there are many contributions that analyze the optimality of its variants. Q-learning, developed by (Watkins, 1989) is one of the most celebrated reinforcement learning algorithms.

This learning algorithm was proved to be convergent for discounted Markov decision problems (Dayan, 1992), and later it was also proved to be convergent in more general cases (Jaakkola, 1994) (Tsitsiklis, 1994). Those proofs allow us to consider the negotiation agent as an optimal negotiator given sufficient iterations. The most important step is the appropriate selection of the cost function, which will take the MPC agents to the global optimum solution or elsewhere.

2.3.1. Elements of Reinforcement Learning

In this section the main elements of RL are explained. A full specification of the reinforcement learning problem in terms of optimal control of Markov decision processes and a deeper explanation about the most important RL topics can be found in (Sutton, 1998).

The learning decision maker is called the *agent*. The agent interacts with the *environment* that includes everything outside the agent. The agent has *sensors* to sense its *state* in the environment and takes *actions* that modify its state. When the agent takes an action, the environment provides a *reward*. Time is discrete as k = 0, 1, 2, ... , and $s_k \in S$ denotes the state of the agent at time k where S is the set of all possible states. $a_k \in A\ (s_k\ )$ denotes the action that the agent takes at time k where A (s_k) is the set of possible actions in state s_k . When the agent in state s_k takes the action a_k , the clock ticks, reward $r_{t+1} \in$ is received, and the agent moves to the next state, s_{k+1}. The problem is modeled using a Markov decision process (MDP) (Alpaydin, 2014).

The *policy* defines the agent´s behavior and is a mapping from the states of the environment to actions: $\pi: S \rightarrow$ A. The policy defines the action to be taken in any state $s_t : a_t = \pi(s_t)$. The *value of a policy* is the expected cumulative reward that will be received while the agent follows the policy, starting from state s_t (Alpaydin, 2014).

In the *finite-horizon* model, the agent tries to maximize the expected reward for the next T steps. Certain tasks are continuing, and there is no prior fixed limit to the episode. In the *infinite-horizon* model, there is no sequence limit, but future rewards are discounted using a *discount rate* α to keep the return finite. If α=0, then only the immediate reward count. As α approaches to 1, reward further in the future count more (Alpaydin, 2014).

RL can be used with or without a model. *Model* in RL, is defined by the reward and the next state probability distributions, when we know these, we can solve for the optimal policy using *dynamic programming*. However these methods are costly and we seldom have such perfect knowledge of the environment. The more interesting and realistic applications of RL is when we do not have the model. This requires exploration of the environment to query the model, in order to do that, it is required the environment model to be stationary (Alpaydin, 2014).

When we explore and get to see the value of the next state and reward, we use this information to update the value of the current state. These algorithms are called *temporal difference* algorithms because what we do is look at the difference between our current estimate of a state and the discounted value of the next state and reward received (Alpaydin, 2014).

In model-free learning, we first discuss the simpler deterministic case, where at any state-action pair, there is a single reward and next state possible (Alpaydin, 2014):

$$Q\ (s_t\ , a_t) + \propto \max Q\ (s_{t+1}, a_{t+1}) \tag{2.1}$$

And we simple use this as an assignment to update $Q\ (s_t\ , a_t)$. When in state s_t we choose an action a_t, which returns a reward and takes us to state s_{t+1}. We then update the value of previous action as (Alpaydin, 2014):

$$Q'\ (s_t\ , a_t) \leftarrow\ r_{t+1}\ \propto \max Q'\ (s_{t+1}, a_{t+1}) \tag{2.2}$$

Some models produce a description of all possibilities and their probabilities; these we call *distribution models*. Other models produce just one of the possibilities, sampled according to the probabilities; these we call *sample models*. Distribution models are stronger than sample models in that they can always be used to produce samples. However, in surprisingly many applications it is much easier to obtain sample models than distribution models (Sutton, 1998).

Models can be used to mimic or simulate experience. Given a starting state and action, a sample model produces a possible transition, and a distribution model generates all possible transitions weighted by their probabilities of occurring. Given a starting state and a policy, a sample model could produce an entire episode, and a distribution model could generate all possible episodes and their probabilities. In either case, we say the model is used to *simulate* the environment and produce *simulated experience* (Sutton, 1998).

The word *planning* is used in several different ways in different fields. In RL the term is used to refer to any computational process that takes a model as input and produces or improves a policy for interacting with the modeled environment (Sutton, 1998).

In artificial intelligence, there are two distinct approaches to planning. In *state-space planning*, planning is viewed primarily as a search through the state space for an optimal policy or path to a goal. Actions cause transitions from state to state, and value functions are computed over states. In what we call *plan-space planning*, planning is instead viewed as a search through the space of plans. Operators transform one plan into another, and value functions, if any, are defined over the space of plans. Plan-space planning includes evolutionary methods and *partial-order planning*, a popular kind of planning in artificial intelligence in which the ordering of steps is not completely determined at all stages of planning. Plan-space methods are difficult to apply efficiently to the stochastic optimal control problems that are the focus in reinforcement learning (Sutton, 1998).

Learning and planning are similar. The heart of both, learning and planning methods, is the estimation of value functions by backup operations. The difference is that whereas planning uses simulated experience generated by a model, learning methods use real experience generated by the environment. Of course this difference leads to a number of other differences, for example, in how performance is assessed and in how flexibly experience can be generated. But the common structure means that many ideas and algorithms can be transferred between planning and learning. In particular, in many cases a learning algorithm can be substituted for the key backup step of a planning method. Learning methods require only experience as input, and in many cases they can be applied to simulated experience just as well as to real experience. Algorithm 1 shows a simple example of a planning method based on one-step tabular Q-learning and on random samples from a sample model. This method, which we call *random-sample one-step tabular Q-planning*, converges to the optimal policy for the model under the same conditions that one-step tabular Q-learning converges to the optimal policy for the real environment (each state-action pair must be selected an infinite number of times in Step 1, and α must decrease appropriately over time).

Algorithm 1 Random-sample one-step tabular Q-planning.

```
1.  Do forever:
2.  Select a state, $s \in S$, and an action $a \in A(s)$, at random
3.  Send $s, a$ to a sample model, and obtain a sample next state $s'$, and a sample
    next $r$
4.  Apply $Q(s,a) \leftarrow Q(s,a) + \propto [r \; + \gamma \; \max_{a'} Q(s',a)' - Q(s,a)]$
5.  end
```

The benefits of planning in small, incremental steps enables planning to be interrupted or redirected at any time with little wasted computation, which appears to be a key requirement for efficiently intermixing planning with acting and with learning of the model. More surprisingly, there is evidence that demonstrate that planning in very small steps may be the most efficient approach even on pure planning problems if the problem is too large to be solved exactly (Sutton, 1998).

2.4 Multi Agent Systems

The term agent has been used indiscriminately until now in this work. In control, distributed and decentralized systems are usually called Multi-agent Systems and their local controllers are called agents. In RL, the controller or the software entity that performs a RL algorithm is also called agent. There is a branch of Artificial Intelligence called Distributed Artificial Intelligence (DAI). This branch arises as a result of the natural evolution of the systems that could be found because they are more and more complex, large and often heterogeneous.

The solution of problems of this nature under a traditional scheme, involved the design of large and complex algorithms that use to consume a very high level of resources for calculation. It was about the 80´s that it was thought that small and simple programs that interact with each other could considerably simplify the design and development of these systems reducing the necessary resources.

Many DAI researchers have defined the term Agent. This term is still a controversial issue. In (Stan, 1996), the main agent definitions are presented and explained, and a taxonomy of autonomous agents is also provided. Next, some of these definitions are presented.

The Maes Agent: "Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed" (Stan, 1996).

The IBM Agent: "Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in doing so, employ some knowledge or representation of the user's goals or desires." (Stan, 1996).

The Wooldridge and Jennings Agent: A hardware or (more usually) software-based computer system that enjoys the following properties:

- Autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;
- Social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language;
- Reactivity: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;
- Pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative (Stan, 1996).

As a result of many years of research in this area, important contributions have been made on theory, methodologies, communications protocols, standards and software tools (Pokahr A., 2013), (Milan Vidakovic, 2013) that lead to the appearance of the Agent Oriented Paradigm (AOP). In (Woolridge, 1995), a survey of agent theories, architectures and programming languages present at that time is presented. Since then, many books journals and conferences have appeared. In (Balke, 2013), the status of agent applications in most important agent-related scientific conferences and journals is presented; the stage of the adoption of this paradigm in industry is also discussed.

The AOP is widely used in software applications and especially in Internet applications (for example e-commerce (Rahman, 2001), (Hartung, 2013), (Hakansson, 2010) and in service oriented computing (Morge, 2013)). Other interesting applications in robotics can be found in literature (Novak, 2013).

In control applications sometimes the terms of agent in DAI and in control are not consistent although there are some applications in terms of agents in the AOP way. Examples of these applications are: (Maturana, 2005) were new tools for developing MAS in distributed control applications are described and a case of study of a chilled-water system of a ship is presented; in (Tatara, 2007), an application in distributed control network of interconnected chemical reactors is presented.

In (Stan, 1996), we can find the following table that shows some properties of the agents.

Property	**Other Names**	**Meaning**
Reactive	(sensing and acting)	Responds in a timely fashion to changes in the environment.
Autonomous		Exercises control over its own actions.
Goal-oriented	pro-active purposeful	Does not simply act in response to the environment.
Temporally continuous		Is a continuously running process
Communicative	socially able	Communicates with other agents, perhaps including people.
Learning	adaptive	Changes its behavior based on its previous experience.
Mobile		Able to transport itself from one machine to another.
Flexible		Actions are not scripted
Character		Believable "personality" and emotional state.

Table 2.1 Characteristics of the Agents.

2.4.1 Potential advantages of Multi-Agent Systems

Some of MAS principal potential advantages over centralized systems are listed in (Busonui L., 2005):

- Speed-up of the system activity, due to parallel computation.
- Robustness and reliability, when the capabilities of the agents overlap. The system is tolerant to faults in one or several agents, by having other agents take over the activity of the faulty ones.
- Scalability and flexibility. In principle, since MAS are inherently modular, adding and removing agents to the system should be easy. In this way, the system could adapt to a changing task on-the-fly, without ever needing to shut down or to be redesigned.
- Ease of design, development, and maintenance. This also follows from the inherent modularity of the MAS. The potential benefits described above should be carefully weighed with the simplicity of a centralized solution, considering the characteristics of the task.

.

Chapter 3. Formulation of the framework

3.1. The problem

In order to control an LSS in a distributed way, some assumptions have to be made on its dynamics, i.e. on the way the system behaves. Let us assume first that the system can be decomposed into *n* subsystems, where each subsystem consists of a subset of the system equations and the interconnections between them. The problem of determining the partitions of the system is not addressed in this work (see e.q. (Lunze, 1992) for classical ways of addressing this problem). The set of partitions should be complete. This means that all system states and control variables should be included at least in one of the partitions.

Definition 3.1 *System partitions.* P is the set of system partitions and is defined by

$$P = \{p_1, p_2, \dots, p_i\}$$

(3.1)

where each system partition (subsystem) p_i is described by a deterministic linear time-invariant (LTI) model that is expressed in discrete-time as follows

$$\begin{aligned} x_i(k+1) &= A_i x_i(k) + B_{ui}(k) + B_{d,i} d_i(k) \\ y_i(k) &= C_i x_i(k) + D_{u,i} u_i(k) + D_{d,i} d_i(k) \end{aligned}$$

(3.2)

where variables *x, y, u* and *d* are the state, output, input and disturbance vectors of appropriate dimentions, respectively; *A, B, C* and *D* are the state, output, input and direct matrices, respectively. Subindexes *u* and *d* refer to the type of inputs the matrices model, either control inputs or exogenous inputs (disturbances). Control variables are classified as internal or shared according if they belong only to the subsystem or are shared with other subsystems.

Definition 3.2 *Internal Variables.* Internal variables are control variables that appear in the model of only one subsystem in the problem. The set of internal variables of a partition *i* is defined by

$$U_i = \{u_1, u_2, \dots, u_{ni}\}$$

(3.3)

Definition 3.3 *Shared Variables*. Shared variables are control variables that appear in the model of at least two subsystems in the problem. Their values should be consistent in the subsystems they appear. They are also called negotiated variables because their values are obtained through a negotiation process. V_{ij} is the set of negotiated variables between partitions i and j, defined by

$$V_{ij} = \{v_1, v_2, \dots, v_{nij}\}$$

(3.4)

Each subsystem *i* is controlled by an MPC controller using:

- the model of the dynamics of subsystem *i* given by Eq. (3.2);
- the measured state $x_i(k)$ of subsystem *i*;
- the exogenous inputs $d_i(k)$ of subsystem *i* over a specific horizon of time;

As a result each MPC controller calculates directly the internal control actions, $u_i(k)$, of subsystem *i*.

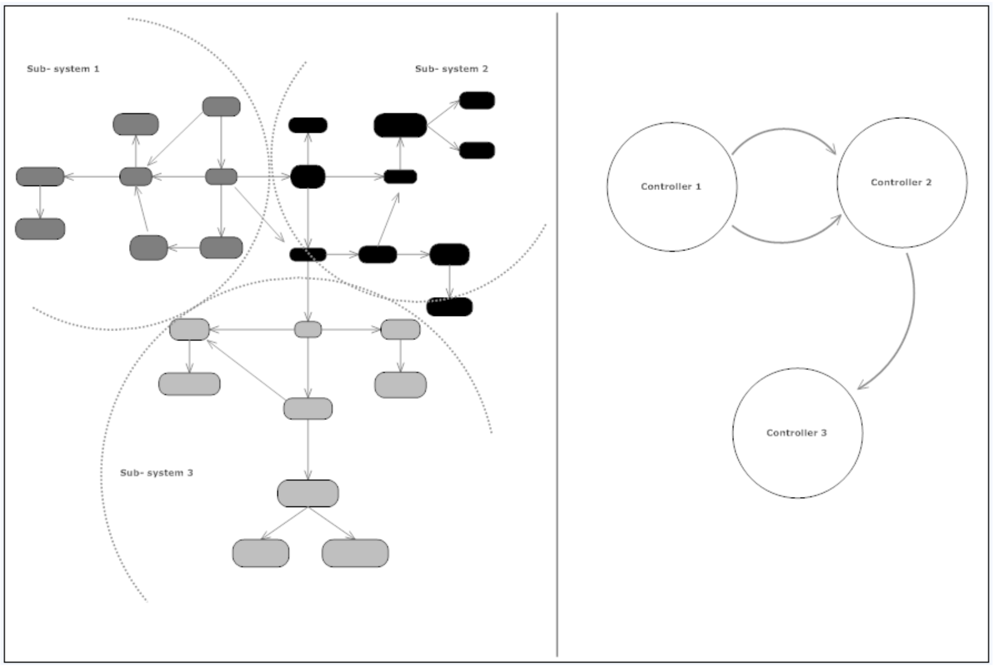


Figure 3.1 The problem of distributed control.

Figure 3.1, on the left, shows a sample system divided into three partitions. Sub-system 1 has two shared variables with sub-system 2 and sub-system 2 has one shared variable with sub-system 3. The relations that represent those variables are shown on the right as lines. The problem consists in optimizing the manipulated variables of the global system using a distributed approach, i.e. with three local control agents that should preserve consistency between the shared variables.

In order to solve the problem described above, a new framework has been developed. This framework comprises a methodology, so called the *MA-MPC methodology* (described in Chapter 5) and the *MA-MPC architecture* described in this chapter. The methodology helps to implement the architecture.

The main idea of this framework is to develop a multi-agent system where the MPC controllers of the LSS partitioned into subsystems above become *MPC agents* that interact with *negotiation agents* in order to solve cooperatively the value of the *shared variables*. According to this concept, the resulting multi-agent system of Figure 3.1 will look like Figure 3.2, where *Negotiator agent 1* will solve the value of two shared variables and *Negotiator agent 2* of one (See relations in Figure 3.1). As can be seen in Figure 3.2 only MPC agents with shared variables among them are communicated. This is made through a bidirectional communication of a negotiator agent with each MPC agent related to the shared variables. Next, the *MA-MPC architecture* and its elements are described.

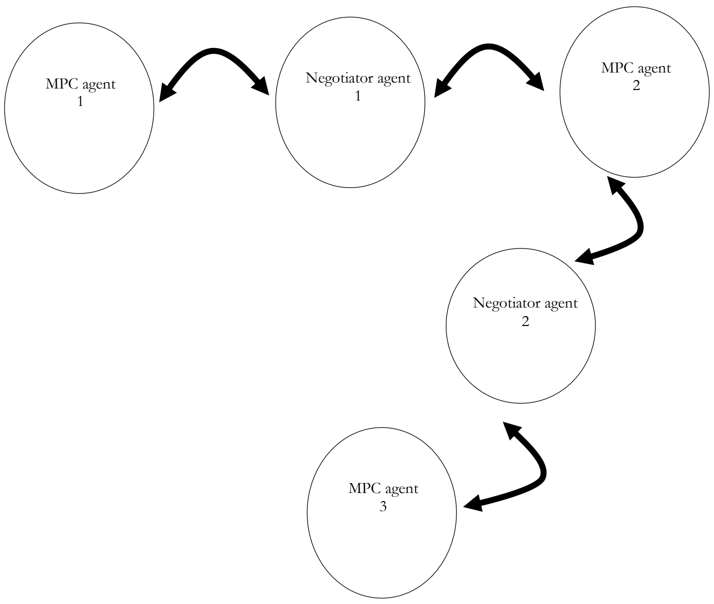


Figure 3.2 Resulting multi- agent system associated to the distributes MPC control of the system presented in Figure 3.1

Definition 3.4 *MA-MPC architecture.* The MA-MPC distributed control architecture is defined as:

$$\gamma = \{M, N, P, W, V, U, b\}$$

(3.5)

where:
M is the set of *MPC agents* and is defined by

$$M = \{M_1, M_2, \dots, M_n\}$$

(3.6)

N is the set of *negotiator agents* and is defined by

$$N = \{n_1, n_2, \dots, n_m\}$$

(3.7)

P is the set of *system partitions* already defined in Eq. (3.1), *W* is the set of *nodes* defined in Eq. (3.6), *V* is the set formed by all sets of *shared variables* in Eq. (3.4), *U* is the set formed by all sets of *internal variables* in Eq. (3.3) and *b* is the Agent platform.

Definition 3.5 *Nodes.* A node is the physical device (commonly a computer) in which the agents are located, they are communicated via some communication infrastructure (LAN, WAN or Internet). *W* is the set of nodes defined by

$$W = \{w_1, w_2, \dots, w_{nw}\}$$

(3.8)

Definition 3.6 *Agent Platform.* The agent platform provides the agents with a homogenous medium to communicate and provides the user a way to manage agents. The agent platform is denoted by *b*. This platform has to be installed and running in all nodes.

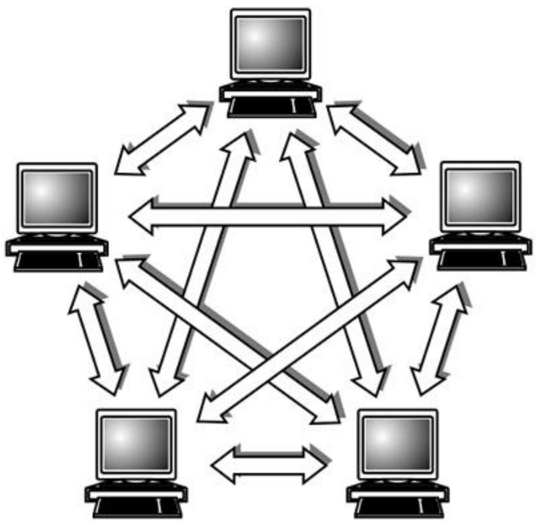


Figure 3.3 Network of five nodes fully communicated.

The main elements of the *MA-MPC architecture* are *MPC agents* and *negotiator agents*. Next, these elements are explained in further detail.

3.2 MPC agents

An *MPC agent* solves an MPC multivariable control problem considering the *internal variables* of its partition and cooperating with one or more negotiator agents to determine the optimal value of the *shared variables*.

Definition 3.7 *MPC agent.* An MPC agent is the entity that is in charge of controlling one specific partition of the system. There is one MPC agent for each system partition (p_i).

Each MPC agent is arranged to cooperate so that the *negotiator agent* solves the optimization of a common goal by means of a reinforcement learning algorithm.

The cooperative behavior of MPC agents is a basic issue in the proposed approach. In order to behave in such a cooperative way, MPC agents implement three actions:

- They provide the data required by the negotiator agent.
- They accept the value(s) provided by the negotiator agent of its shared variable(s).

- They solve the MPC control problem of its partition, adjusting the value(s) of its shared control variable(s) in order to coordinate the solution of the negotiation.

3.2.1 Internal Structure of MPC agents

The internal structure of *MPC agents* has three main elements: *models, an MPC controller, and a communication module* (Figure 3.4). Next, the three main elements will be explained.

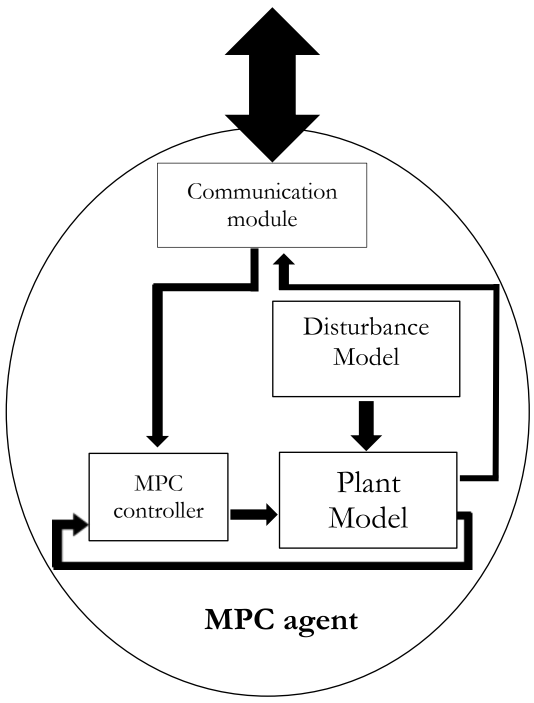


Figure 3.4 Internal structure of MPC agents.

Definition 3.8 *Models.* Plant model and disturbance model are used in order to implement the MPC technique of the MPC agent. They are also involved in the learning process as it will be explained later. The model of each MPC agent is described by a deterministic linear time-invariant model expressed in discrete-time defined in Eq. (3.2).

Definition 3.9 *MPC controller.* A local MPC controller is in charge of the control of each partition P_i, formed by all its internal variables Eq. (3.3), constraints, objective functions, prediction horizon (H_p) and control horizon (H_c).

Definition 3.10 *Constraints*. Physical or security limits of the sensors and actuators. The *Prediction Horizon (H_p)* (Interval of finite future time in which the MPC computes the predictive values by using the model in Eq. (3.2) and *Control Horizon (H_c)* (Interval of finite future time in which the MPC computes the control values by using the model in Eq. (3.2) are also constrains.

Definition 3.11 *Objective function*. It is a function that represents a performance criterion. This function is denoted by:

$$\sum_{i=0}^{H_p} J(i) \tag{3.9}$$

where i is an interval of time, H_p is the prediction horizon and J represents the function criteria.

Definition 3.12 *Communication module*. The communication module is the interface that communicates and synchronizes the MPC agent with other agents (specifically negotiator agents).

3.3 Negotiator agents

Definition 3.13 *Negotiator agent*. A negotiator agent (NA) is the entity that is in charge of determining the value of one or more shared variables between two MPC agents. A negotiator agent exists for every pair of MPC agents that have one or more shared variables in common.

Each negotiator agent determines the optimal value of one or more shared variables in the set V. Each shared variable is solved seeking a global optimum for both MPC agents which are agree to cooperate. The NA carries out its optimization based on the reinforcements given at each step and on the experience obtained. This experience is stored in a knowledge base.

The NA considers the shared variables as belonging to a single problem with a single goal, instead of two different problems with conflicting goals. The negotiator agent solves the optimization problem for that variable and communicates the result to the MPC agents at each sampling time. Then, MPC agents set those values as constraints in their respective internal control variables and solve again the MPC problem associated to its partition. The optimization algorithm of the negotiator agent is based on its experience and on maximizing the reinforcements received at every action taken in the past on similar situations. The internal architecture of a negotiator agent is defined next.

3.3.1 Internal Architecture of negotiator Agents

The internal architecture of the Negotiators agents (Figure 3.5) comprises the following elements: *Communication module, knowledge base* and *behaviors module*. Next, these elements are described in further detail.

The *communication module* of the NA is the analogous of Definition 3.12 in the MPC agent. It deals with the interaction between NA and MPC agents involved in the solution of one or more shared variables.

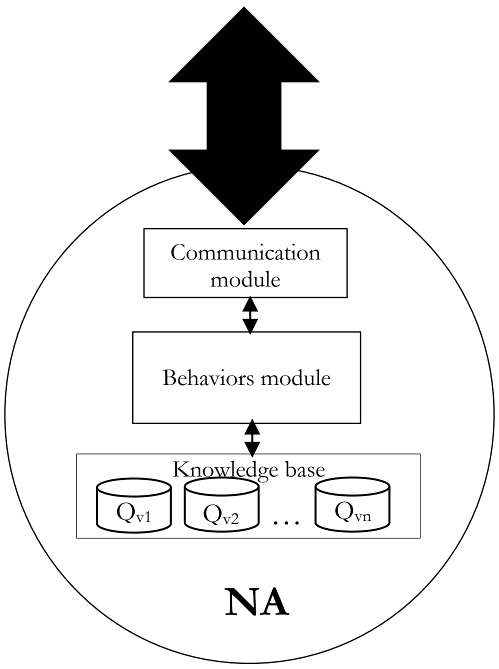


Figure 3.5 Internal structure of NA

Definition 3.14 *Q-table*. A Q-table is a tri-dimensional matrix that represents the knowledge related to one particular shared variable. It maintains the Q-value gained for each possible pair of states (of the MPC agents related to that shared variable) and an action. Each *Q-table* is defined as follows:

$$Q(s_{a1}, u_n, s_{a2}) \tag{3.10}$$

where s_{a1} and s_{a2} are the *states* of two *MPC agents* and u_n is their shared control (*action)*.

Definition 3.15 *Knowledge base*. The Knowledge base of the Negotiator agent is constituted by a set of *Q-tables*, one for each shared variable.

Since the negotiator has to reach the global optimum, it has to observe the state of the agents related to the shared variable, and map it to an action, leading to tridimensional *Q-tables*.

Definition 3.16 *State of an MPC agent (s_a).* A state of an MPC agent is a measure used to describe the state of an MPC agent as a whole.

Considering that an MPC agent can have many state variables (*internal variables*) used by the MPC controllers to calculate its outputs (see Definition 3.9), for the negotiator agent there is just one state variable (s_a) for each MPC agent. This state variable has to be aggregated enough to represent the state of the agent.

In order to build the *Q-table*, the states s_{a1} and s_{a2} are taken as indexes of this table. Moreover, the minimum and maximum values that s_{a1} and s_{a2} can take have to be defined and they need to be discretized. Another index that should be defined is u_n that represents all the possible actions that the agents can take. The values have to be discretized in order to be used in the *Q-table*. The *Q-table* is updated with *Q-values* according to the NA *behavior* with knowledge acquired from the interaction with its environment.

Definition 3.17 *Q-values.* Q-values represent how good an action is for a specific pair of states s_{a1} and s_{a2}, The largest the Q-value is, the better the action is. Q-values are calculated in the behavior module of a NA and updated in the Q-table during a training process.

Definition 3.18 *Behaviors module.* The behaviors module is the core of the NA. All the processes related to negotiation and learning are defined here. Through its behaviors, the NA updates and uses its Knowledge Base.

Chapter 4 describes the learning processes which involve all the definitions introduced above.

Chapter 4. Cooperation and learning.

As it was mentioned in Chapter 3, the main elements of the MA-MPC architecture are MPC Agents (Definition 3.7) and Negotiator Agents (NA) (Definition 3.13). Cooperation and learning are key elements of the NA. In this chapter the learning problem is formulated and the behaviors of the NA are described. Due to LSS are critical systems, the proposed behaviors separates the learning process from control, in order to eliminate the number of learning steps while the NA are controlling and coordinating the system. The *planningByIntruction* and *planningByExploration* behaviors are two *planning* (see section 2.3.1) techniques proposed. But the models used in these techniques are not stochastic or tabular, as the ones commonly used in *planning*. In this work, the models used are the LTI models of the system partitions in charge of MPC agents. Once the knowledge is obtained thru an *off line* training, an exploitation algorithm; *greedy* behavior; is use *in line*, in order to coordinate and optimize de value of *shared variables*. This work proposes, with this combination of techniques, a solution to unfeasibility when applying RL in critical control systems where the number of learning steps required to converge toward an optimal or sub-optimal policy is an issue.

4.1 Cooperation and learning

According to Definition 3.3, shared variables appear in the model of at least two subsystems in the problem and their values should be consistent in the subsystems they appear. This is necessary because when a network is partitioned, some variables appear duplicated in two or more subsystems. This is done in order to provide each MPC Agent involved in the relation with an internal representation of the shared variable. In Figure 4.1, variable *V* is *in MPC agent1* and in *MPC agent2* and Q_{vn} is the representation of *V* in the NA. The Negotiator agent seeks to restore the connections broken in the partitioning problem, connecting what was divided, unifying these duplicate variables in just one as in the original model. Therefore, for the Negotiator Agent, these two control variables are taken as just one.

The philosophy of the negotiation agent is to consider the determination of shared variables not as two different problems with conflicting goals but as one problem with just one goal, a global optimum, like in the centralized approach. This is why in the internal structure of the NA there is one *Q-table* for each shared variable in its *knowledge base*.

Therefore, each shared variable constitutes an optimization problem that it is assigned to the negotiator agent. This particular optimization problem is a *learning problem* and the way of dealing with it corresponds to the *learning approach* of the negotiator.

The *learning approach* is in every component of the negotiator agent, defining the way that it interacts with other agents, in the *role* that it is playing, in its internal processes, its *behaviors* etc. The coordination between subsystems is necessary and learning is the way to make it happen.

4.2 A model driven control and a model driven integrated learning.

The MA-MPC architecture integrates a model driven control (MPC) and a model driven learning process. In order to perform the negotiation of the shared variables, the negotiator agent learns to think globally, by means of an *offline training* where negotiator and MPC agents interact and accumulate meaningful experience. This offline training is made using a model of each subsystem environment computing value functions (*Q-tables*) whose optimality and efficiency are proved in the *experimentation phase*, in order to be used later in the negotiation process. This allows to eliminate iterative communication between agents in the negotiation process, increasing efficiency, decreasing time of response and making a safe *implementation phase*, (see Chapter 5 for the definition of these phases).

Usually, RL systems can be considered as trial-and-error learners. The use of models is a relatively new development in RL systems. In the RL literature, the use of simulated experience generated by a model is called *planning* (Barto, 1998). But in this case, the models used are not stochastic or tabular, as the ones commonly used in *planning*. Here the models used are the models of each system partition (subsystem) p_i, Eq. (3.1) of the MPC agents.

Figure 4.1 shows the integration of the models of MPC agents in the *planning* process. The NA assigns the values of V to the MPC agents. Each MPC agent has its own reference, disturbance model and plant model according to Eq. (3.2). The local MPC takes V as constraints, computes vector c and applies the control action to the *plant model* producing y and e it that feedback y to the MPC controller and e to the NA. e is an error vector that indicates to the NA how good or bad the actions (V) were. In order to evaluate that it is necessary to calculate the state of both MPC agents. This is made based in the cost function of the MPC agents, as for example,

$$s_1 = \sum_{i=0}^{Hp} J(i) = \sum_{i=0}^{Hp} J_x(i) + \sum_{i=0}^{Hp} J_{\Delta u}(i) \tag{4.1}$$

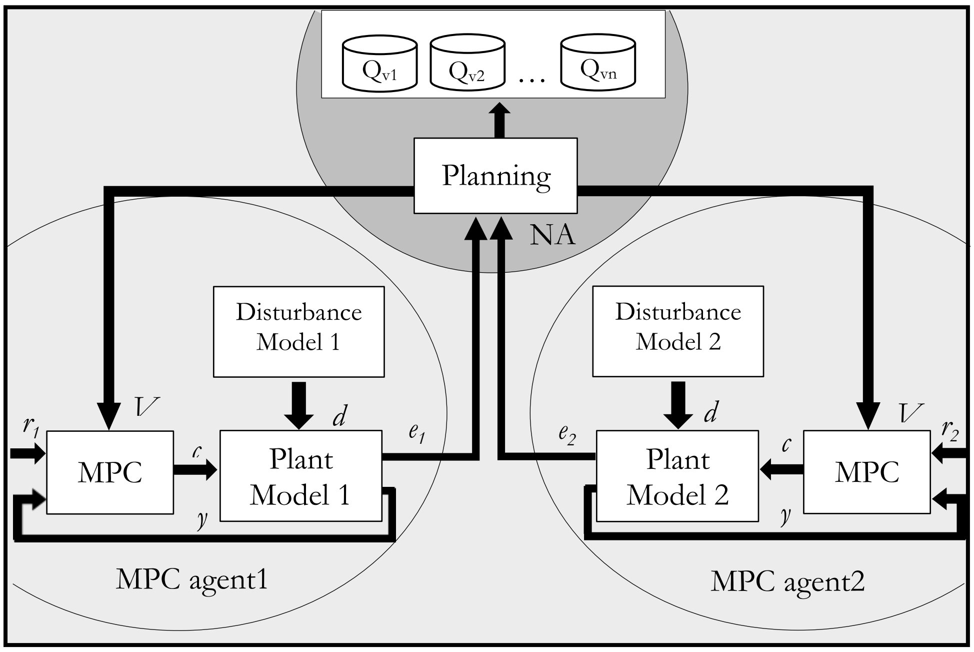


Figure 4.1 Integration of the models of MPC agents in the planning process

$$s_2 = \sum_{i=0}^{Hp} J(i) = \sum_{i=0}^{Hp} J_x(i) + \sum_{i=0}^{Hp} J_{\Delta u}(i) \tag{4.2}$$

where,

$$J_x(i) = \vec{e}^T(i)\, w_x\, \vec{e}(i) \quad \text{and} \quad J_{\Delta u}(i) = \overrightarrow{\Delta u}^T(i) w_{\Delta x} \overrightarrow{\Delta u}(i) \tag{4.3}$$

The *reward* (*r*), is calculated using the states of both MPC agents with the equation:

$$\sigma = \rho - s_1 - s_2 \tag{4.4}$$

where σ represents the *reward r* and ρ is a constant that satisfies:

$$s_1 + s_2 < \rho \tag{4.5}$$

Given that s_1 and s_2 represents the *state* (Definition 3.16) of *MPC agent1* and *MPC agent2* of Figure 4.1 respectively and the state is a sum of quadratic errors (4.1), (4.2), the *reward* (4.4) will be always positive. With a smaller sum of errors the reward will be larger and vice versa.

s_1 and s_2 have to be discretized in order to be use in (4.6) that is the function that updates each *Q-table* where the parameters $\propto$ is a learning rate (see section 2.3.1) and rates past experience.

$$Q(s'_1, a', s'_2) \leftarrow r + \propto Q(s_1, a, s_2) \tag{4.6}$$

The purpose of this three-dimensional matrix is to map the state of *MPC agent 1* (s_1) and the state of *MPC agent 2* (s_2) to a single action. The coordination feature of the NA lies in the fact that, in exploitation, the NA will map to an optimal (or sub-optimal) action every s_1 and s_2 eliminating with this conflicts between MPC agents assigning the value of *shared variables.*

NA uses this simulated experience and updates the Q-*values* in the *Q-tables* (see Definition 3.14), one for each *shared variable* of the vector *V* in order to improve its policy. All this process is implemented through the *PlannigByInstruction* and *PlaningByExploration* behaviors of the NA that will be explained in further detail in Section 4.4.1 and 4.4.2 respectively. In Figure 4.1 the communication module of the agents is omitted in order to clarify how the relation of the models in the agents is made, although all interactions between agents is made through the communication module of the agents.

The integration of RL with the model and MPC in this approach offers high cohesion to the system. The support that the LTI model offers is completely deterministic, descriptive and highly trusted. So, the integration of these techniques coupled by the implementation of the methodology makes the planning process efficient and trusted.

The policy obtained is evaluated in the *experimentation phase* (see section 5.4). The fact that the policy is obtained *offline* is a very important characteristic of this approach due to the critical nature of LSS. The use of a standard trial-and-error technique of RL would make the implementation of this approach unfeasible. If the learning process is driven from real experience in the plant, the system will be unfeasible most of the time at the beginning of the process and the actuators can be damaged. That is why, in this framework, in order to arrive to the *implementation phase* (see Section 5.5), the optimality of the obtained policy has to be tested beforehand.

4.3 Roles of the Negotiator agent.

In the last section it was discussed how a learning technique of the NA called planning is integrated, but the NA also has to optimize (negotiate) and, in some cases, it has to learn and optimize simultaneously. To achieve this, the negotiator agent plays many roles.

Definition 4.1 *Role of an agent.* The role of the agent is the particular function that the agent plays in its environment. The role of the agent determinates the behavior that the agent performs during a specific situation.

In this work, there are four roles defined for the negotiator agent: Learner, Explorer, Negotiator and Adaptive negotiator.

Definition 4.2 *Learner role.* The NA is a learner when it is acquiring knowledge from a teacher in a process performed offline. During this process the learner updates its knowledge base implementing the *PlanningByInstruction* behavior.

Definition 4.3 *Explorer role.* The NA is an explorer when it is acquiring knowledge from his own experience by means of the interaction with its environment in a process performed offline. During this process the explorer updates its knowledge base implementing the *PlannigByExploration* behavior.

Definition 4.4 *Negotiator role.* The NA is a negotiator when it is optimizing. This process is performed online. During this process the negotiator exploits its knowledge base implementing the *Greedy behavior.*

Definition 4.5 *Adaptive negotiator role.* NA is an adaptive negotiator when it is optimizing for a given period of time and exploring new actions the rest of the time. Both processes are performed online. During this processes the negotiator exploits and update its knowledge base implementing the *Soft-max behavior* (Sutton, 1998).

4.4 Behaviors

As it was mentioned before, the role of the agent determines its behavior. In this section, the behavior of each role of the NA is defined. The NA uses four behaviors: *PlanningByInstruction, PlannigByExploration, Greedy* and *Soft-max.* Table 4.1 shows the roles of the NA and their respective behavior. Next, the definition of *behavior* is made.

Role	Behavior
Learner	PlanningByInstruction
Explorer	PlanningByExploration
Negotiator	Greedy
Adaptive	Soft-max

Table 4.1 Roles of the NA and they respective behavior.

Definition 4.6 *Behavior.* It is the way in which the agent interacts with its environment according to its role in order to archive its goal.

In order to fulfil its goal, as it was mentioned before, the NA has to learn first. The main learning behaviors in this work are *PlanningByInstruction* and *PlanningByExploration* behaviors. The difference between these behaviors is the way of choosing the actions and the type of feedback in the learning process. *PlanningByInstruction* applies *leaning by instruction* and *evaluative feedback* (see Section 4.4.1) and *PlanningByExploration* applies *learning by exploration* and *selective feedback* (see Section 4.4.2).

4.4.1 Learning by instruction- evaluation.

In contrast to some IA learning methods, like supervised learning, in this work, the term *instruction* refers to the way in which the action is selected in the learning process, and not to the type of the feedback used. Hereafter the terms planning and learning will be synonyms (because planning is a type of learning and is the learning technique used in this work as it was justified in Section 4.2). So, *PlannigByInstruction behavior* (PBIB) is a learning behavior that implements a specific combination of choosing actions and providing feedback.

The purpose of this learning behavior is to obtain an *optimal policy (Q),* constructing a *knowledge base* (Definition 3.15) based on the evaluation of actions given by a teacher. This teacher has to be a trustable controller, like a centralized MPC or the actions taken by a human expert. These actions are simulated in the model system of MPC agents and the result (states s_{a1} (4.1) and s_{a2} (4.2), is evaluated obtaining a *reward* (4.4) that is used to obtain the new *Q-value* (4.6). *n* iterations are made for the complete control horizon with random initial conditions. This behavior is performed offline in the *training phase* of the MA-MPC methodology described in Chapter 5. Assuming that there is a single negotiation variable, the *PlanningByInstruction behavior algorithm* describes the training algorithm that the NA executes in order to update its *Q-table* by this learning behavior:

Algorithm 2 PlanningByInstruction behavior algorithm.

```
1.  Define $\rho$ that satisfies (4.3), n, $s_{a1} \leftarrow$ random, $s_{a2} \leftarrow$ random, controlHorizon,
    teacherAction (1-control horizon), k=1
2.  loop while iterations $\leq$ n
3.     loop while k $\leq$ controlHorizon
4.        $V_{a1}$ (k) $\leftarrow$ teacherAction (k)
5.        $V_{a2}$ (k) $\leftarrow$ teacherAction (k)
6.        $s_{a1} \leftarrow$ send $V_{a1}$ (k) to MPCagent1 , MPCagent1 set the action $V_{a1}$ (k) and
          calculates its internal variables, apply all the controls (actions) obtained
          (and given) for step k to its LTI model of its partition and calculates $s_{a1}$
          using (4.1).
7.        $s_{a2} \leftarrow$ send $V_{a2}$ (k) to MPCagent2, MPCagent2 set the action $V_{a2}$ (k) and
          calculates its internal variables, apply all the controls (actions) obtained
          (and given) for step k to its LTI model of its partition and calculates $s_{a2}$
          using (4.2).
8.        r $\leftarrow \rho - s_{a1} - s_{a2}$
9.        Q ($s_{a1}$', teacherAction (k)', $s_{a2}$' )$\leftarrow$ r +$\alpha$ Q($s_{a1}$, teacherAction (k), $s_{a2}$)
10        $s_{a1}$'$\leftarrow s_{a1}$
11        $s_{a2}$'$\leftarrow s_{a2}$
12        k=k+1
13     end loop
14     iterations=iterations+1
15  end loop
```

In this algorithm, s_{a1} and s_{a2} represents the *states* of *MPCagent1* and *MPCagent2* (the two MPC agents that share that particular negotiation variable). V_{a1} and V_{a2} are the internal representations of the shared variable in *MPCagent1* and *MPCagent2* (sub-indices *a1* and *a2* respectively) for *k* instant. *teacherAction* is a vector that contains the actions dictated by the teacher for the complete control horizon of the MPC agents, that most be the same for both.

In order to define the number of iterations of the training *n* (step 1) it is necessary to evaluate the *Q (s_{a1}', teacherAction (k)', s_{a2}')* obtained. This is made in an iterative process in the *experimentation phase* of the MA-MPC methodology explained in Chapter 5, the performance analysis and validation process is made after training. There are two methods to perform this analysis: one is comparing the shape of the graphics of the *Q* obtained with different values of *n*, if the shape of *Q* obtained whit 100 iterations its similar than the one obtained with 150 for example, the training can stop, if they are different increase the value of *n* and compare again the last two versions of *Q* obtained.

The other method is to calculate the average absolute error of both parameters in (4.1) of a succession of simulations (using *greedy* behavior, section 4.4.3) using for example different initial conditions, references, demands, etc., increment the value of *n,* calculate the new average as before and compare, if the average absolute errors of both simulations are similar, the training can stop.

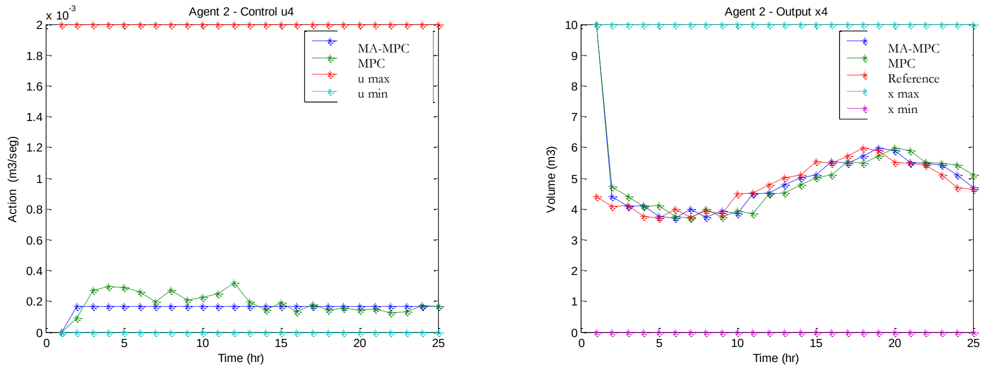


Figure 4.2 On the left, comparison of the control obtained by means of a centralized MPC and decentralized MA-MPC using PBIB. On the right, contrast between outputs of the centralized MPC and MA-MPC system with PBIB.

Figure 4.2 presents the results of the simulation of a shared variable in which *instructed-evaluative* learning was applied. The learning behavior used for the training was *PlanningByInstruction behavior (PBIB).* In this case, the teacher was a centralized MPC version of the system. The NA performed a *learner role.* On the left, controls applied in a 24 hours simulation of a shared variable trained with instructed-evaluative learning after a training of 50 iterations are shown. On the right, the output (volume in a tank) of the corresponding state variable is depicted.

As it can be appreciated on the left part of Figure 4.2, the controls applied by the NA were more constant in time than the centralized MPC, even though the outputs were good enough. In Chapter 6, a case of study that uses the *instructed-evaluative* learning implementing *PlanningByInstruction behavior* will be presented in detail.

4.4.2 Learning by exploration-selection.

Learning by exploration is the main type of learning technique used in RL. It is based on trying random actions from a deterministic and finite set, in order to obtain a feedback that represents how good the taken action was. Learning by exploration in LSS can be a difficult task because of the size and complexity of these systems. The *PlanningByExploration* behavior (PBEB) implements learning by exploration combined with selective feedback. The use of selective feedback reduces drastically the time of training needed in order to obtain an *optimal policy (Q)* and the difficulty to find a good parameterization of the learning process in the *experimentation phase* (Section 5.6).
The purpose of this learning behavior is to obtain an *optimal policy (Q),* constructing a *knowledge base* based on the exploration of a deterministic and finite set of actions. These actions are simulated in the model system and the result (states s_{a1} and s_{a2}) is evaluated and only in case a feasible solution for both agents (*MPCagent1* and *MPCAgent2*) is found, the feedback is

selected for leaning. For those cases, a *reward (r)* is obtained and used to calculate the new *Q-value (Q* (s_{a1}',*a'*, s_{a2}')). *n* iterations are made for the complete control horizon with random initial conditions. This behavior is performed offline in the *training phase* of the MA-MPC methodology described in section Chapter 5. Assuming that there is a single negotiation variable, the *PlanningByExploration behavior algorithm* describes the training algorithm that the NA executes in order to update its *Q-table* by this learning behavior, the name of the variables and the definition of the number of iterations of the training *n* (step 1) are the same than the ones in Algorithm 2.

Algorithm 3 PlanningByExploration behavior algorithm.

```
1.  Define $\rho$ that satisfies (4.3), n, $s_{a1} \leftarrow$ random, $s_{a2} \leftarrow$ random, controlHorizon,
    k=1
2.  loop while iterations $\leq$ n
3.    loop while k $\leq$ controlHorizon
4.      a $\leftarrow$ random (a) $\in A$ $Q(s_1', a, s_2')$
5.      $V_{a1}$ (k) $\leftarrow$ a
6.      $V_{a2}$ (k) $\leftarrow$ a
7.      $s_{a1} \leftarrow$ send $V_{a1}$ (k) to MPCagent1 , MPCagent1 set the action $V_{a1}$ (k) and
        calculates its internal variables, apply all the controls (actions) obtained
        (and given) for step k to its LTI model of its partition and calculates $s_{a1}$
        using (4.1).
8.      $s_{a2} \leftarrow$ send $V_{a2}$ (k) to MPCagent2, MPCagent2 set the action $V_{a2}$ (k) and
        calculates its internal variables, apply all the controls (actions) obtained
        (and given) for step k to its LTI model of its partition and calculates $s_{a2}$
        using (4.2).
9.      if MPCagent1 and MPCagent2 have a feasible solution
10.       r $\leftarrow \rho - s_{a1} - s_{a2}$
11.       Q ($s_{a1}$', a', $s_{a2}$' )$\leftarrow$ r $+\alpha$ Q($s_{a1}$, a, $s_{a2}$)
12.       $s_{a1}' \leftarrow s_{a1}$
13.       $s_{a2}' \leftarrow s_{a2}$
14.     else
15.       $s_{a1}' \leftarrow$ random
16.       $s_{a2}' \leftarrow$ random
17.     end if
18.     k=k+1
19.   end loop
20.   iterations=iterations+1
21  end loop
```

Figure 4.3 presents, the results of the simulation of a shared variable in which *instructed-evaluative learning* was applied. Below three shared variables trained with explorative-selective learning after a training of 50 iterations. Centralized MPC (green) and decentralized MA-MPC (blue), u_{max} (red) and u_{min} (cyan). Above, output (volume in a tank) of a state variable related with the tree shared variables. Centralized MPC output (green), MA-MPC (blue), reference (red), x_{max} (cyan) and x_{min}(purple). The learning behavior used for the training was *PlanningByExploration* behavior. The NA performed a *learner role.*

As it can be appreciated in Figure 4.2 (above), the performance of this behavior is better than one obtained by a centralized MPC controller and the values of the shared variables (below) were more constant in time than the centralized MPC.

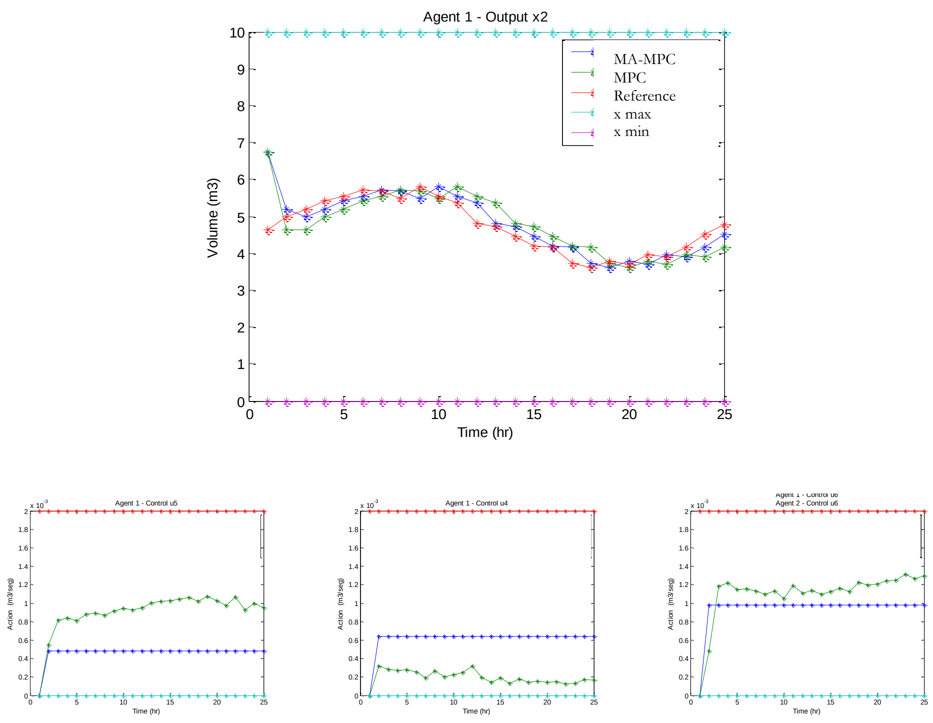


Figure 4.3 simulation of a shared variable in which *instructed-evaluative learning* was applied.

This learning behavior was developed for cases were there was no teacher available, but the combination of explorative-selective learning resulted to be very efficient. The use of selective feedback reduces drastically the time of training needed and, when a full exploration is made (this is when all possible actions are evaluated), an optimal policy is guaranteed. In Chapter 7, a case of study that uses explorative-selective learning implementing *PlanningByExploration behavior (PBEB)* is presented in detail.

4.4.3 Negotiation-Optimization process.

These are processes executed by the NA when it is optimizing on a *negotiator role*. In order to achieve this negotiation-optimization the NA uses its *greedy behavior*. This behavior only can be used after a training phase when the knowledge base is already constructed. Due to LSS are critical systems, the *greedy behavior* is implemented separated from the learning process, in order to eliminate the number of learning steps while the NA are controlling and coordinating the system. Once the knowledge is obtained thru an *off line* training using *planningByIntruction* and/or *planningByExploration* behaviors, an exploitation algorithm; *greedy* behavior; is use *in line*, in order to coordinate and optimize de value of *shared variables*. This combination of techniques provides a solution of the unfeasibility of applying RL in critical control systems where the number of learning steps required to converge toward an optimal or sub-optimal policy is an issue. The algorithm of the greedy behavior is:

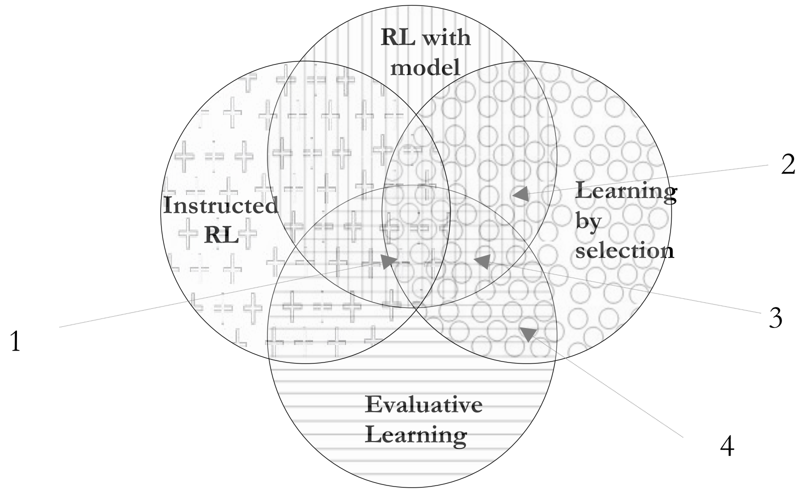
Algorithm 4 Greedy behavior algorithm

```
1.  $Q(s_1, a, s_2)\ \forall\ s \in S,\ a \in A$
2.  observe initial state, $s_1, s_2$
3.  loop
4.      a ←max $a' \in A\ \ Q(s_1', a, s_2')$
        $s_{a1}$ ← send $V_{a1}$ (k) to MPCagent1, MPCagent1 set the action $V_{a1}$ (k) and
        calculates its internal variables, apply all the controls (actions) obtained
        (and given) for step k to its LTI model of its partition and calculates $s_{a1}$
        using (4.1).
        $s_{a2}$ ← send $V_{a2}$ (k) to MPCagent2, MPCagent2 set the action $V_{a2}$ (k) and
        calculates its internal variables, apply all the controls (actions) obtained
        (and given) for step k to its LTI model of its partition and calculates $s_{a2}$ using
        (4.2).
5.      $s_1 \leftarrow s_1'$
6.      $s_2 \leftarrow s_2'$
7.  end loop
```

The NA can also perform an adaptive role, this is an important characteristic of any Multi-agent system. This characteristic is implemented through a *soft-max behavior*. This means that the NA, after having been trained by any of the learning techniques described above, when optimizing, combines the use of the *greedy behavior* for a period of time and the *PlanningByExploration behavior (PBEB)* for the rest of time. This combination is not trivial and has to be properly tested.

4.5 The learning approach in context.

The *negotiation-learning approach* of the negotiator can be beyond the behaviors and roles described in this work. Other roles, behaviors and learning techniques can be extended and applied to the MA-MPC architecture. The techniques described in this chapter have proved efficiency at the experimentation made in this work (see case of studies in Chapter 6 and Chapter 7). They also integrate efficiently the elements of the architecture as a whole. These learning techniques are in the field of RL (Sutton, 1998) and were combined in order to suit the problem. Some of these techniques are: Instructed RL, RL with model, evaluative learning and learning by selection (see Figure 4.4).



▨	1	Instructed RL+RL with model+ evaluative learning +learning by selection.
▨	2	RL with model + learning by selection.
▨	3	RL with model+ evaluative learning + learning by selection.
▨	4	learning by selection + evaluative learning

Figure 4.4 Combination of learning techniques used in the negotiation-learning approach of the NA.

The diagram shows the combination used in the *negotiation-learning approach* described in this chapter. The next table shows the *roles* and *behaviors* described and the corresponding area in the diagram according to the combination of techniques implemented.

Diagram area	Role	Behavior
1	Learner	PlanningByInstruction (PBIB)
3	Explorer	PlanningByExploration (PBEB)
2	Negotiator	Greedy
4	Adaptive	Soft-max

Table 4.2 Relation of roles and behaviors.

Chapter 5. Description of the MA-MPC Methodology.

This chapter describes the MA-MPC methodology. This methodology has been developed in order to properly define and integrate the MA-MPC architecture (see Chapter 4). First attempts to define this methodology can be found in (Javalera V., 2010) where a distributed MPC for a drinking water network was developed using the proposed framework and compared against a centralized MPC controller.

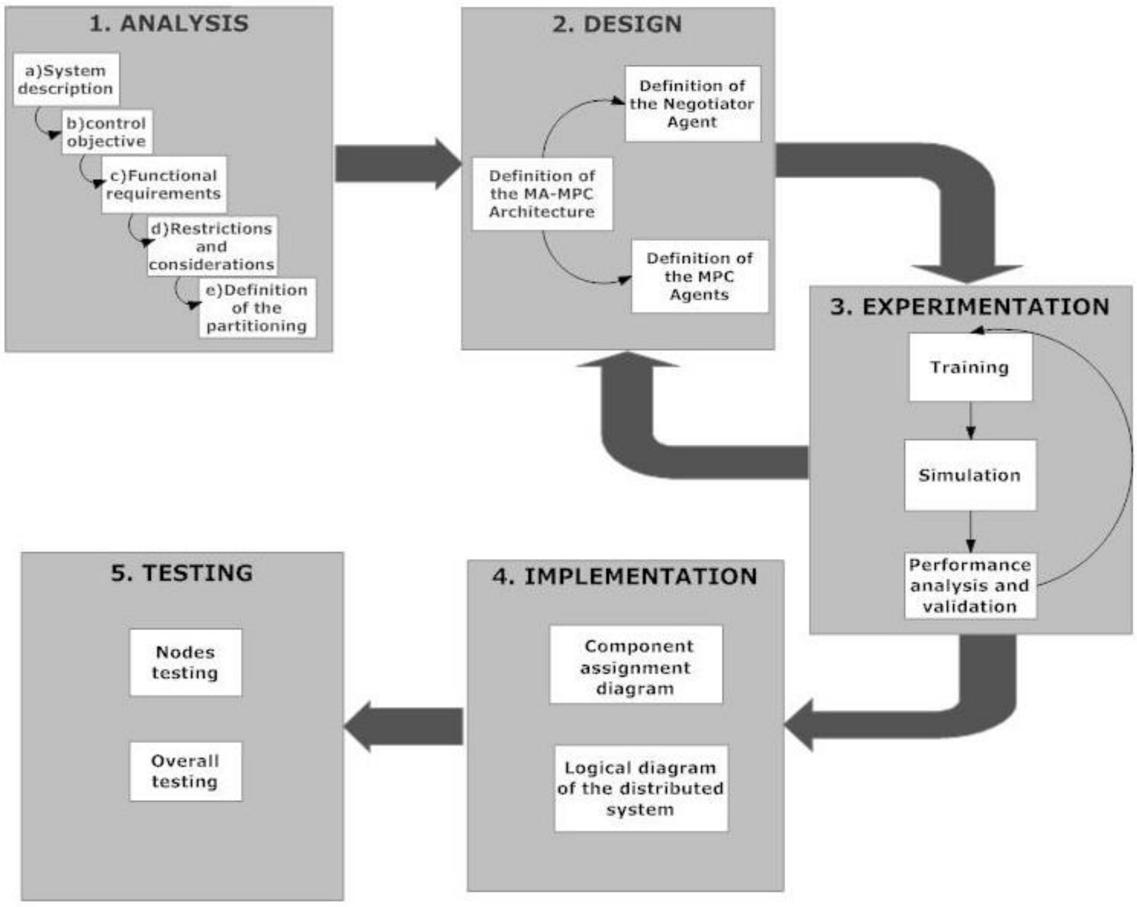

1. ANALYSIS
a)System description
b)control objective
c)Functional requirements
d)Restrictions and considerations
e)Definition of the partitioning
2. DESIGN
Definition of the Negotiator Agent
Definition of the MA-MPC Architecture
Definition of the MPC Agents
3. EXPERIMENTATION
Training
Simulation
Performance analysis and validation
4. IMPLEMENTATION
Component assignment diagram
Logical diagram of the distributed system
5. TESTING
Nodes testing
Overall testing

Figure 5.1 Flow of the MA-MPC methodology process.

The MA-MPC methodology comprises five phases: analysis, design, experimentation, implementation and testing. Figure 5.1 shows the process of the methodology. Each grey box represents a phase and the white boxes represent processes that are part of that phase. As it can be seen, some processes are sequential (as the case of all the processes of the analysis phase), some others can be made in parallel (as the definition of the MPC agent and definition of negotiator agent processes of the design phase), and there are also iterative processes (as. e.g., the processes of the experimentation phase). Next, each phase is described in further detail.

5.1 Analysis phase.

The purpose of the analysis phase (Figure 5.2) is to define the problem and the requirements of the system. It is the basis of all the processes of the MA-MPC methodology. In the analysis phase there are five steps to be defined:

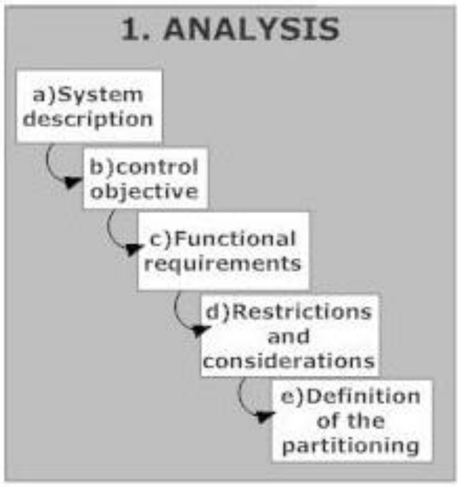

1. ANALYSIS
a)System description
b)control objective
c)Functional requirements
d)Restrictions and considerations
e)Definition of the partitioning

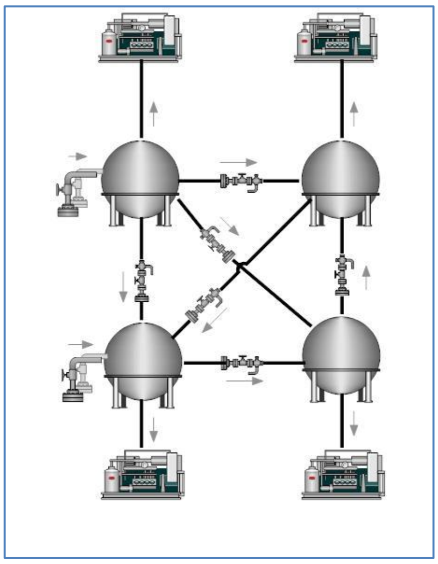
Figure 5.2 Processes of the analysis phase.

- System description.
- Definition of control objectives.
- Definition of functional requirements.
- Definition of restrictions and considerations.
- Definition of the partitioning.

The processes are sequential, each process is the basis for the next one. An explanation of each one is presented below.

5.1.1 System description.

It is a description of the system to be controlled. It should include the new required elements and the ones that will be preserved (if there is a system already in use), the relations between elements, boundaries of the system, etc. The system description also includes a *system diagram* that shows the topology and elements of the system to be controlled in a distributed way (see Figure 5.3 for exemplification purposes). Some important elements of *system diagrams* are: components (e.g., reservoirs), relations (connections), actuators (valves, pumps, etc.).



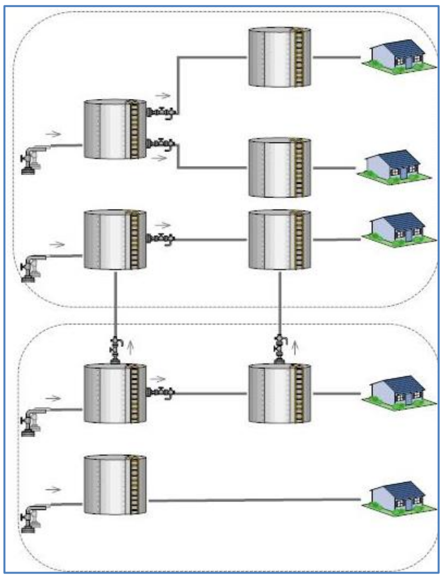


Figure 5.3 Examples of system diagrams

5.1.2 Definition of control objectives.

The control objectives are the desirable criteria that should rule the system. The system can have multiple objectives; for example: minimize the cost of operation, maintain the value of some variables close to some desired set-points, preserve the life of actuators by means of the smooth operation of the actuators, etc.

5.1.3 Definition of functional requirements

In this process all the functions that the distributed control system should be able to do are defined. All its desirable performances will be listed in the *functional requirement table*, where they should be assigned a reference or identification (FR1, FR2, etc.), a name and a description of the requirement.

Requirements are important especially in complex systems because all the efforts of the system development have to be guided by what it is expected to achieve. They are also helpful in the communication between the development team and especially between the team and final users.

5.1.4 Definition of restrictions and considerations.

In this section, the constraints (Definition 3.10) and other considerations, as e.g. safety levels for reservoirs and other elements should be defined. It is important to define all the minimum and maximum values that sensors and actuators can have. The physical restrictions of every element of the system has to be considered. The exact values of these restrictions will be needed in the design phase.

5.1.5 Definition of the partitioning.

Once the system is defined and the objectives, requirements and restrictions are clear; it is time to define the partitioning of the network. There can be different partitioning criteria: geographical, structural, etc. Here, the partitioning will be defined by a *general partitioning diagram* (see Figure 5.4 as an example).

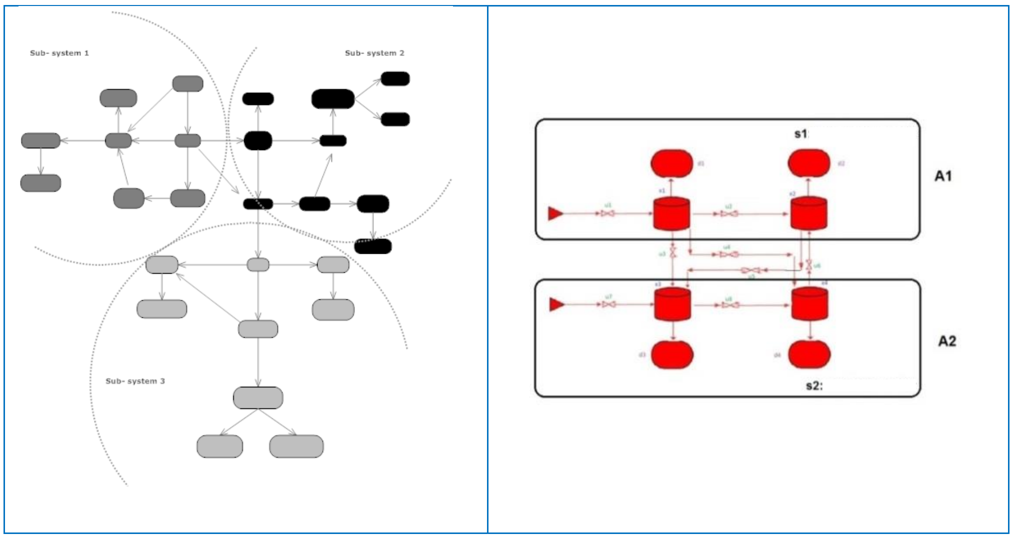


Figure 5.4 Example of a general partitioning diagrams.

A *general partitioning diagram* has to show all the elements of a *system diagram* but specifying the partitioning by grouping the elements of each new partition. The name of relations, or control elements between subsystems and the name of subsystems (partitions) have to be shown clearly in the diagram.

5.2 Design phase

The design phase (Figure 5.5) comprises three processes: definition of the MA-MPC architecture, definition of the MPC agents and definition of negotiator agents. The definition of the MA-MPC architecture is made first, once defined the architecture, the definition of the MPC agents and NA can be made. The whole problem formulation is done in this phase. This problem formulation is based on the information gathered in the *analysis phase.*

5.2.1 Definition of the MA-MPC Architecture.

In order to apply the MA-MPC Architecture to the problem, first, the *architecture of the system diagram* (Figure 5.6) based on the *general partitioning diagram* has to be constructed according to the following directions:

- Circles represent agents.
- Make one MPC agent for each partition represented in the *general partitioning diagram.*
- Identify in the *general partitioning diagram*, partitions with relations among them.

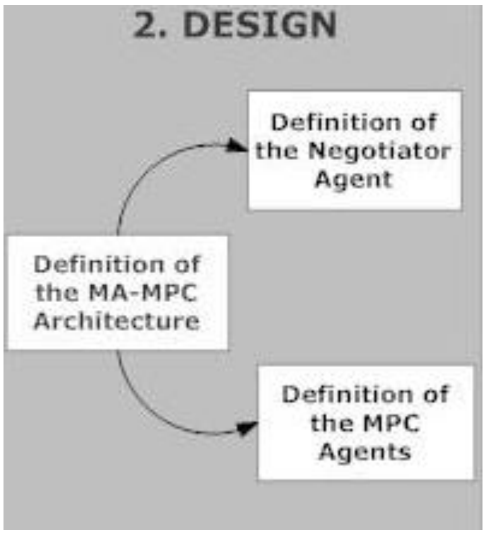

2. DESIGN
Definition of the Negotiator Agent
Definition of the MA-MPC Architecture
Definition of the MPC Agents

Figure 5.5 Design phase process.

- Place a NA between the corresponding MPC agents that have these relations.
- Represent the relations using arrows.

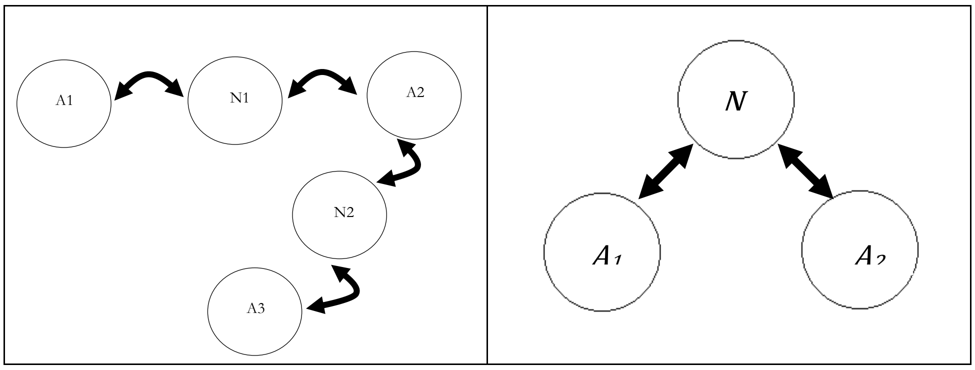


Figure 5.6 Examples of architecture of the system diagrams.

Once the architecture of the system diagram is made, the next step is to construct the *relations diagram*. *The relations diagram* shows the relations between MPC agents (The NA are not shown). The MPC agents are placed and each relation is represented with arrows respecting the direction of the relation (see Figure 5.7 as example).

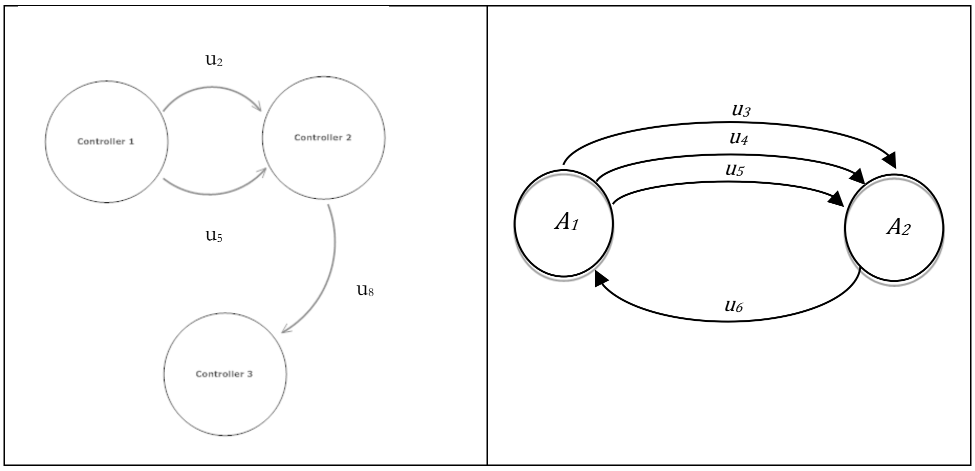


Figure 5.7 Examples of relation diagrams.

Element	Definition
M	{set of MPC agents}
N	{set of Negotiator agents }
P	{X, U, E, D}
W	{set of nodes}
V	{sets of V's}
U	{sets of U´s}

Table 5.1 Table γ

Next, based on the *architecture of the system diagram* and the *relations diagram,* identify the elements of the MA.MPC architecture (Definition 3.4) using Table 5.1.

$$\gamma = \{M, N, P, W, V, U, b\} \tag{5.1}$$

where: M is the set of MPC Agents, N is the set of Negotiator Agents, P is the system partitions, W is the set of nodes, V is the set formed by all sets of shared variables, U is the set formed by all sets of Internal Variables and b is the Agent platform.

Next define the elements of the *MPC agent definition table* (Table 5.2) for each MPC agent.

M_n
$p_n = \{X_{Mn}, U_{Mn}, E_{Mn}, D_{Mn}, V_{Mn}\}$
$X_{Mn} = \{\}$
$U_{Mn} = \{\}$
$E_{Mn} = \{\}$
$D_{Mn} = \{\}$
$V_{Mn} = \{\}$

Table 5.2 MPC agent definition table.

At this point, an important step is to check that the partitioning of the plant leads to a complete set of partitions. This is accomplished verifying that all the state variables in the problem are considered in one partition and all the control variables belong to the set of shared or internal variables. The following relation has to be verified:

$$P = \{p_1 \cup p_2 \cup p_n\} \tag{5.2}$$

5.2.2 Definition of the MPC Agents.

Now is time to define the elements of the internal structure of MPC agents. At this point, the elements of *P* have been defined for each MPC agent in the *MPC agent definition table*, such that, the elements defined in the analysis phase are the basis of the construction of MPC agents.

In order to define the *MPC controller* the constraints (Definition 3.10) and objective functions (Definition 3.11) for each MPC agent have to be defined.

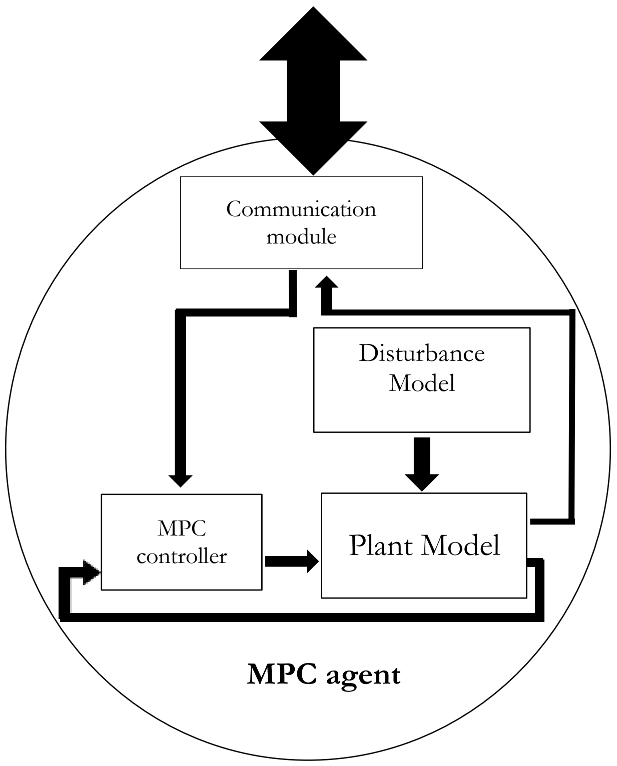


Figure 5.8 Internal structure of the MPC agents.

In order to define the *plant model* (see Figure 4.1) and the *disturbance model,* the *agent definition table, the general partitioning diagram* and the *relation diagram* are used to obtain:

$$x_i(k+1) = A_i x_i(k) + B_{u},u_{ui}(k) + B_{d,i} d_i(k)$$
$$y_i(k) = C_i x_i(k) + D_{u,i} u_i(k) + D_{d,i} d_i(k)$$

(5.3)

where variables *x, y, u* and *d* are the state, output, input and disturbance vectors, respectively; *A, B, C* and *D* are the state, output, input and direct matrices, respectively. Subindexes *u* and *d* refer to the type of inputs in the matrices model, either control inputs or exogenous inputs (disturbances).

For the configuration of the *communication module* it is necessary to construct an *internal correspondence table* for each MPC agent. This table is used to track the name of the variables between the centralized and decentralized models and diagrams.

5.3.3 Definition of the Negotiator Agents

The definition of the internal elements of the Negotiator agents (Figure 5.9) is made using the *architecture of the system diagram,* the *relations diagram*, the *table* γ and each set V of the *MPC agent definition tables.*

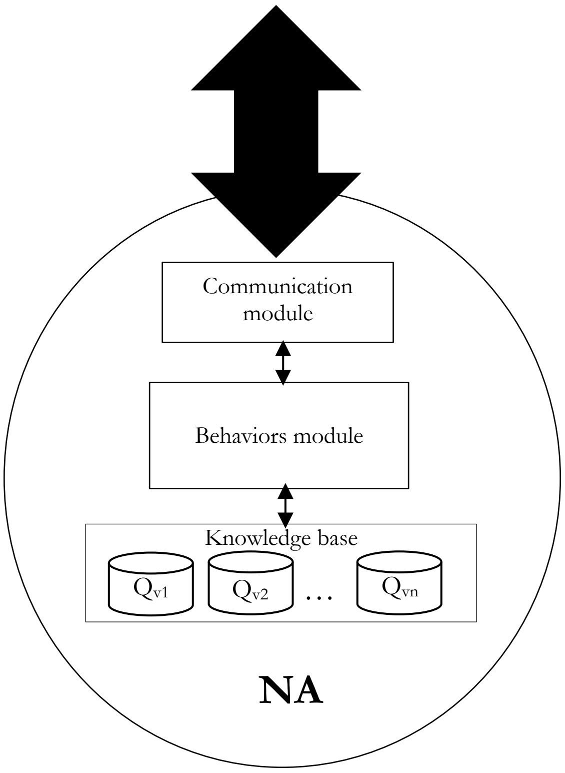


Figure 5.9 Internal structure of NA

The communication module of each NA also needs an *internal correspondence table* to help finding the shared variables under the name that gives each subsystem.

Q_n	Q_1	Q_2	...	Q_n
V_n				
U_{a1}				
U_{a2}				

Table 5.3 Internal correspondence table of NA

The behaviors module was defined in Chapter 4. The *knowledge base* (Definition 3.15) is defined by means of the *Q-tables* (Definition 3.14) which in turn needs to define the discretized *states* (Definition 3.16) and *actions* of the *Q-tables.*

$$Q(s_{a1}, u_n, s_{a2}) \tag{5.4}$$

where S_{a1} and S_{a2} are the *states* of the two *MPC agents* related and u_n is the associate *action*.

5.4 Experimentation phase

The experimentation phase has three iterative processes. It is also an iterative phase along with the design phase (Figure 5.1). This is because it can be necessary to adjust parameters or, even more, make changes in the design of the agents in order to improve the results obtained in the *simulation* and *performance analysis and validation* processes.

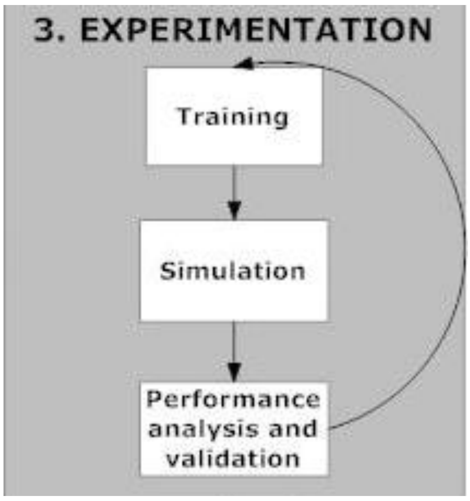


Figure 5.10 Experimentation phase.

The *training process* is an off-line iterative procedure to set up the negotiator agents knowledge bases (see Chapter 4). The agents' shared variables are assigned according to the agent behavior. On training, the behaviors used are *PlanningByInstruction (PBIB)* (see Section 4.4.1) and *PlanningByExploration (PBEB)* (Section 4.4.2).

The *simulation process* uses the knowledge obtained in the *training process* to simulate the performance of the large scale system in a decentralized way. The shared variables of each MPC agent are assigned according to *Greedy* behavior (see Section 4.4.3). The performance is analyzed and validated in the *performance analysis and validation process,* where many iterations are made until the performance meets the objectives and all the functional requirements listed in the analysis phase are achieved.

5.5 Implementation phase.

The implementation phase aims to integrate the MA-MPC system to the real world. Before that, it is necessary to make a *component assignment diagram* in order to specify how the physical components of the network (hardware) will be distributed; and a *logical diagram of the distributed system,* in order to assign each agent and knowledge base to a specific computer. The distribution and component assignment should fulfill the requirements of the analysis phase.

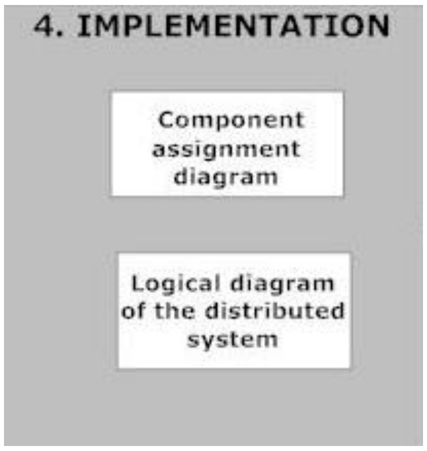

4. IMPLEMENTATION
Component assignment diagram
Logical diagram of the distributed system

Figure 5.11 Implementation phase.

5.6 Testing

Before the system release, some testing has to be done. The test should comprise *node testing* in order to evaluate de performance of a specific node and its assigned agents and an *overall testing* in order to evaluate the performance of the complete system and the communications between remote nodes and agents. The tests have to evaluate how well the requirements of the functional requirements table were accomplished.

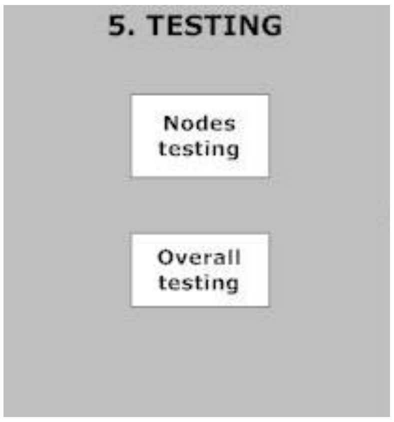


Figure 5.12 Processes of testing phase

Chapter 6. Learning by instruction, an application of the MA-MPC Architecture

In this chapter, the MA-MPC framework is applied to a small water distribution network composed of four tanks with multiples interdependencies among them for illustrative purposes. This example assesses the behavior and efficiency of the framework when strong dependencies between shared and internal variables are present. A step by step application of the methodology described in Chapter 5 is shown. The learning method performed in this example is instructed learning, which means that the *negotiator agent (NA)* will learn from a teacher, in this case a centralized MPC controller, the behavior of the *shared variables* (by the PBIB). On the experimentation phase, illustrative results are shown.

As it was explained in Chapter 5, the MA-MPC methodology comprises five phases (see Figure 6.1). In this example the first three phases are going to be developed and explained (analysis, design and experimentation). Chapter 6 will show a different training using PBEB, in the methodology process this chapter represents an iteration of the phase three.

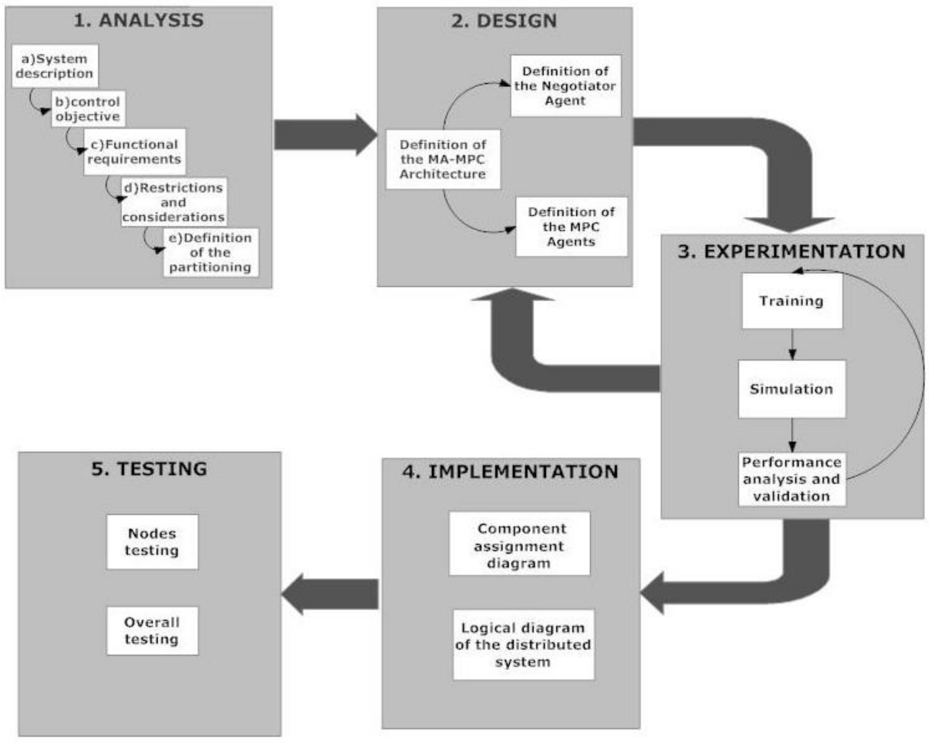

1. ANALYSIS
a)System description
b)control objective
c)Functional requirements
d)Restrictions and considerations
e)Definition of the partitioning
2. DESIGN
Definition of the Negotiator Agent
Definition of the MA-MPC Architecture
Definition of the MPC Agents
3. EXPERIMENTATION
Training
Simulation
Performance analysis and validation
4. IMPLEMENTATION
Component assignment diagram
Logical diagram of the distributed system
5. TESTING
Nodes testing
Overall testing

Figure 6.1 MA-MPC methodology processes

6.1 Analysis

Phase one of the flow of the processes shown in Figure 6.1 is analysis. The purpose of this phase is to define the problem and the requirements of the system. Figure 6.2 shows the processes of the analysis phase.

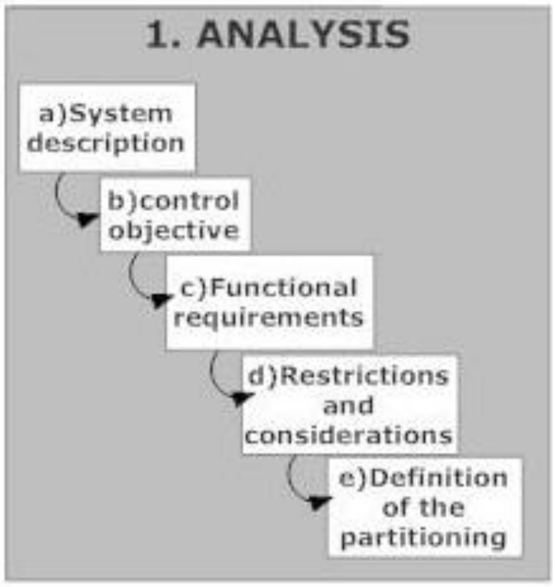

1. ANALYSIS
a)System description
b)control objective
c)Functional requirements
d)Restrictions and considerations
e)Definition of the partitioning

Figure 6.2 Processes of the Analysis phase

Next, each process of the analysis of the four tanks with multiple interdependences problem is defined.

a) System description.

The optimization of the water distribution network of the Figure 6.3, in a distributed manner, is required. The partitioning of the network will obey a geographical criterion, so there will be two partitions, north and south. The tanks x_1 and x_2 will belong to the north sector where a local control is required. The tanks x_3 and x_4 will belong to the south sector, with its corresponding local controller.

There are two supply sources and four demand points, one for each tank. Typically the demands have a sinusoidal behavior throughout the day that try to emulate the actual demand behavior. The system shall operate in a distributed way but looking for global optimum in the controlled tank levels, satisfying the demand points of both subsystems, and avoiding collisions or conflicts among them.

It is expected that the performance of the tank levels follow a reference variable in time, but without performing drastic actions in the actuators. That is, control to be applied on the actuators should be smooth, in order to prevent actuator breakdowns.

b) Control Objective
The target control is defined as follows:

For each tank (x_1, x_2, x_3, x_4) there is a given reference that describes the desirable behavior of the levels of these tanks. These levels will be achieved through the manipulation of the control variables (u_1, u_2,… , u_8) with minor variations over time.

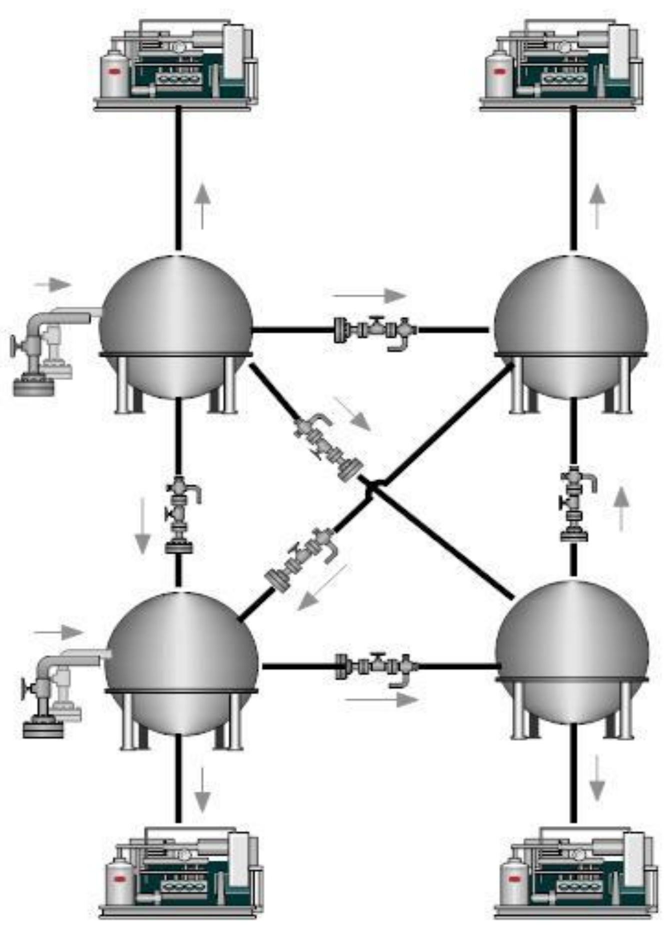


Figure 6.3 System diagram of the four tanks with multiple independences problem.

c) Functional requirements

The functional requirements are obtained from the *system description*. For the four tanks system, seven requirements were defined.

Req No.	Name of the requirement.	Description.
FR1	Type of partitioning.	Geographical criterion. Two partitions, north and south are required.
FR2	Distributed control.	One controller for each partition.
FR3	Tracking.	The behavior of tank levels should follow a reference variable in the time given.
FR4	Smooth control.	Control actions should increase / decrease in small quantities.
FR5	Avoid conflicts and collisions.	Avoid conflicts and collisions between subsystems.
FR6	Satisfy demands.	All demands have the same priority.
FR7	Global optimization	Seek the global optimality of the system.

Table 6.1 Functional requirements table of the four tanks with multiple dependencies problem

d) Restrictions and considerations.

The following restrictions and considerations were abstracted from the s*ystem description* and from the *system diagram* of the four tanks with multiple dependences problem (Figure 6.3)

- The demands are considered measurable disturbances. Typically, demands have a sinusoidal-like behavior throughout the day.
- The network of this example does not contain nodes.
- Each tank has a minimum and maximum volume.
- Each control variable has a minimum and maximum value.

e) Definition of the partitioning.

Taking into account the functional requirements FR1 and FR2, the partitioning of the network is defined as follows:

$$P = \{p_1, p_2\}$$

(6.1)

where p_1 and p_2 are described by a deterministic linear time-invariant model that is expressed in discrete-time as follows

$$\begin{gathered} x_i(k+1) = A_i x_i(k) + B_{u,i} u_i(k) + B_{d,i} d_i(k) \\ y_i(k) = C_i x_i(k) + D_{u,i} u_i(k) + D_{d,i} d_i(k) \end{gathered}$$

(6.2)

where variables *x*, *y*, *u* and *d* are the state, output, input and disturbance vectors, respectively; *A*, *B*, *C* and *D* are the state, output, input and direct matrix, respectively. Subindexes *u* and *d* refer to the type of inputs the matrix model, either control inputs or disturbances. Control variables are classified as internal or shared.

Systems matrices for p_1 are

$$A_1 = \begin{array}{c} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array} \begin{array}{c} x_1 \\ x_2 \end{array}$$

(6.3)

$$B_1 = \begin{array}{cccccc|c} u_1 & u_2 & u_3 & u_4 & u_5 & u_6 & \\ 1 & -1 & -1 & -1 & 0 & 0 & x_1 \\ 0 & 1 & 0 & 0 & -1 & 1 & x_2 \end{array}$$

(6.4)

$$B_{d,1} = \begin{array}{cc|c} x_1 & x_2 & \\ -1 & 0 & d_1 \\ 0 & -1 & d_2 \end{array}$$

(6.5)

while system matrices for p_2 are

$$A_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{array}{c} x_3 \\ x_4 \end{array}$$

(6.6)

$$B_2 = \begin{array}{cccccc|c} u_7 & u_3 & u_5 & u_8 & u_4 & u_6 & \\ 1 & 1 & 1 & -1 & 0 & 0 & x_3 \\ 0 & 0 & 0 & 1 & 1 & -1 & x_4 \end{array}$$

(6.7)

$$B_{d,2} = \begin{array}{cc} x_3 & x_4 \\ \end{array}$$
$$B_{d,2} = \left[\begin{array}{cc} -1 & 0 \\ 0 & -1 \end{array}\right]\begin{array}{c} d_3 \\ d_4 \end{array}$$

(6.8)

In these matrices, the corresponding name of the variable (as it appears in the general partitioning diagram, Figure 6.4) is indicated.

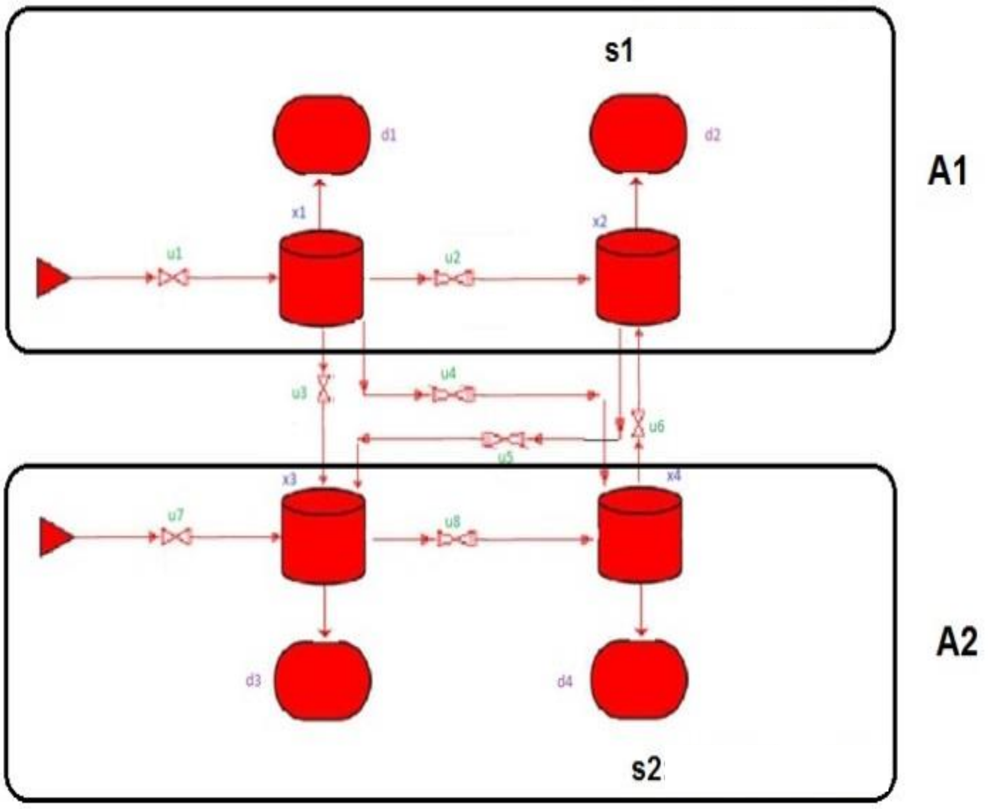


Figure 6.4 General partitioning diagram.

6.2 Design.

a) Definition of the MA-MPC Architecture.

According to Eq. (6.1), two partitions have been defined. Each partition is assigned to an MPC agent, M_1 for p_1 and M_2 for p_2. These two agents have four shared variables and the negotiation of the value of these variables will be in charge of the *NA* named N_1. Figure 6.6 shows the general structure of the MA-MPC architecture of this problem.

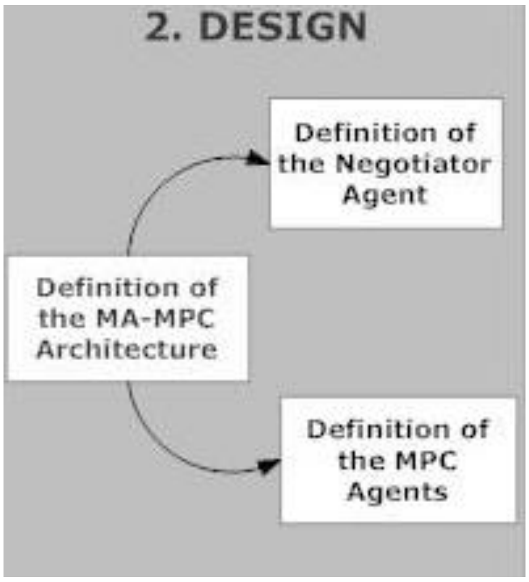

2. DESIGN
Definition of the Negotiator Agent
Definition of the MA-MPC Architecture
Definition of the MPC Agents

Figure 6.5 Processes of the design phase

Figure 6.7 shows the *shared variables* between M_1 and M_2 as well as the direction of the flow. The variables u_3, u_4 and u_5 will provide flow to M_2 and u_6 to M_1.

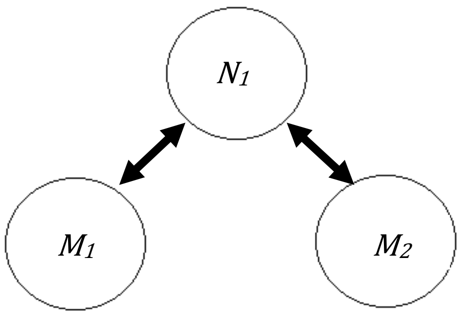


Figure 6.6 General architecture of the system diagram.

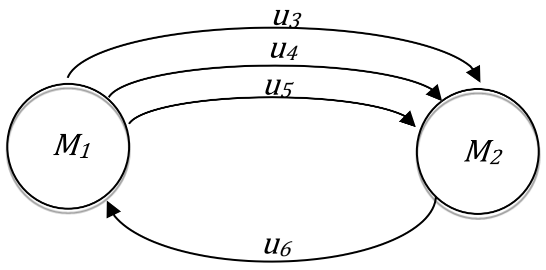


Figure 6.7 Relation diagram of the four tank system.

According to Definition 3.4, the MAMPC distributed control architecture is defined as:

$$\gamma = \{M, N, P, W, V, U, b\}$$

(6.9)

where: *M* is the set of MPC agents, *N* is the set of negotiator agents, *P* is the set of system partitions, *W* is the set of nodes, *V* is the set formed by all sets of shared variables, *U* is the set formed by all sets of control variables and *b* is the agent platform. Next, each of these variables is defined.

Variable	Definition
M	$\{M_1, M_2\}$,
N	$\{N_1\}$,
P	$\{X, U, E, D\}$,
W	$\{W_1, W_2\}$,
V	$\{V\}$,
U	$\{U_1, U_2\}$

Table 6.2 Table γ

where:

$X = \{X_{M1}, X_{M2}\}$, $U = \{U_{M1}, U_{M2}, V\}$, $E = \{E_{M1}, E_{M2}\}$, $D = \{D_{M1}, D_{M2}\}$ and $V = \{V_{M1}, V_{M2}\}$

(6.10)

E is the set of node matrices. In this case there are no nodes in M_1 or M_2, and that is why E_{M1} and E_{M2} are empty. D is the set of disturbance matrices, in this case the demands. This variable is related to the variable B_d of the LTI model of the plant in Eq. (6.5) and Eq. (6.8) and in *the plant model definition* shown below.

Defining MPC Agents M_1 and M_2 as follows

M_1 $p_1 = \{X_{M1}, U_{M1}, E_{M1}, D_{M1}, V_{M1}\}$	M_2 $p_2 = \{X_{M2}, U_{M2}, E_{M2}, D_{M2}, V_{M2}\}$
$X_{M1} = \{x_1, x_2\}$ $U_{M1} = \{u_1, u_2\}$ $E_{M1} = \{\emptyset\}$ $D_{M1} = \{d_1, d_2\}$ $V_{M1} = \{u_3, u_4\}$	$X_{M2} = \{x_3, x_4\}$ $U_{M2} = \{u_7, u_8\}$ $E_{M2} = \{\emptyset\}$ $D_{M2} = \{d_3, d_4\}$ $V_{M2} = \{u_5, u_6\}$

Table 6.3 Definition of MPC Agents M_1 and M_2

W is the set of physical control stations that the system will have. Two control stations were assigned obeying to the functional requirement FR2. The multi-agent platform on which the system will operate will be Jade (Pokahr A., 2013).

Verifying the *completeness property:*

$$P = \{p_1 \cup p_2\} \tag{6.11}$$

$$\{x1, x2, x3, x4, u1, u2, u3, u4, u5, u6, u7, u8, d1, d2, d3, d4\} = \{X_{M1}, X_{M2}, U_{M1}, U_{M2}, V_{M1}, V_{M2}, \emptyset, D_{M1}, D_{M2}\} \tag{6.12}$$

Therefore p_1 and p_2 are a *complete set of partitions* of the system.

b) Definition of MPC agents.

Plant model definition.

According to Eqs. (6.3) to (6.8), the plant model of the *M₁* and *M₂* are the following:

Name of the agent: *M₁* Type of agent: Agent MPC	Name of the agent: *M₂* Type of agent: Agent MPC
$A_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$A_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
$B_1 = \begin{bmatrix} 1 & -1 & -1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 1 \end{bmatrix}$	$B_2 = \begin{bmatrix} 1 & 1 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & -1 \end{bmatrix}$
$B_{p,1} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$	$B_{p,2} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$

Table 6.4 Plant model of M_1 and M_2

Cost function

Both MPC agents (M_1 and M_2) have the same cost function. The cost function was designed in order to fulfill functional requirements FR3 and FR4 (see Table 6.1). FR3 requires that the behavior of tank levels should follow a reference variable in the given time. FR4 requires a smooth control that is achieved by forcing that the control actions increase/decrease in small quantities. So, the cost function needed should minimize the error of the output (level of tanks) with respect to the given reference and minimize de variation of control actions applied. The cost function for this problem is:

$$\sum_{i=0}^{Hp} J(i) = \sum_{i=0}^{Hp} J_x(i) + \sum_{i=0}^{Hp} J_{\Delta u}(i) \tag{6.13}$$

where,

$$J_x(i) = \vec{e}^T(i)\, w_x\, \vec{e}(i) \tag{6.14}$$

$$J_{\Delta u}(i) = \overrightarrow{\Delta u}^T(i) w_{\Delta x} \overrightarrow{\Delta u}(i) \tag{6.15}$$

The first term of the cost function J_x in Eq. (6.13) considers the sum of the quadratic errors (e) of the state variables (x) with respect to the state set-points for all tanks in the i instant for the complete prediction horizon H_p. In the second term, $J_{\Delta u}$in Eq. (6.13) considers the sum of the control effort applied for all control variables (u) of the sub-system for the complete prediction horizon H_p.

As a result the aggregated value of $J(i)$ gives a scalar that takes into account both objectives state J_x and $J_{\Delta u}$ in the prediction horizon H_p. w_x and $w_{\Delta x}$ are weights that are used to establish the priority of each objective, being selected to satisfy $0 \leq w_x \leq 1$ and $0 \leq w_{\Delta x} \leq 1$, where 1 is more important and 0 not important.

Since $J_x(i)$ is related to the variation of the states around the set-points and $J_{\Delta u}(i)$ is related to the variation of the control variables, the resulting values of the term related to $J_x(i)$ will be much bigger than the resulting values of the term related to $J_{\Delta u}(i)$. So, in order to achieve a fair prioritization of these objectives, a normalization is needed.

Constraints.

Next, the physical restrictions of the state variables (*x*) and control variables (*u*) are shown. The restrictions of *x* correspond to the minimum and maximum volume in tanks and are measured in m^3. The restrictions of *u* represent the minimum and maximum flow in actuators and are measured in m^3/seg.

X_{M1}	x_1	x_2
Min.	0	0
Max.	10	10

Table 6.5 State variables restrictions of set X_{M1}

X_{M2}	x_1	x_2
Min.	0	0
Max.	10	10

Table 6.6 State variables restrictions of set X_{M2}

U_{M1}

	u_1	u_2	u_3	u_4	u_5	u_6
Min.	0	0	0	0	0	0
Max.	0,0020	0,0050	0,0020	0,0020	0,0020	0,0020

Table 6.7 Control variables restrictions of U_{M1}

U_{M2}

	u_1	u_2	u_3	u_4	u_5	u_6
Min.	0	0	0	0	0	0
Max.	0,0020	0,0020	0,0020	0,0050	0,0020	0,0020

Table 6.8 Control variables restrictions of U_{M2}

Internal correspondence.

In order to preserve the relation between subsystems, the MA-MPC architecture overlaps the sets of *U* and *V*. In this case this overlapping is shown in Figure 6.8.

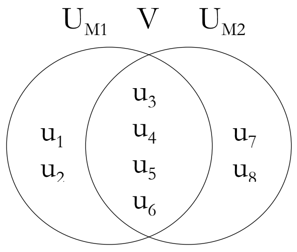


Figure 6.8 Overlapping of *U* and *V* sets of the system.

Due to u_3, u_4, u_5 and u_6 are shared variables, a copy of each one has to be in the three sets but under different names, that is why it is necessary to use internal correspondence tables. The state variables do not overlap but they also change their name, as it is shown below.

M_1		
X_{M1}	x_1	x_2
Centralized system	x_1	x_2

Table 6.9 Internal correspondence of state variables between X_{M1} and the centralized system

M_1						
U_{M1}	u_1	u_2	u_3	u_4	u_5	u_6
Centralized system	u_1	u_2	u_3	u_4	u_5	u_6

Table 6.10 Internal correspondence of control variables between U_{M1} and the centralized system. Shared variables are highlighted.

M_2		
X_{M2}	x_1	x_2
Centralized system	x_4	x_5

Table 6.11 Internal correspondence table of state variables between X_{M2} and the centralized system

M_2						
U_{M2}	u_1	u_2	u_3	u_4	u_5	u_6
Centralized system	u_7	u_3	u_5	u_8	u_4	u_6

Table 6.12 Internal correspondence table of control variables between M_2 and the centralized system. Shared variables are highlighted.

c) Definition of the Negotiator.

Once the MPC agents are defined the NA has to be designed. For the problem addressed here with two MPC agents, M_1 and M_2, there is just one NA needed between them. This NA will have to negotiate the value of four variables. Figure 7.9 shows the resulting internal architecture of the NA. There can be seen that the NA communicates bi-directionally trough the communication module. The negotiation-learning module is comprised of the behaviors implemented, in this case PBIB and Greedy (see 4.4.1 and 4.4.3). The knowledge base consists of four tables (Q_1, Q_2, Q_3, Q_4), one for each *negotiation variable.*

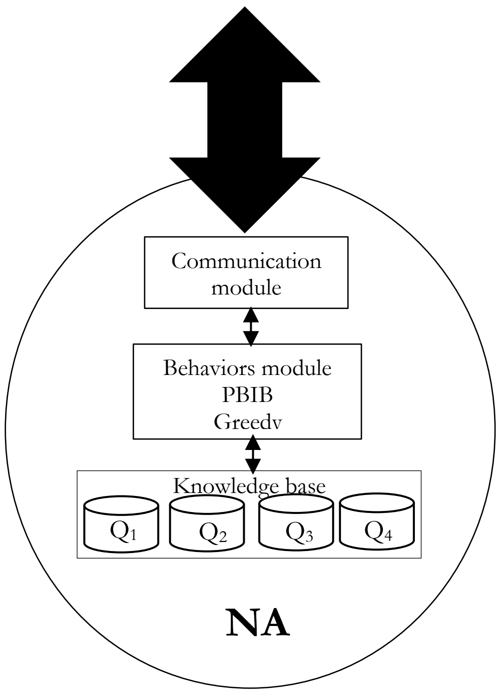


Figure 6.9 Negotiator's structure

Table 6.13 the correspondence between each *Q-table* and the shared variables of the centralized system (*V*) is shown as well as the corresponding internal control variables for M_1 and M_2 (U_{M1} and U_{M2} respectively)

Q_n	Q_1	Q_2	Q_3	Q_4
V	u_3	u_4	u_5	u_6
U_{M1}	u_3	u_4	u_5	u_6
U_{M2}	u_2	u_5	u_3	u_6

Table 6.13 Internal correspondence

One important issue in the design of the NA is the formulation of the negotiation-learning problem and the definition of the *Q-tables*, structurally and logically. Structurally for the addressing problem each *Q-table* is defined as follows:

$$Q(s_{a1}, u_n, s_{a2}), s_{a1}, s_{a2} \in \{0 - 200\}, u_n \in \{0 - 100\} \tag{6.16}$$

The purpose of this three-dimensional matrix is to map the state of *M_1* (s_1) and the state of *M_2* (s_2) to a single action. In this case, there are 200 possible states defined for s_1 and s_2 and 100 possible actions.

The negotiation feature of the NA lies in the fact that, in exploitation, the NA will map to an optimal (or sub-optimal) action every s_1 *and* s_2.

The logical definition of the NA is related to the way in which the states (in this case s_1 and s_2*)* are defined. In order to make this important definition, it is necessary to find a measure, aggregated enough, to define the global state of the corresponding MPC agent.

In this case the definition of the states has been defined as follows:

$$s_1 = \sum_{i=0}^{Hp} J(i) = \sum_{i=0}^{Hp} J_x(i) + \sum_{i=0}^{Hp} J_{\Delta u}(i) \tag{6.17}$$

$$s_2 = \sum_{i=0}^{Hp} J(i) = \sum_{i=0}^{Hp} J_x(i) + \sum_{i=0}^{Hp} J_{\Delta u}(i) \tag{6.18}$$

where,

$$J_x(i) = \vec{e}^T(i)\, w_x\, \vec{e}(i) \quad \text{and} \quad J_{\Delta u}(i) = \overrightarrow{\Delta u}^T(i) w_{\Delta x} \overrightarrow{\Delta u}(i) \tag{6.19}$$

Table 6.14 some particular parameters values in the implementation of the NA, that are considered in the illustrative example, are shown.

Parameters
$w_{\Delta x} = w_x = 1$
$U_{X1\,\mathrm{min}}=0$
$U_{X2\,\mathrm{max}}=0.0020$

Table 6.14 Parameters of the NA

s_1 and s_2 have to be discretized. This is made as follows

$$S_n \leftarrow 100\frac{s_n}{s_{n\,max}} \tag{6.20}$$

for both s_1 and s_2, where s_{max} is the maximum value that the state of the agent can have (before discretization).

Reward calculation

The *reward* (*r*), was calculated with the equation:

$$\sigma = \rho - s_1 - s_2 \tag{6.21}$$

where σ represents the *reward r* and ρ is a constant that satisfies:

$$s_1 + s_2 < \rho \tag{6.22}$$

Given that s_1 and s_2 represents a sum of quadratic errors (see Eq. (6.17) to (6.19)), the *reward* will be always positive. With a smaller sum of errors the reward will be larger and vice versa.

Eq. (6.23) is the function that updates each Q-table where the parameters α rates past experience and is set to 0.5 in the illustrative experiments

$$Q(s'_1, a', s'_2) \leftarrow r + \propto Q(s_1, a, s_2) \tag{6.23}$$

6.3 Experimentation

Once all agents are designed, it is time to define some experiments to illustrative how the proposed approach works. As it was mentioned at the beginning of this chapter, the type of learning implemented is *instructed learning* (described in Chapter 5). *Instructed learning* is an easy and efficient way to train the NAs. The training is made offline and no exhaustive training is needed. It is a good option of training, when a centralized solution of the system is available. The learning behavior used in this example (also defined in Chapter 4) is *planningByInstruction (PBIB)*. In this case, the teacher is a centralized MPC version of the system. The first part of the experimentation phase, according to MA-MPC methodology is *training*.

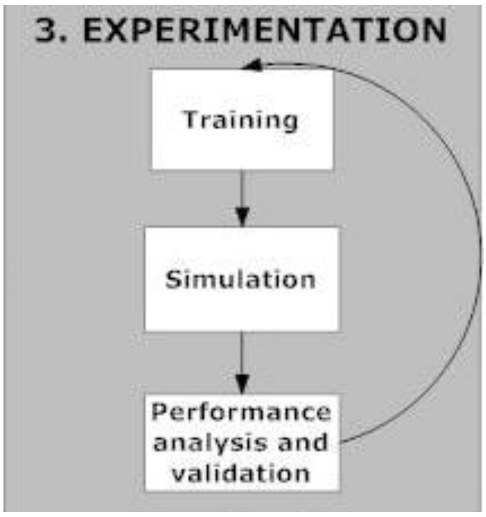
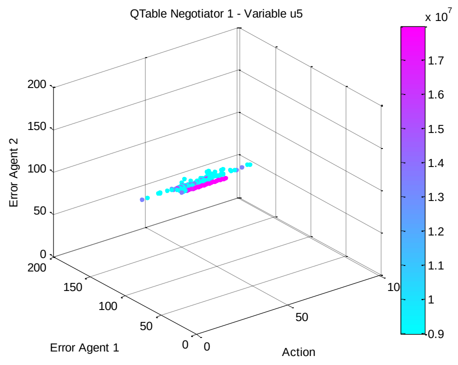


Figure 6.10 Process of experimentation phase.

a) Training
In the training phase of this example, the *planningByInstruction* behavior (PBIB, Algorithm 2 of section 4.4.1) was implemented and executed 300 times (line 1 of the algorithm PBIB, *n*=300), for the complete control horizon (line 4, 24 hrs.) of agents M_1 and M_2. Random initial conditions were set for each complete horizon. During the training, the *Q-table* for the shared variable was filled with the *Q-values* calculated for each state visited. the value of s_1 and s_2 is calculated by it respective MPC agent using Eqs. (6.17) and (6.18), the value is discretized using Eq. (6.20).

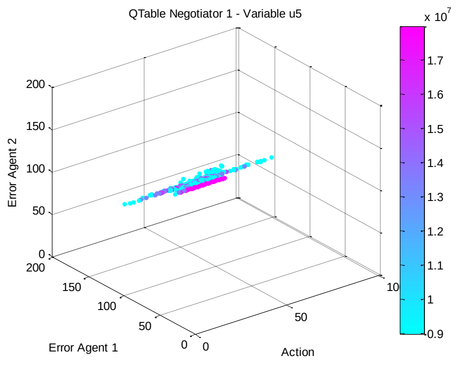
Figure 6.11 shows a representation of the *Q-values* calculated in different phases of the training of the variable u_5. The *Q-table* contrast the error of M_1 and M_2 (or the discretize state of each agent) with the action taken.

In order to use only positive errors, in the Figure 6.11, errors range from 0 to 200. Negative errors range from 0 to 99, 100 corresponds 0 and from 101 to 200 are range the positive errors. Actions are ranging from 0 to 100. As it can be appreciated in the figure, the states visited in this training tend to be denser near the optimum state (100). This is because all the actions were dictated by the teacher, the centralized system. Making a comparison between sub-figures (a), (b), (c) and (d) of the Figure 6.11, it can be seen that the *Q-values* cloud is spreading on the axis of the actions and becomes denser as the training progresses.



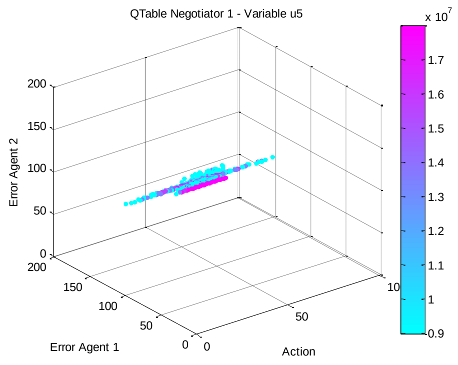
Q-table Negotiation variable u_5 training of 50 iterations

(a)



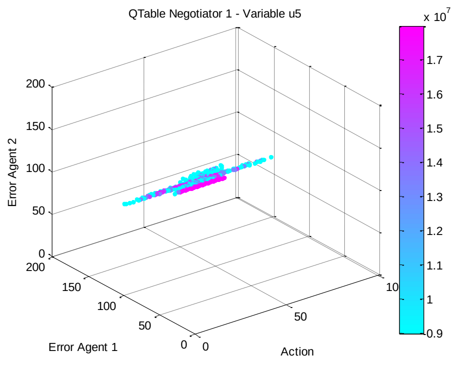
Q-table Negotiation variable u_5 training of 100 iterations

(b)



Q-table Negotiation variable u_5 training of 200 iterations

(c)



Q-table Negotiation variable u_5 training of 300 iterations

(d)

Figure 6.11 Different phases of the training using PBIB of the variable u_5

It is important to notice that the only random factor in this training (using PBIB) are the initial states of the M_1 and M_2. The fact that in this training *instructed learning* is used makes it fast and efficient. The *Q-values* stored in these *Q-tables* represents meaningful and evaluated experience (because of the accumulation of the rewards).

It can be noticed that between section c and d of the Figure 6.11, there is not much difference. This is one of the factors that can show that no more iterations are needed. Additionally, the results of the exploitation phase are necessary in order to determinate that the training phase is finished. Similar results are obtained for the rest of the *q-tables*

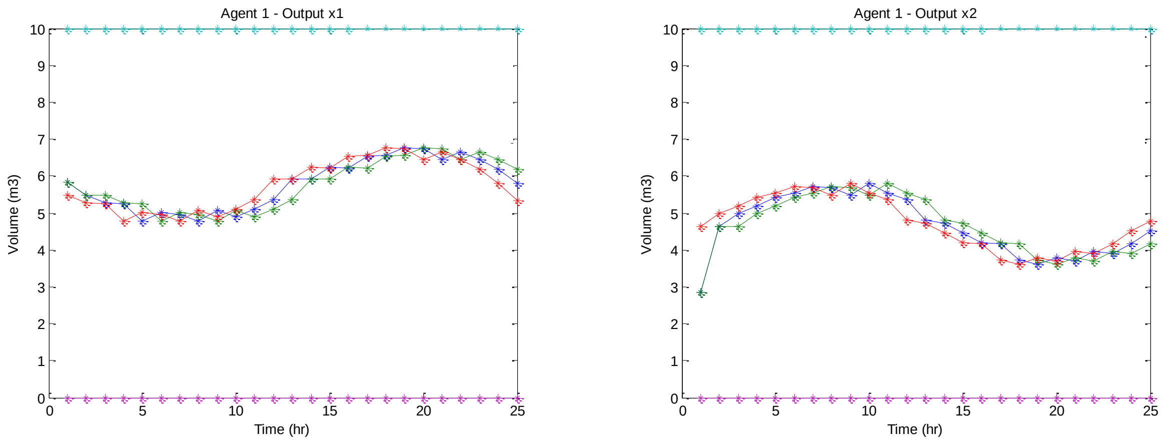
A training based on PBIB can be also used as a good start (or seed) before a *non-instructed learning* technique.

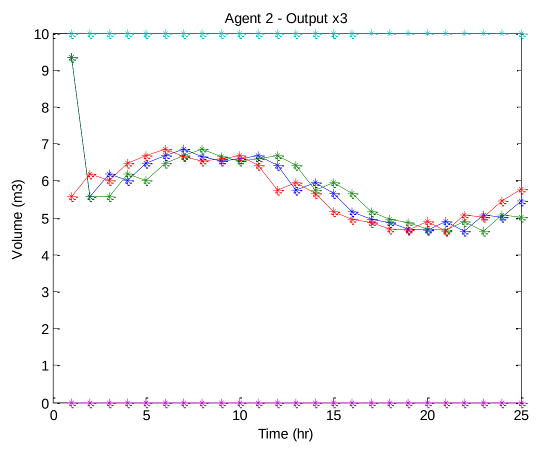
b) Simulation.

As it was mentioned before, in order to know if the training phase is finished it is necessary to evaluate the *Q-tables* making test and exploiting. In order to do that, the *greedy behavior* (Section 4.4.3) is implemented using the Algorithm 4 *Greedy* behavior algorithm of section 4.4.3.

This algorithm observes the state of the MPC agents s_1 and s_2 (in a discretized way) and maps it to the action that maximizes the accumulated Q-value. Figure 6.12 shows the result of a simulation in the exploitation phase. The outputs (the level in tanks) of the four tanks with multiple dependences problem are shown (see Figure 6.1 and Figure 6.2).

Figure 6.12 compares the output given by the proposed framework, the MA-MPC architecture using PBIB, the centralized solution and the reference. Many simulations were made in to prove the efficiency of the learning performed.





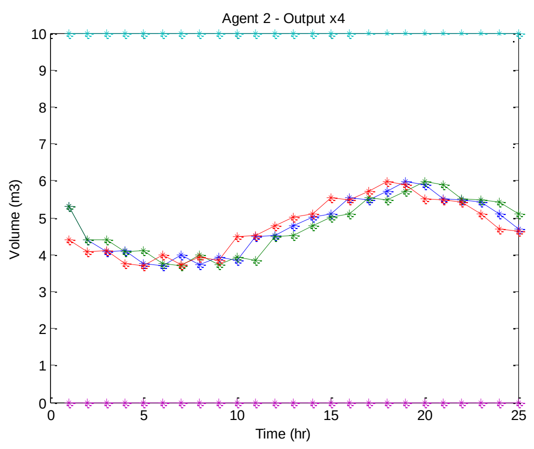
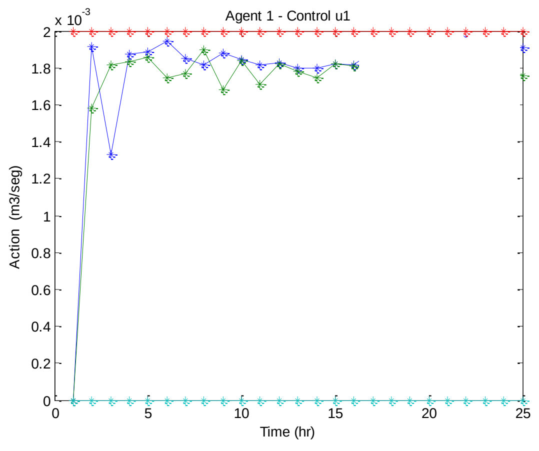
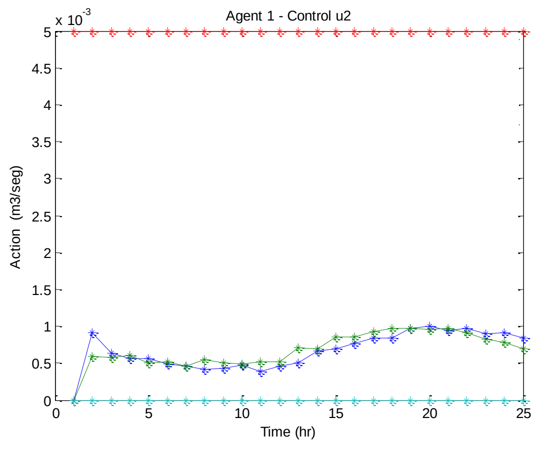


Figure 6.12 Contrast between outputs of centralized MPC (green) and MA-MPC using PBIB (blue) outputs, red line is the reference, cyan x max, purple x min.

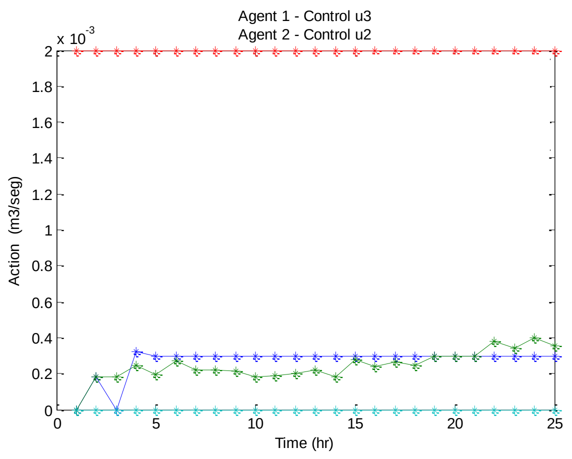
Figure 6.13 shows the resulting actions applied in the simulations shown above. Sections (c), (d), (e) and f of this figure show *shared variables*, while sections (a), (b), (g) and (h) show *internal variables* of the MPC agents M_1 and M_2, respectively. As it can be noticed, the actions calculated by the NA (the shared variables) vary less over time, as it was required in FR4 (see Table 6.1). This is archived without sacrificing performance.



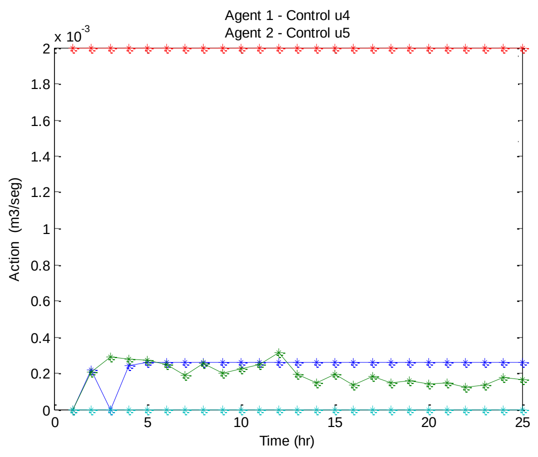
(a)



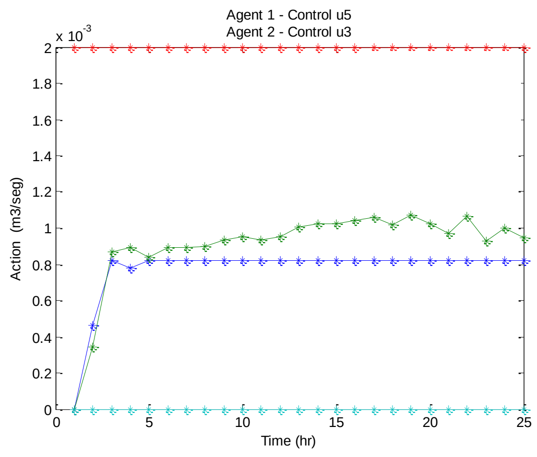
(b)



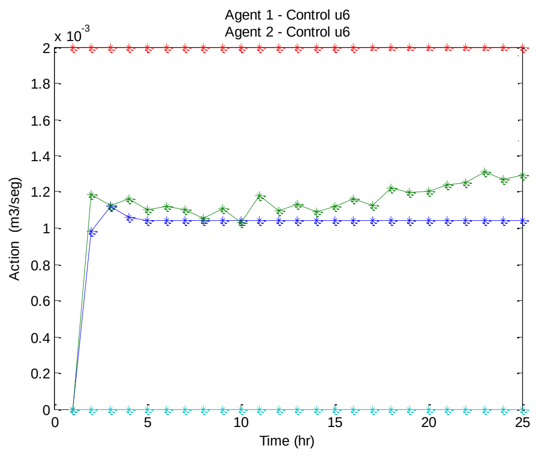
(c)



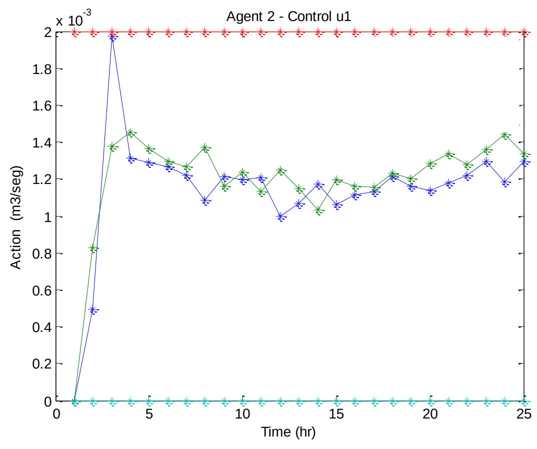
(d)



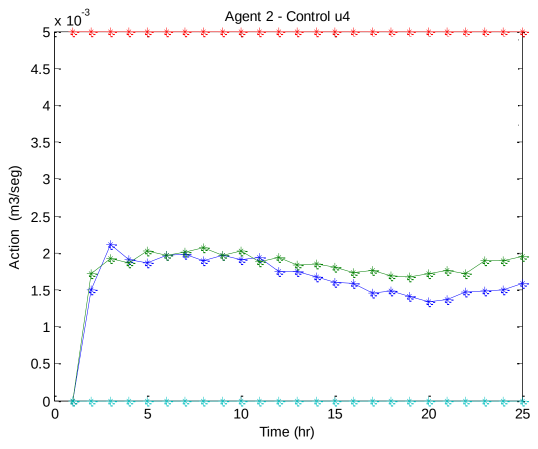
(e)



(f)



(g)



(h)

Figure 6.13 Actions (*u*´s) applied by the MA-MPC using PBIB (blue) and the centralized (green) solution in a simulation of the four tanks with multiple dependences problem. Red line *u* max, cyan *u* min.

c) Performance analysis and validation.

Table 6.15 shows the average absolute error of the output of 30 simulations. The first column was calculated with a training of 50 iterations, next ones with 100 iterations, 200 and 300 iterations. The sum of the error M_1 and error M_2 is provides the total error. It can be seen how the error in the MA-MPC system (the first three) decreases as the iterations of the training progresses. Also it can be noticed that between 200 and 300 iterations there is not much difference in the error. The analysis of this table and the differences between the resulting *Q-tables* (see Figure 6.11) is useful to establish when the training is completed. The results shown in the Table 6.15 show that the MA-MPC system using instructed learning by implementing the *PlanningByInstruction* behavior (PBIB) has a better performance than the centralized MPC solution from iteration 200.

J_e	**50 it**	**100 it**	**200 it**	**300 it**
Error M_1	62,36	24,04	18,07	17,14
Error M_2	60,11	24,23	17,20	17,37
Total error MA-MPC (PBIB)	122,47	48,27	35,27	34,49
Total error Centralized MPC	45,91	44,08	45,04	44,71

Table 6.15 Average of absolute error between increasing iterations during training with PBIB

Table 6.16 shows the accumulative $\boldsymbol{\Delta u}$ objective applied by the MA-MPC and the centralized MPC solution in 30 simulations. The first column was calculated with a training of 50 iterations, next ones with 100, 200 and 300 iterations.

$J_{\Delta u}$	**50 it**	**100 it**	**200 it**	**300 it**
Centralized MPC	4,666e-05	6,333e-05	5,000e-05	6,333e-05
MA-MPC	0,0140833	0,0153533	0,0116933	0,0155466

Table 6.16 Accumulative $J_{\Delta u}$ between increasing iterations during training

6.4 Conclusions

The results of this example shows that a system with multiples dependences between its components can be governed efficiently using distributed controllers and, even more, it can increase its performance using the MA-MPC architecture implementing instructed learning by the PBIB behavior.

It can also be observed that the actions calculated by the NA (the shared variables) vary less over time without sacrificing performance. But the accumulative control effort is minor compared with the centralized MPC.

Other experiments have been carried varying the weights of the parameters $w_{\Delta x}$ and w_x of Eq. (6.19) presented in Table 6.14. Making the same changes in the teacher (the centralized MPC) and performing a new training, the NA adapts to the new parameterization providing similar results than ones obtained with the weights presented in Table 6.14.

Chapter 7. Learning from exploration-validation, an application.

In Chapter 6, the four tanks with multiple interdependencies problem was successfully solved applying instructed learning, but, what happens when there is no teacher available? Is exploration-validation a better method of training? This chapter proposes an extension of the MA-MPC methodology to the case that no teacher is available. The achieved performance is illustrated with the same four tank problem already introduced in Chapter 6 allowing the comparison with the teacher based approach. Figure 7.1 shows the process flow of the extended methodology. The starting point of this chapter is the finishing point of Chapter 6, i.e., the final process of the experimentation phase. From there, we will go back to the design phase, i.e. to the *definition of the NA*. This will change the core of the NA (that is, the behaviors module) in order to try a learning *by exploration-validation* behavior, PBEB (presented in Section 4.4.2). This means that the *negotiator agent (NA)* will learn from its own experience the behavior of the *shared variables* in a selective way. Finally, promising results are shown on the experimentation phase.

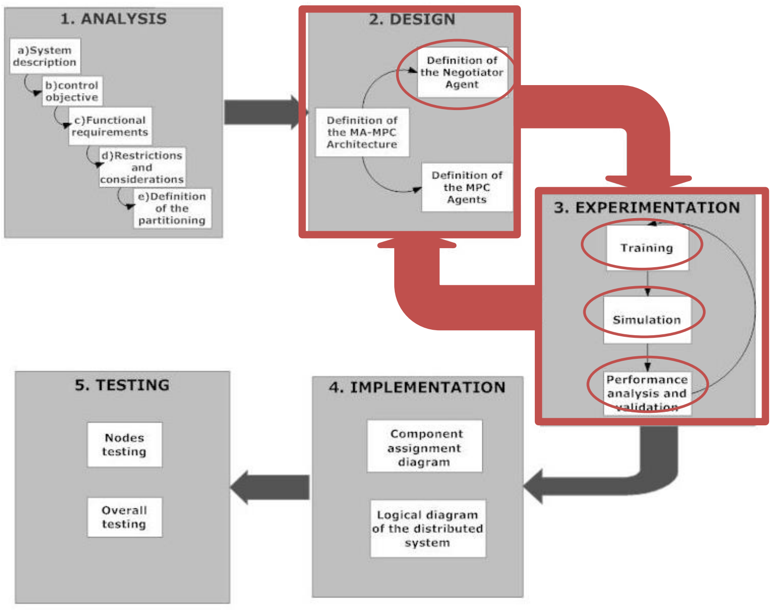

1. ANALYSIS
a)System description
b)control objective
c)Functional requirements
d)Restrictions and considerations
e)Definition of the partitioning
2. DESIGN
Definition of the Negotiator Agent
Definition of the MA-MPC Architecture
Definition of the MPC Agents
3. EXPERIMENTATION
Training
Simulation
Performance analysis and validation
4. IMPLEMENTATION
Component assignment diagram
Logical diagram of the distributed system
5. TESTING
Nodes testing
Overall testing

Figure 7.1 Process flow made in the iteration between experimentation and design phases

7.1 Design

As it was mentioned before, the proposed extended methodology modifies the *definition of the NA* process, the other processes of the design phase remains unchanged. So, in the behaviors module of the NA, the behavior PBEB is added while the greedy behavior is still used. All the other definitions of the NA remain the same (see Section 7.2). The new structure of the NA is shown in Figure 7.2.

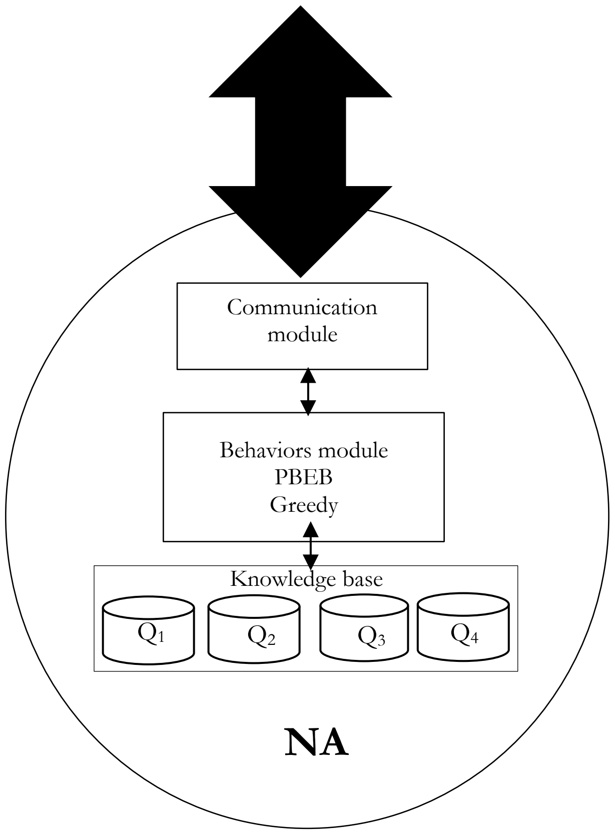


Figure 7.2 New internal structure of the NA

The PBEB algorithm used is explained in Section 4.4.2. In this algorithm, s_1 Eq. (6.17) and s_2 Eq. (6.18) represents the *state* of M_1 and M_2 (the two MPC agents that share that particular negotiation variable). These variables are calculated taking in to account Eqs. (6.19), (6.17) and (6.18), the value is discretized using Eq. (6.20). and the parameters presented in Table 6.13.

Once calculated, they have to be discretized using Eq.(6.20). The discretized value of s_1 and s_2 are used in the algorithm. V_1 and V_2 are the internal representation if the shared variable in M_1 and M_2 for each time instant *k*. *a* is the random action selected. The calculus of the *reward* (*r*), in line 9 obeys Eq.(6.21) and Eq. (6.22).

The objective of this algorithm is to learn by exploration, trying random actions but using just the meaningful experience and discarding the actions that lead to unfeasible states.

7.2. Experimentation.

Once the design phase is completed, it is time to define the experiments. The first part of the experimentation phase, according to MA-MPC methodology is *training*.

7.2.1 Training.

The training implements the *PlanningByExploration* behavior. Many experiments were made in this phase. First experiments were made using just explorative learning. Then, the *PlanningByExploration* behavior (PBEB) was implemented varying the number of the iterations in the training. Then, PBEB with selective penalization of reward was implemented. All training was made for the complete control horizon (24 hrs.) for each shared variable. Random initial conditions were set for each complete horizon. During training, the *Q-table* for each shared variable was filled with the *Q-values* calculated for all states visited.

Eq. (7.1) is the function that updates each Q-table. α Rates past experience and is set on 0.5

$$Q(s'_{a1}, a', s'_{a2}) \leftarrow r + \propto Q(s_{a1}, a, s_{a2}) \tag{7.1}$$

The learning behavior *PlanningByExploration*, selects the actions that leads to a feasible solution of the related MPC agents. In this experiment, PBEB with selective penalization of reward was implemented applying a penalization in the opposite case, this means that if there is no feasible solution for MA-MPC agents, a negative reward was assigned (-1000). This negative reward ensures that the *Q-value* of state-action-state that leads to critical states stays low and accelerates drastically the training process allowing the MA-MPC system to improve the centralized MPC solution from the iteration 20 (see Table 7.1).

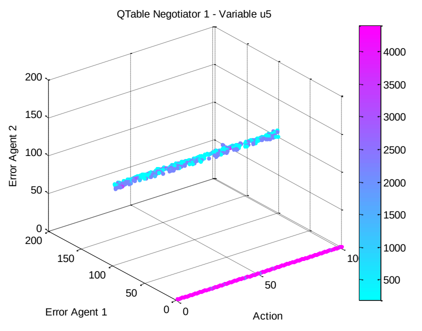
Algorithm 5 PlanningByExploration behavior algorithm with selective penalization

```
1.  Define $\rho$ that satisfies (4.3), n, $s_{a1} \leftarrow$ random, $s_{a2} \leftarrow$ random, controlHorizon,
    k=1
2.  loop while iterations $\leq$ n
3.    loop while k $\leq$ controlHorizon
4.      a $\leftarrow$ random (a) $\in A$ $Q(s_1', a, s_2')$
5.      $V_{a1}$ (k) $\leftarrow$ a
6.      $V_{a2}$ (k) $\leftarrow$ a
7.      $s_{a1} \leftarrow$ send $V_{a1}$ (k) to MPCagent1 , MPCagent1 set the action $V_{a1}$ (k) and
        calculates its internal variables, apply all the controls (actions) obtained
        (and given) for step k to its LTI model of its partition and calculates $s_{a1}$
        using (4.1).
8.      $s_{a2} \leftarrow$ send $V_{a2}$ (k) to MPCagent2, MPCagent2 set the action $V_{a2}$ (k) and
        calculates its internal variables, apply all the controls (actions) obtained
        (and given) for step k to its LTI model of its partition and calculates $s_{a2}$
        using (4.2).
9.      if MPCagent1 and MPCagent2 have a feasible solution
10.       r $\leftarrow \rho - s_{a1} - s_{a2}$
11.       Q $(s_{a1}', a', s_{a2}') \leftarrow r + \alpha\, Q(s_{a1}, a, s_{a2})$
12.       $s_{a1}' \leftarrow s_{a1}$
13.       $s_{a2}' \leftarrow s_{a2}$
14.     else
15.       r $\leftarrow$-1000
16.       Q $(s_{a1}', a', s_{a2}') \leftarrow r + \alpha\, Q(s_{a1}, a, s_{a2})$
17.       $s_{a1}' \leftarrow$ random
18.       $s_{a2}' \leftarrow$ random
19.     end if
20.     k=k+1
21.   end loop
22.   iterations=iterations+1
23. end loop
```

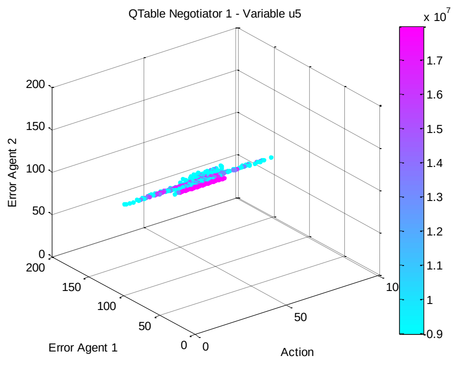
The experimentation made on this example shows that using just using exploration, the system can not recover from states related to unfeasible solutions. In addition, these states have high frequency of visits because it is more likely that the random action selected were not the good one. This affects negatively the learning process because the accumulation of many small rewards becomes in larges *Q-values*.

In order to solve that issue, selective feedback was applied (as shown in the algorithm described in Section 4.4.2). This reduces drastically the iterations needed using just exploration and the *Q-values* result more reliable. Moreover, the use of a negative reward in the selected actions that lead to unfeasible states also provide a huge improvement. After assigning the negative reward, s'_1 and s'_2 are set to random in order to continue the learning process effectively.

With these conditions a training of 100 iterations was carried out. Figure 7.3 (a) shows a color representation of the *Q-values* calculated in the learning process. The *Q-table* allows to present the error of M_1 and M_2 (or the discretize state of each agent) with the action taken. In order to use only positive errors, the errors are scaled from 0 to 200. Negative errors are scaled from 0 to 99, 100 is 0 while values from 101 to 200 correspond to positive errors. Actions are ranging from 0 to 100. The figure compares the *Q-tables* obtained using PBEB (a) and (PBIB) (b). From Figure 7.3 (a), it can be noticed that the cloud of data spreads all over the action axis, meaning that all actions were explored. Figure 7.3 (b) shows the Q-table of shared variable u_5 with a training of 300 iterations using PBIB. In this *Q-table*, the cloud of *Q-values* is more compact because its training only tried the actions dictated by the teacher (in this case, centralized MPC).



(a)



(b)

Figure 7.3 Comparison of the resulting *Q-Tables* of the variable u_5 using PBEB (a) and PBIB(b)

7.2.2 Simulation.

In other to know if the training phase is finished it is necessary to evaluate the elements of the *Q-table* by means of testing and exploiting. The simulation process implements the *greedy behavior* (section 4.4.3). This algorithm observes the state of the MPC agents, s_1 and s_2 (in its discretized form), and maps it to the action that maximizes the accumulated *Q-value*.

Figure 7.5 shows the results of a simulation in the exploitation phase presenting the states (the volume in tanks) of the four tanks with multiple dependences problem (see Figure 7.4).

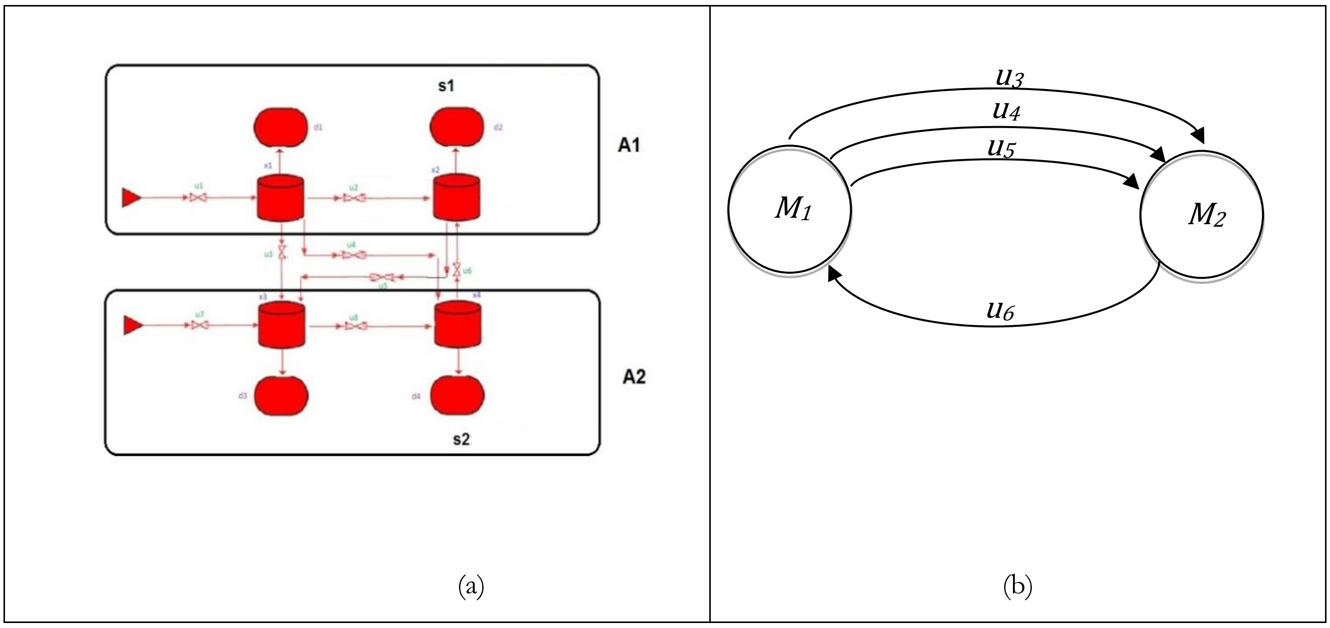
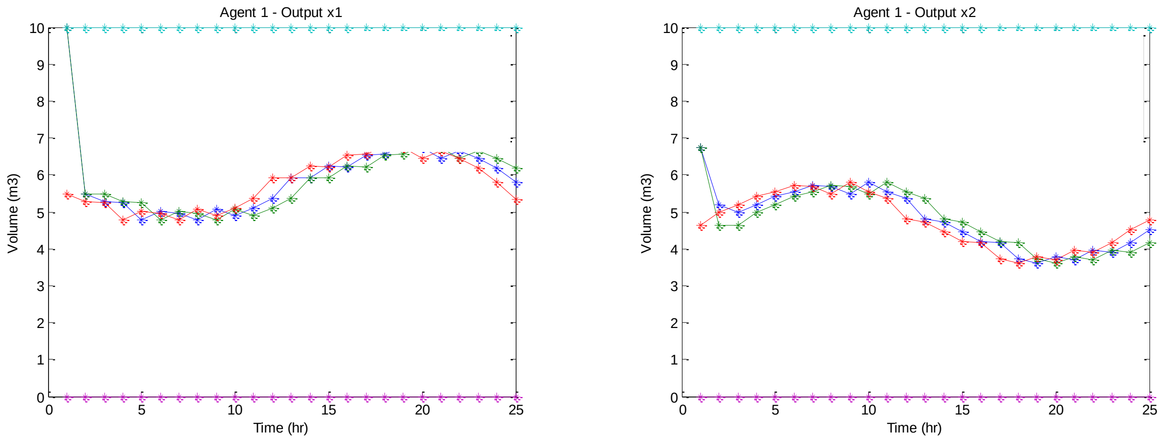
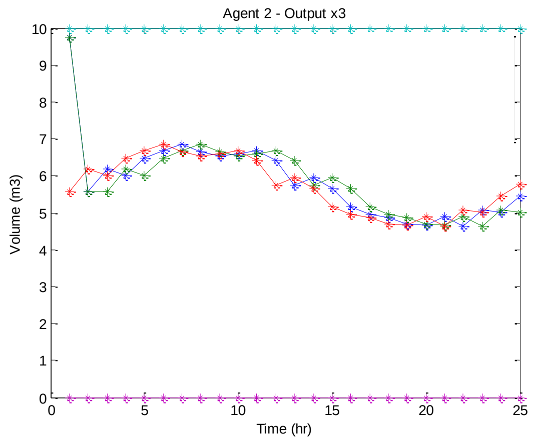


Figure 7.4 (a) System partitions. (b) Shared variables between M_1 and M_2

The simulation results presented in Figure 7.5 allow to compare the MA-MPC using PBEB (blue line) and the centralized solution (green line) with the same random initial conditions and references (red line), obtained after a training of 100 iterations using PBEB with selective penalization of reward explained above. Notice that the reference is variable in time. The parameters of MPC agents and the centralized MPC system are the same. From Figure 7.5, it can be noticed that both approach force the system to track the reference.





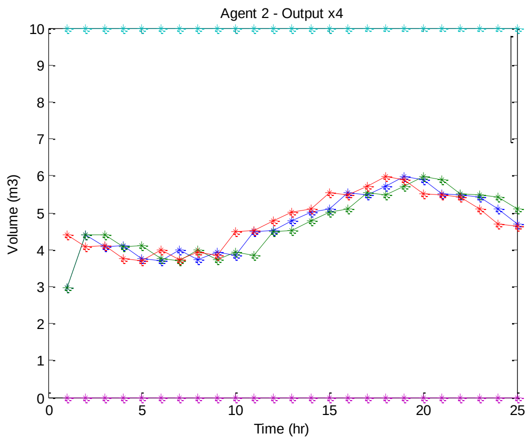
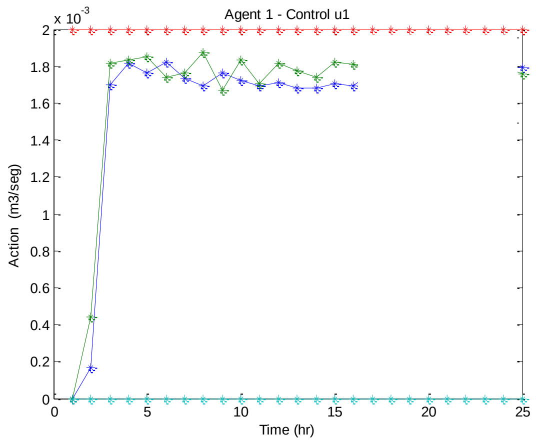
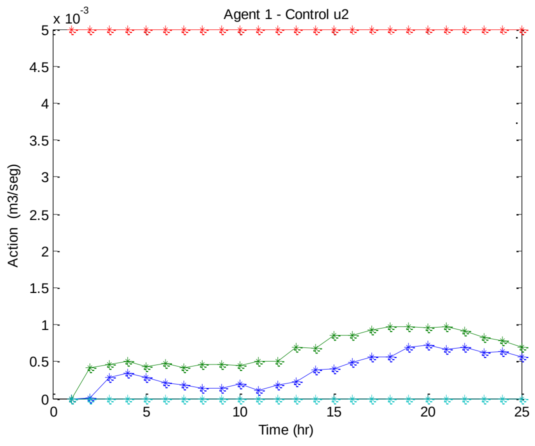


Figure 7.5 Results of the MPC agents (blue) compared with the centralized MPC (green) solution. The red line is the reference, purple x min, cyan x max.

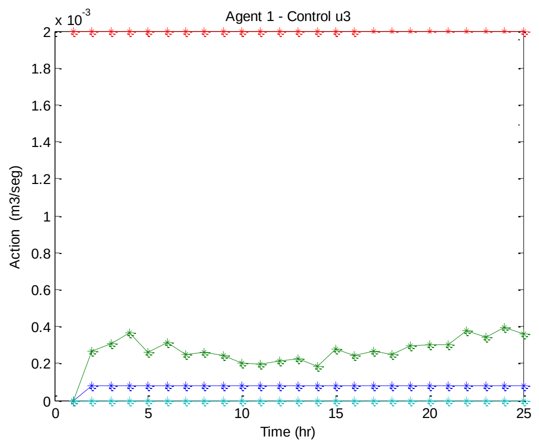
Figure 7.6 shows the resulting actions applied to the four tanks with multiple dependences problem. Sections c, d, e and f of Figure 7.6 show *shared variables* while section a, b, g and h of this figure shows *internal variables* of the MPC agents M_1 and M_2, respectively. It can be noticed, the actions calculated by the NA (the shared variables) vary less over time, without sacrificing performance.



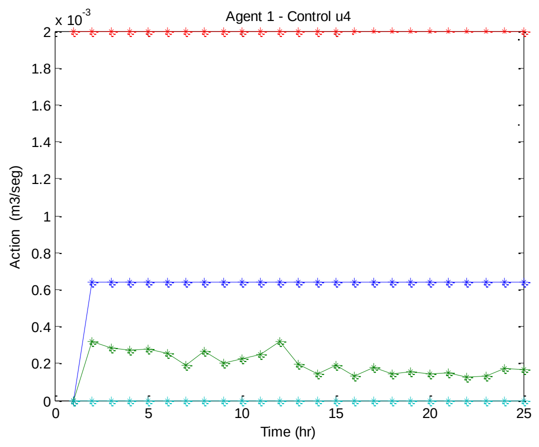
(a)



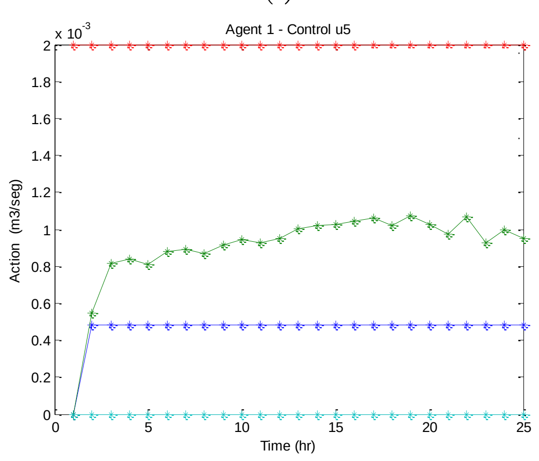
(b)



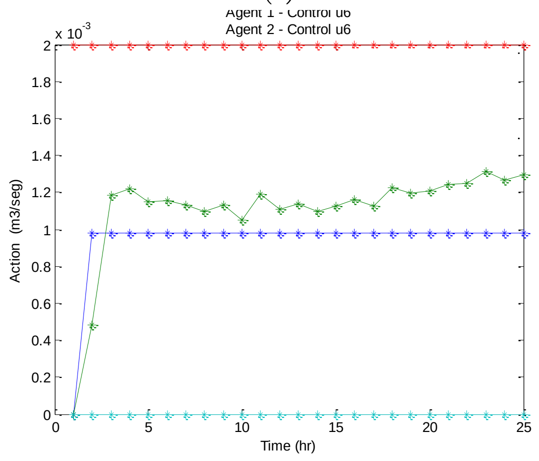
(c)



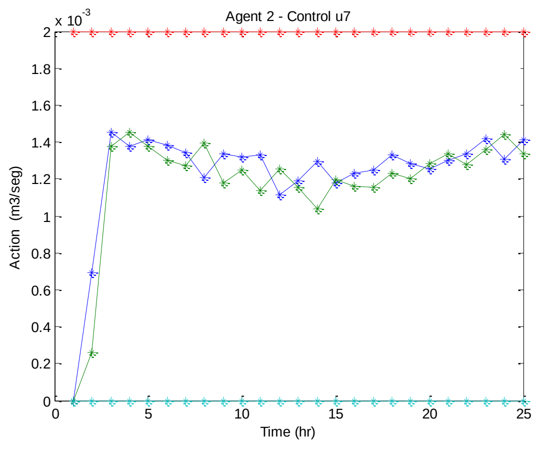
(d)



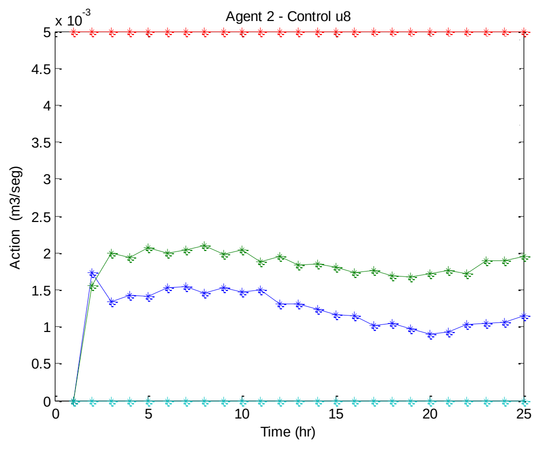
(e)



(f)



(g)



(h)

Figure 7.6 Actions (u´s) applied by the MA-MPC using PBEB with selective reward (blue) and the centralized (green) solution in a simulation of the four tanks with multiple dependences problem. Red line u_{max}, cyan u_{min}..

7.2.3 Performance analysis and validation.

Many simulations were made to assess the performance of the extended proposed approach. Table 7.1 shows the comparison of the average absolute error (with respect to the reference) of 30 simulations in the training process of the best Q-tables found, the ones obtained using PBEB with selective penalization of reward. Columns show the results of a training of 20, 50 and 100 iterations with random reference and initial conditions. From this table, it can be noticed that the MA-MPC solution improves the centralized solution since the first 20 iterations of the training and keeps improving slightly as iterations increase.

J_e	**20 it**	**50 it**	**100 it**
Error a1	17,2429	15,8219	16,3230
Error a2	18,3069	17,5714	16,3808
Total error MA-MPC	35,5499	33,3932	32,7038
Total error Centralized MPC	45,3116	43,7657	42,8805

Table 7.1 Comparioson of the average absolute error between MPC-agents, MA-MPC system and centralized MPC solution with trainings of 20, 50 and 100 iterations.

Table 7.2 shows the comparison of the average $J_{\Delta u}$ between MA-MPC system and the centralized MPC solution with trainings of 20, 50 and 100 iterations. The system stops improving after a training of 100 iterations.

$J_{\Delta u}$	**20 it**	**50 it**	**100 it**
Centralized MPC	0,0001	0,0001	0,0001
MA-MPC	0,0109	0,0110	0,0111

Table 7.2 Comparison of average $J_{\Delta u}$ between MA-MPC system and centralized MPC solution with trainings of 20, 50 and 100 iterations.

It was observed that the actions calculated by the NA (the shared variables) vary less over time without sacrificing performance. But, the accumulative control effort is grater compared with the centralized MPC.

Other experiments were made increasing or decreasing the negative reward but for this problem the best negative reward was -1000.

7.3 Conclusions

Explorative training is usually exhaustive. This complexity is reduced applying selective feedback (using PBEB) but the combination of the use of negative reward for the selected feedbacks not just improves the results compared to the centralized MPC but also the *PlanningByInstruction Behavior* (PBIB) and decrease drastically the iterations needed in the training phase. Table 7.3 shows the average absolute error with respect to the reference of 30 simulations of the PBEB with selective penalization of reward and the PBIB. Random initial conditions and random references were use. The random cases calculated for PBEB with selective penalization of reward were different than the ones calculated for PBIB. The training of the PBEB with selective penalization of reward, involves 100 iterations while in the case of the PBIB uses 300 iterations.

	PBEB selective reward	**PBIB**
Error M_1	16,3230	24,04
Error M_2	16,3808	24,23
MA-MPC	32,7038	48,27
Centralized MPC	42,8805	44,08

Table 7.3 contrast of errors between the modification of PBEB and PBIB

Table 7.4 shows the average $J_{\Delta u}$ obtained using the MA-MPC and the centralized MPC solution in the same experimentation conditions that those used to obtain the results presented the Table 7.3.

	PBEB	**PBIB**
Centralized MPC	0,0001	6,333e-05
MA-MPC	0,0107	0,0153533

Table 7.4 contrast of the $J_{\Delta u}$ between the modification of PBEB and PBIB

Thus, the experimentation results obtained in this example show that the modification of the PBEB is a more efficient learning technique than PBIB due to the reduction of the error and the iterations needed in training.

Chapter 8. Application to the Barcelona drinking water network case study.

In this chapter, the PBEB with selective penalization of reward was applied to the Barcelona drinking water network (DWN) case study. The Barcelona DWN, managed by Aguas de Barcelona, S.A. (AGBAR), not only supplies drinking water to Barcelona city but also to the metropolitan area. The sources of water are the Ter and Llobregat rivers, which are regulated at their head by some dams with an overall capacity of 600 cubic hectometers. Currently, there are four drinking water treatment plants (WTP): the Abrera and Sant Joan Despí plants, which extract water from the Llobregat river, the Cardedeu plant, which extracts water from Ter river, and the Besòs plant, which treats the underground flows from the aquifer of the Besòs river. There are also several underground sources (wells) that can provide water through pumping stations. Those different water sources currently provide a flow of around 7 m_3/s. The water flow from each source is limited and with different water prices depending on water treatments and legal extraction canons.

The Barcelona DWN is structurally organized in two layers. The upper layer, named *transport network*, links the water treatment plants with the reservoirs distributed all over the city. The lower layer, named *distribution network* is sectored in subnetworks. Each subnetwork links a reservoir with each consumer. This application case study is focused on the transport network. Thus, each subnetwork of the distribution network is modeled as a demand sector. The demand of each sector is characterized by a demand pattern, which can be predicted using a time-series model (Quevedo, 2010). The control system of the transport network is also organized in two layers. The upper layer is in charge of the global control of the network, establishing the set-points of the regulatory controllers at the lower layer. Regulatory controllers are of PID type, while the supervisory layer controller is of MPC type. Regulatory controllers hide the network non-linear behavior to the supervisory controller. This fact allows the MPC supervisory controller to use a control-oriented linear model.

In this chapter, the Barcelona DWN aggregate network presented in Figure 8.1 has been used. From this figure, it can be seen that the network is comprised of 17 tanks (state variables), 61 actuators (26 pumping stations and 35 valves), 11 nodes and 25 main sectors of water demand (model disturbances). The model has been derived using the control oriented modeling methodology proposed in (Ocampo-Martinez, 2011). The obtained model has been compared against real behavior assessing its validity. The detailed information about physical parameters and other system values are reported in (Fambrini, 2009).

Using the partitioning obtained in (Ocampo-Martinez, 2011), the aggregate model of the Barcelona DWN is decomposed in three subsystems, as depicted in Figure 8.1 in different colors.

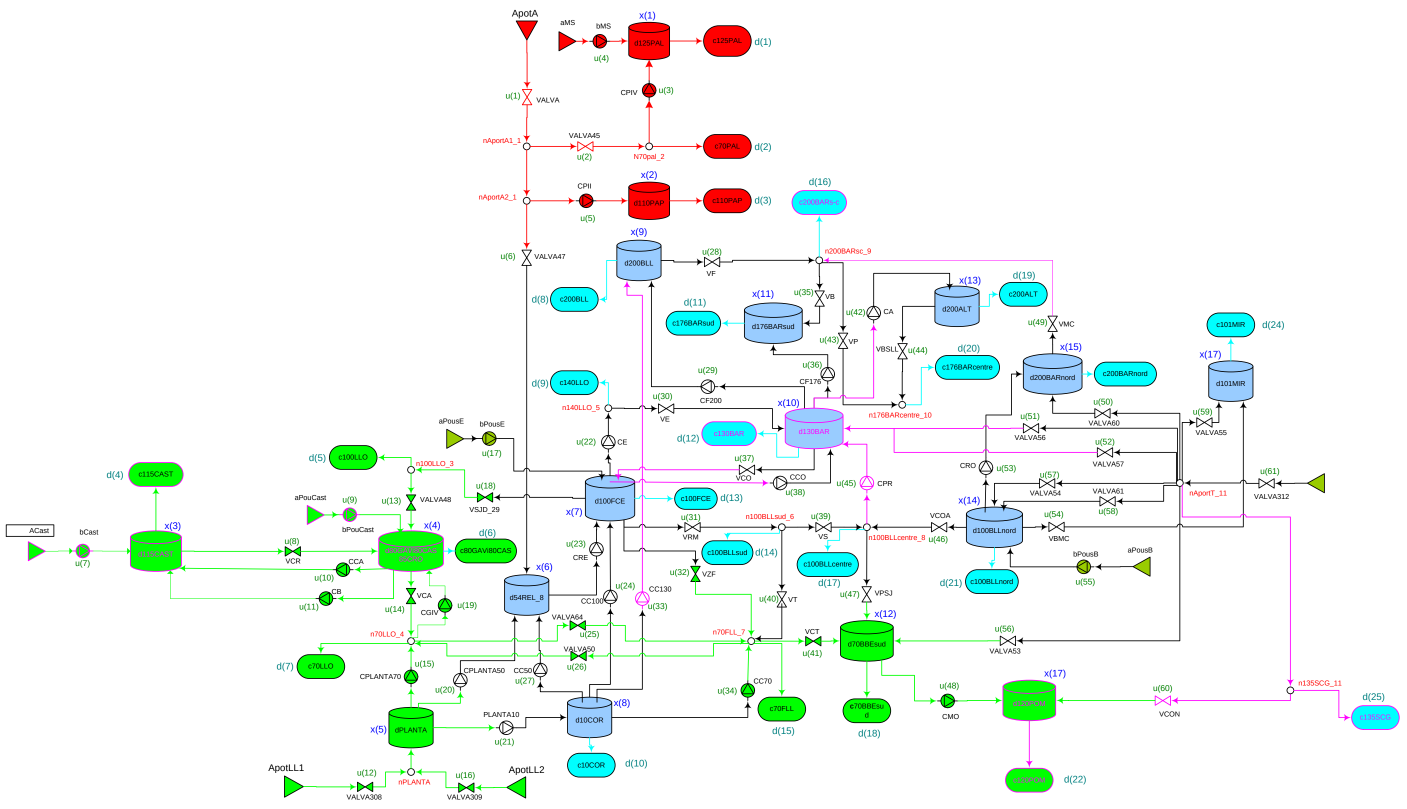


Figure 8.1 System diagram of the Barcelona DWN aggregate network

Table 8.1 collects the resultant dimensions for each subsystem and the corresponding comparison with the dimensions of the vectors of variables for the entire aggregate network.

Elements	Subsyst 1	Subsyst 2	Subsyst 3	Whole Model
Tanks	2	5	10	17
Actuators	5	22	34	61
Demands	4	9	22	25
Nodes	2	3	6	11

Table 8.1 Dimension comparison between the subsystems and the whole network

The proposed framework was applied to the Barcelona DWN using the partitioning of the aggregated network (Figure 8.1). The functional requirements of this system are presented in Table 8.2. The control objective in this case, is reflected in FR3 and FR4, this means that the priority of the control is to maintain the system inside the security levels, a desirable reference is use but the priority are FR3 and FR4, the last one, refers to a smooth control, that means that control actions should increase /decrease in small quantities.

Req No.	**Name of the requirement.**	**Description.**
FR1	Type of partitioning.	As defined in Figure 8.1.
FR2	Distributed control.	One controller for each partition.
FR3	Security levels.	The behavior of tank levels should maintain in the defined limits.
FR4	Smooth control.	Control actions should increase / decrease in small quantities.
FR5	Avoid conflicts and collisions.	Avoid conflicts and collisions between subsystems.
FR6	Satisfy demands.	All demands have the same priority.
FR7	Global optimization	Seek the global optimality of the system.

Table 8.2 Functional requirements of the Barcelona DWN.

According to FR2, an MPC agent (named M_1, M_2 and M_3 respectively) was assigned to each partition (subsystem). The Figure 8.2 shows the MPC agents and the relations between them in the relation diagram of the system.

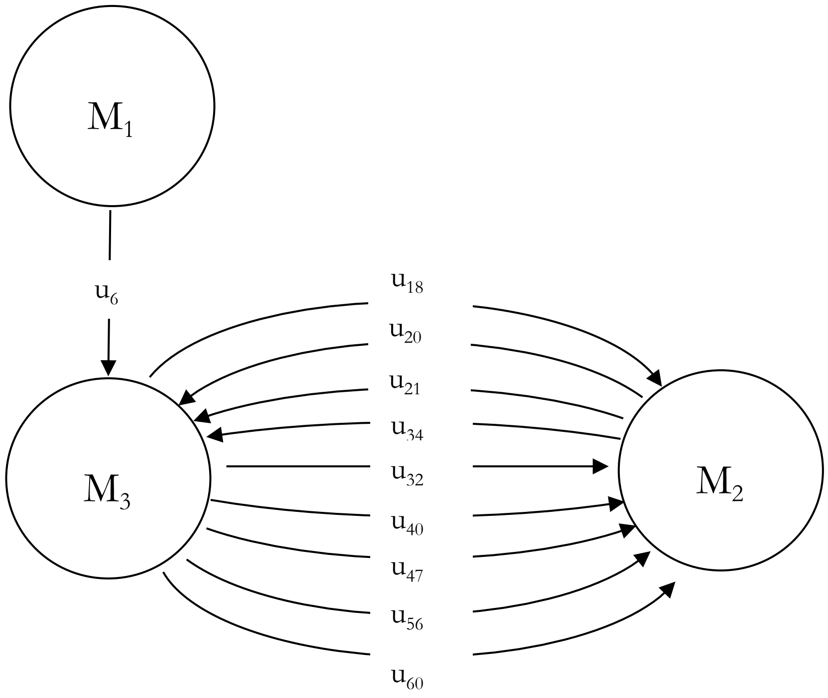


Figure 8.2 Relation diagram of the Barcelona aggregate DWN

A NA was placed between the MPC agents with shared variables between them. Three MPC agents and two negotiator agents were required. Figure 8.3 shows the resulting general structure of the DWN system diagram. Eq. (8.1) and Table 8.3 defines de MA-MPC architecture for this system.

$$\gamma = \{M, N, P, W, V, U, b\} \tag{8.1}$$

Variable	Definition
M	$\{M_1, M_2, M_3\}$,
N	$\{N_1, N_2\}$,
P	$\{X, U, E, D\}$,
W	$\{W_1, W_2, W_3\}$,
V	$\{V_1, V_2, V_3\}$,
U	$\{U_1, U_2, U_3\}$

Table 8.3 Table γ of the Barcelona DWN .

where:

$$X = \{X_1, X_2, X_3\},\ U = \{U_1, U_2, U_3, V\},\ E = \{E_1, E_2, E_3\},\ D = \{D_1, D_2, D_3\} \text{ and } V = \{V_1, V_2, V_3\} \qquad (8.2)$$

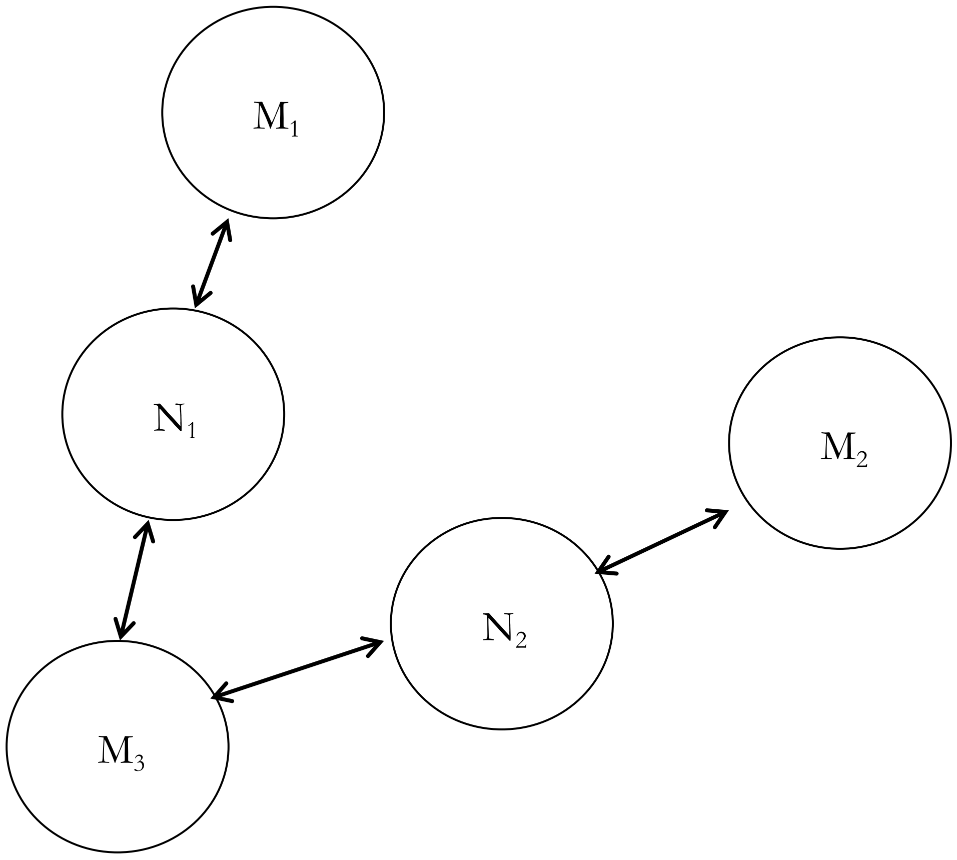


Figure 8.3 General structure of the Barcelona BWN MA-MPC implementation.

In this way, N_1 is in charge of shared variable u_6 and N_2 is in charge of u_{18}, u_{20}, u_{21}, u_{34}, u_{32}, u_{40}, u_{47}, u_{56} and u_{60}. Figure 8.4 and Figure 8.5 show the internal structure N_1 and N_2, respectively.

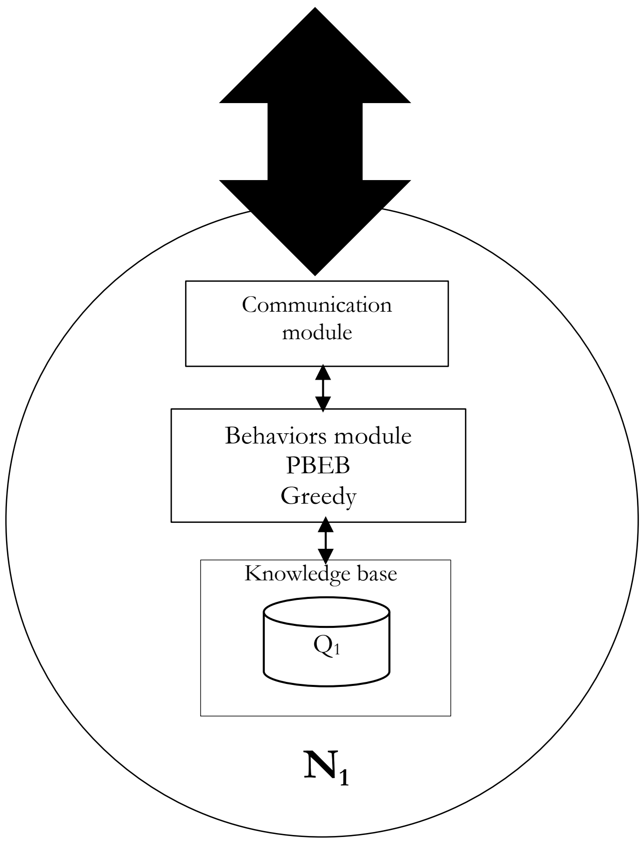


Figure 8.4 Internal structure of N_1 of the Barcelona DWN

Q$_1$ (see Figure 8.4) represents the *Q-table* for shared variable u_6 and it has the form $Q(s_1, a, s_2)$. The knowledge base of N_2 has *Q-tables* with the same structure as Q_1 to Q_9 for shared variables u_{18}, u_{20}, u_{21}, u_{34}, u_{32}, u_{40}, u_{47}, u_{56} and u_{60}, respectively (Figure 8.5). In the behaviors module PBEB and *greedy behavior* are implemented.

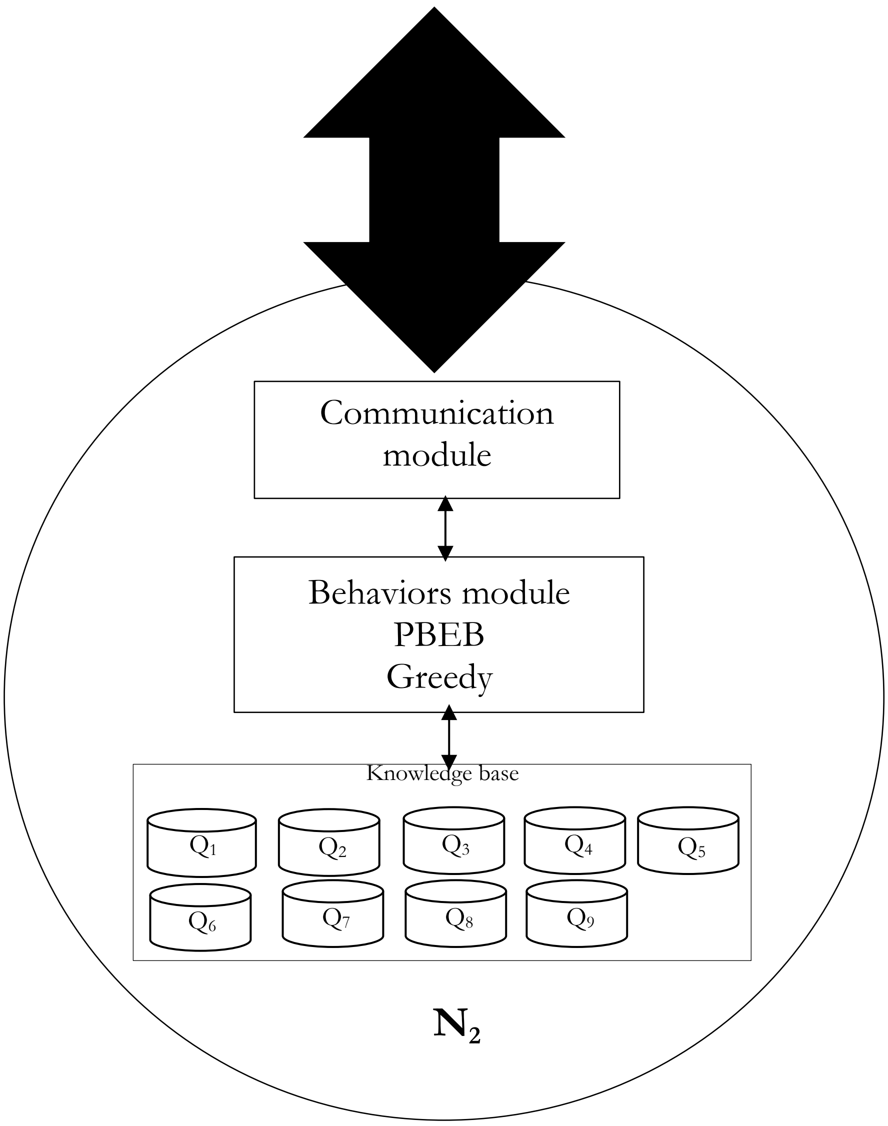


Figure 8.5 Internal structure of N2 of the BWN system

The calculus of states, reward and the prediction orizon H_p are the same for all MPC agents and are defined next,

$$s_1 = \sum_{i=0}^{Hp} J(i) = \sum_{i=0}^{Hp} J_x(i) + \sum_{i=0}^{Hp} J_{\Delta u}(i)$$

(8.3)

$$s_2 = \sum_{i=0}^{Hp} J(i) = \sum_{i=0}^{Hp} J_x(i) + \sum_{i=0}^{Hp} J_{\Delta u}(i)$$

(8.4)

$$J_x = \vec{e}^T w_x \vec{e} \quad \text{and} \quad J_{\Delta u} = \overrightarrow{\Delta u}^T w_{\Delta u} \overrightarrow{\Delta u}$$

(8.5)

the weights have selected as follows

$$w_{\Delta u} = w_x = 1$$

(8.6)

and the prediction horizon is chosen as

$$H_p = 24$$

(8.7)

Finally, s_1 and s_2 have been discretized as in (6.21).

Reward calculation

The *reward* (*r*) was calculated as follows

$$\sigma = \rho - s_1 - s_2$$

(8.8)

where σ represents the *reward r* and ρ is a constant that complies:

$$s_1 + s_2 < \rho$$

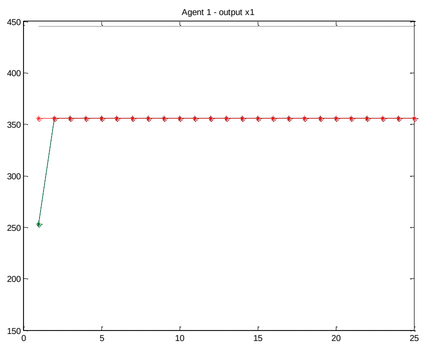
(8.9)

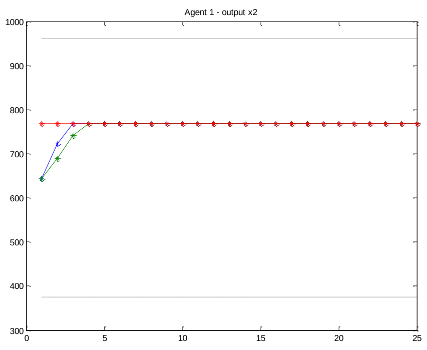
The PBEB was implemented using a penalization instead of just discarding actions that leads to unfeasible states. The algorithm below was used simultaneously for each *Q-table*. So M_1 and M_2 represent the two MPC agents with shared variables between them (M_1 and M_3; M_3 and M_2).

Algorithm 5 PlanningByExploration behavior algorithm with selective penalization

```
1.  Define $\rho$ that satisfies (8.9), n, $s_{a1} \leftarrow$ random, $s_{a2} \leftarrow$ random, controlHorizon,
    k=1
2.  loop while iterations $\leq$ n
3.    loop while k $\leq$ controlHorizon
4.      a $\leftarrow$ random (a) $\in A$ $Q(s_1', a, s_2')$
5.      $V_{a1}$ (k) $\leftarrow$ a
6.      $V_{a2}$ (k) $\leftarrow$ a
7.      $s_{a1} \leftarrow$ send $V_{a1}$ (k) to MPCagent1 , MPCagent1 set the action $V_{a1}$ (k) and
        calculates its internal variables, apply all the controls (actions) obtained
        (and given) for step k to its LTI model of its partition and calculates $s_{a1}$
        using (8.3).
8.      $s_{a2} \leftarrow$ send $V_{a2}$ (k) to MPCagent2, MPCagent2 set the action $V_{a2}$ (k) and
        calculates its internal variables, apply all the controls (actions) obtained
        (and given) for step k to its LTI model of its partition and calculates $s_{a2}$
        using (8.4).
9.      if MPCagent1 and MPCagent2 have a feasible solution
10.       r $\leftarrow \rho - s_{a1} - s_{a2}$
11.       Q ($s_{a1}'$, a', $s_{a2}'$ )$\leftarrow$ r $+\alpha$ Q($s_{a1}$, a, $s_{a2}$)
12.       $s_{a1}' \leftarrow s_{a1}$
13.       $s_{a2}' \leftarrow s_{a2}$
14.     else
15.       r $\leftarrow$-1000
16.       Q ($s_{a1}'$, a', $s_{a2}'$ )$\leftarrow$ r $+\alpha$ Q($s_{a1}$, a, $s_{a2}$)
17.       $s_{a1}' \leftarrow$ random
18.       $s_{a2}' \leftarrow$ random
19.     end if
20.     k=k+1
21.   end loop
22.   iterations=iterations+1
23. end loop
```

The objective of this algorithm is to learn by exploration, trying random actions but using just the meaningful experience and penalizing the actions that lead to unfeasible states. A training of just 50 iterations using a negative reward of -1000 was applied in order to obtain the results below. Simulations use same random initial state and reference. The results obtained by means of the proposed framework are compared with those obtained when a centralized MPC strategy is used. The model parameters and measured disturbances (demands) have been supplied by AGBAR. Demands data correspond to consume of drinking water of the city of Barcelona during the year 2007.

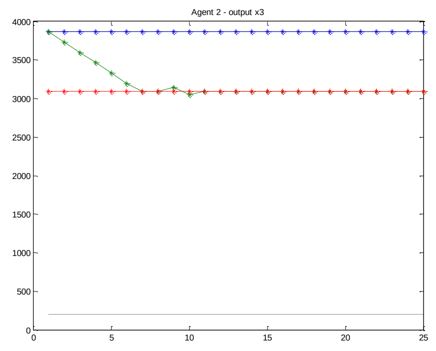




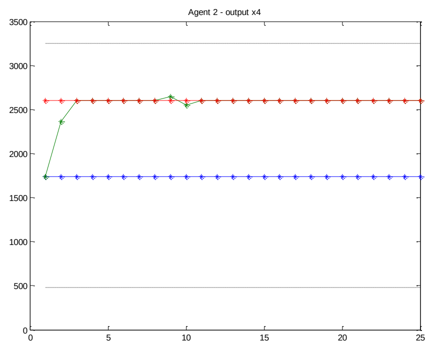
(a) (b)

Figure 8.6 Tank volume evolutions of M_1 (red sub-sytem). Blue line represents MA-MPC solution and green line represents the centralized MPC.

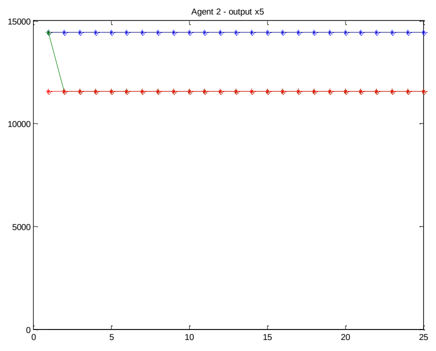
Figure 8.6 shows the volume evolution of tanks of M_1 (red sub-system). Blue line represents MA-MPC solution and green line centralized MPC. Black dotted lines are min and max volume values. Red line is the reference. In Figure 8.6 (a) Volume evolution of tank 1 of the MPC agent and centralized MPC overlap.



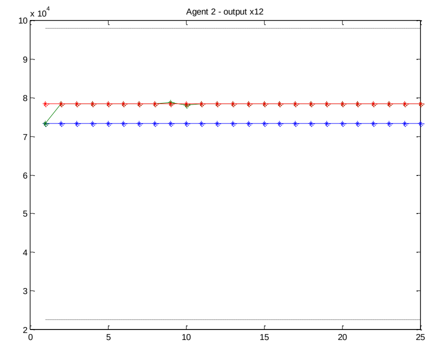
(a)



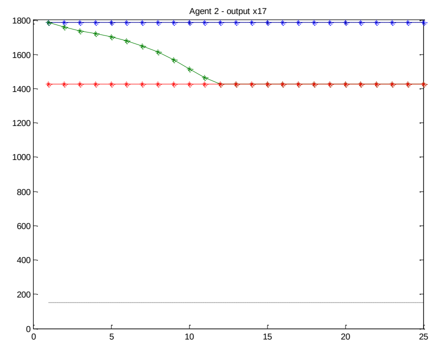
(b)



(c)



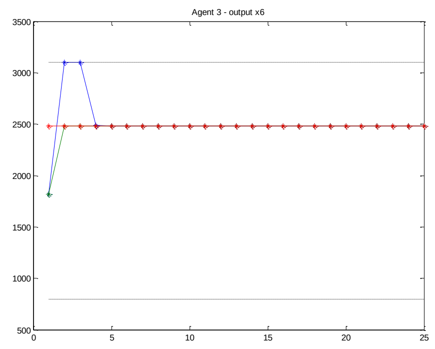
(d)

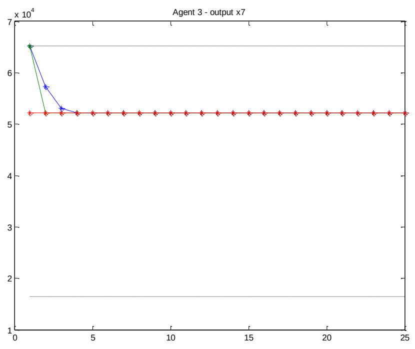


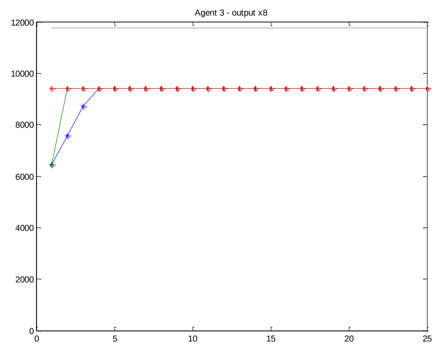
(e)

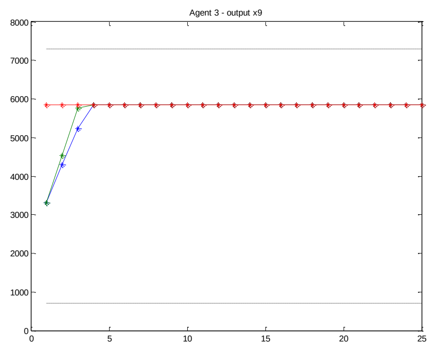
Figure 8.7 Tank volume evolutions of M_2 (green sub-system). Blue line represents MA-MPC solution and green line centralized MPC.

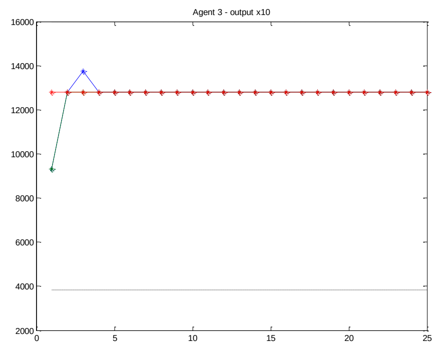
Tank volume evolutions presented in Figure 8.7 (a), (b) and (c) are the ones that are farthest from the reference they are kept in the boundaries of security. On the other hand, tank volume evolutions presented in Figure 8.7 (d) and (e) corresponds to the ones that better approach to the reference.

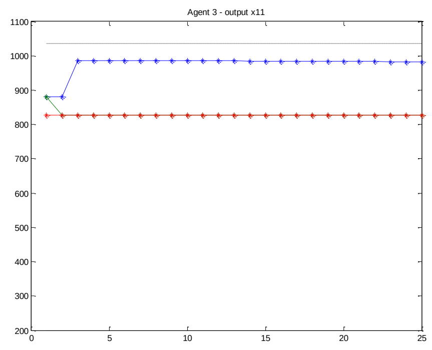

Agent 3 - output x6

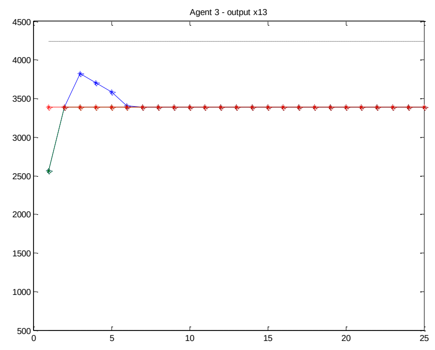

Agent 3 - output x7
x 10^4

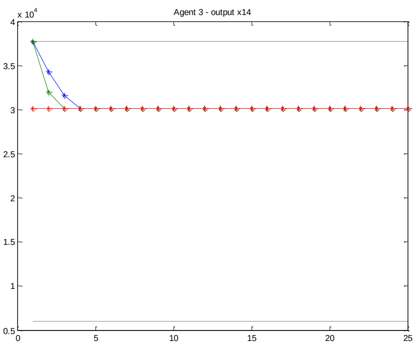

Agent 3 - output x8


Agent 3 - output x9


Agent 3 - output x10


Agent 3 - output x11


Agent 3 - output x13


Agent 3 - output x14
x 10^4

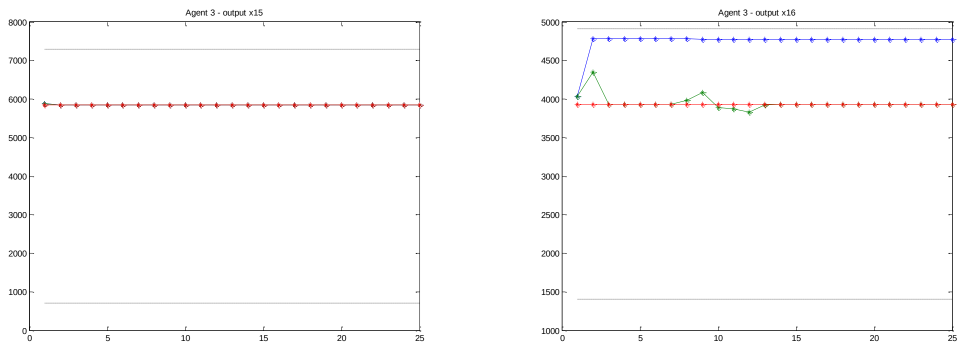


Figure 8.8 Tank volume evolutions of M_3 (blue sub-system). Blue line represents MA-MPC solution and green line centralized MPC.

	Abs. error MA-MPC	**Abs. error Centralized MPC**
M_1	274,50	332,96
M_2	249.867,02	15.000,60
M_3	74.702,00	35.244,99
Total	324.843,53	50.578,55

Table 8.4 Average absolute error of MA-MPC and centralized MPC solutions.

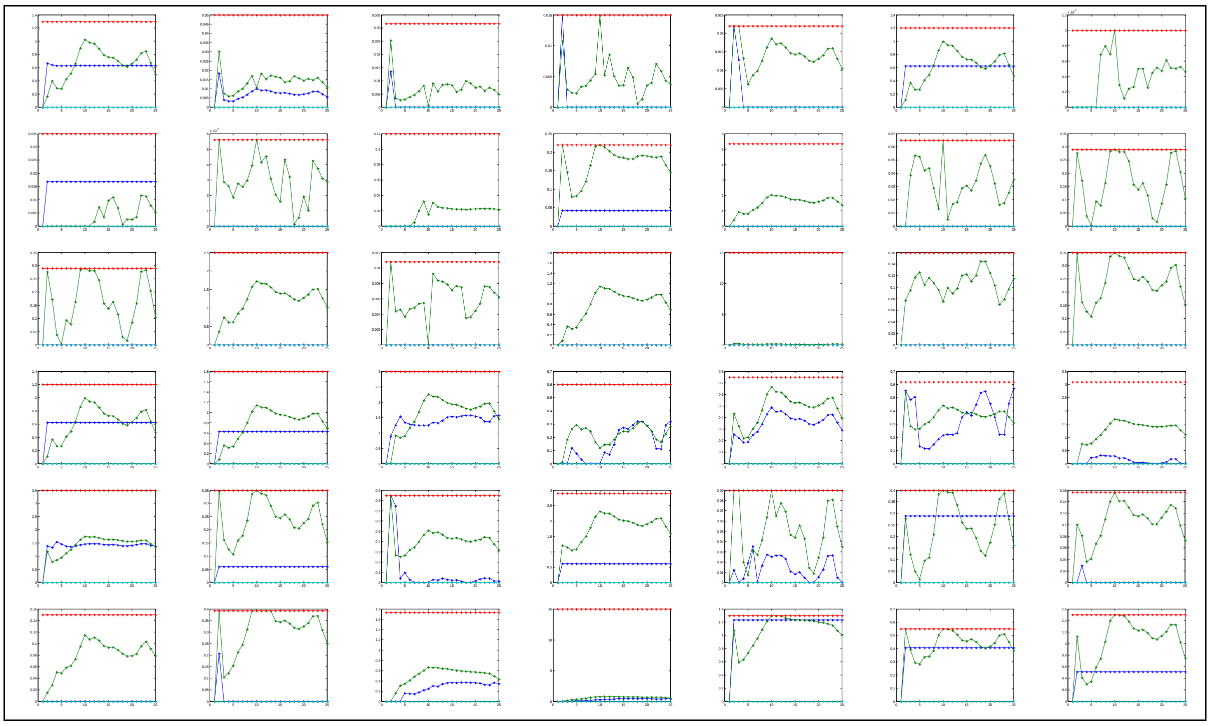


Figure 8.9 Some of the control variable evolution corresponding to the Barcelona DWN. Blue line represents MA-MPC solution and green line centralized MPC. Cyan and red lines are min and max values of *u*.

$J_{\Delta u}$	M1	M2	M3	Total
Centralized MPC	4,7837	1,7244	132,4717	138,9798
MA-MPC	1,4916	0	69,4476	70,9393

Table 8.2 Average $J_{\Delta u}$ of MA-MPC and centralized MPC solutions.

Conclusions.
The implementation of the PBEB with selective reward of -1000 in the case of the Barcelona DWN leads to a good solution were all the states are kept within the limits with a cost $J_{\Delta u}$ of almost half of the centralized solution. Two of the tree MPC agents (M_1 and M_3) had better performance (according to FR2 and FR3 defined in Table 8.2) that the centralized system, this means that the system accomplish the objectives of keep within the security levels and to maintain a smooth control better that to track the reference. It seems that with a more balanced partitioning of two agents the DWN performance could still improve.

Conclusions and further research.

The results presented in this thesis suggest that the use of the MAMPC architecture for the implementation of distributed MPC can converge and even improve the centralized MPC taking advantage from the MAS properties. Moreover, the Agent Oriented Paradigm provides a suitable framework for development and implementation. Even more, the application of learning techniques allow to develop the Negotiator Agents that allow the coordination of the MPC Agents. Training of the Negotiator Agents can be made directly from a centralized MPC, from human operator driven control (PBIB) or from exploration with selective feedback and negative reward. Data from the centralized MPC is advisable but not essential. The type and quality of the training is a very important issue in order to obtain an efficient optimization. Moreover, the compromise between exploration and exploitation can be implemented on-line to enable the system not just adaptation to the problem but adaptation to changes in time. Communication protocols and coordination methods for MAS have to be studied and tested in a more complex case of study in which many agents interact. The application of the framework using different partitioning of the network has to be also studied.

The first attempts of establishing a meaningful cost function were focused on taking the MPC agent error as the error associated to the subsystem state directly related to the negotiated variable. That way of managing the state of an agent poses several questions, such as: what to do when a negotiated variable is related to more than one state in the subsystem; what to do when a negotiated variable is weakly but directly related to one particular state and strongly but indirectly related to other state(s); what to do if other optimization objectives (economic costs associated to control variables, for instance) are wanted to be considered. The error as defined in this work is general enough to give proper answer to all those questions.

The MAMPC architecture presented in this work is currently being tested on the complete Barcelona DWN. The Barcelona water network is comprised of 200 sectors with approximately 400 control points. At present, the Barcelona information system receives, in real time, data from 200 control points, mainly through flow meters and a few pressure sensors. This network has been used as a LSS case of study to test several LSS control approaches, see (Brdys, 1994) and (Fambrini V., 2009) (Barcelli, 2008) (Ocampo-Martinez C., 2011). As starting point for the application of the MAMPC Architecture, recent work on centralized and decentralized MPC (Ocampo, 2014) (Morcego B, 2014). applied to the

Barcelona network is being used, as well as the partitioning algorithm developed by (Watkins, 1989).

Bibliography

Agostini, A. &. (2005). Feasible control of complex systems using automatic learning. *ICINCO*. Barcelona.

Alpaydin, E. (2014). *Introduction to Machine Learning.* Cambridge: The MIT Press.

Badwell, Q. (2003). A survey of industrial model predictive control technology. *Control Engineering Practice, 11(XX):*, 733–764.

Bakhtiari, A. N. (2007). A cooperative learning aproach to mixed performance controller design: A behavioural viewpoint. *Intelligent Systems Technologies and applications , 2*, 137-160.

Balke, B. H. (2013). Assessing Agent Applications — r&D vs. R&d. En M. G. (Eds.), *Multiagent Systems & Applications* (págs. 1-20). Berlin Heidelberg: Springer-Verlag.

Barcelli, D. (2008). *Optimal decomposition of Barcelona's water distribution network system for applying distributed Model Predictive Control.Master thesis.* Universitat Politècnica de Cataluña-IRI-Universitá degli Study di Siena.

Barto, R. S. (1998). *Reinforcement Learning, an Introduction.* Cambridge, Massachusetts: MIT Press.

Betti, M. F. (2014). Distributed MPC: A Noncooperative Approach Based on Robustness Concepts. En J. M. Negenborn (Ed.), *Distributed Model Predictive Control Made Easy* (Vols. Intelligent Systems, Control and Automation: Science and Engineering 69, págs. 421-436). Dordrecht: Springer Science+Business Media.

Boutilier. (1999). Secuential Optimality and coordination in Multi-Agent Systems. *Sixteenth International Joint Conference on Artificial Intelligence.*

Brdys, M. A. (1994). *Operational control of water systems, Structures, Algorithms and Applications.* Great Britain: Prentice Hall International.

Busonui L., D. S. (2005). *Learning and coordination in dynamic Multiagent Systems.* The Neatherlands: Delf center for Systems and control .

Camacho, B. (2007). *Model Predictive Control (Advanced Textbooks in Control and Signal Processing).* Springer.

Camponogara, E. J. (2002). Distributed Model Predictive Control. *IEEE Control Systems Megazine*, 44-52.

Claus, B. (1998). The dynamics of Reinforcement Learning in cooperative multiagent systems. *Fifteenth National Conference on Artificial Intelligence.*

Dayan, C. W. (1992). Q-learning. *Machine Learning, 8(X),*, 279–292.

Du, Y. X. (2001). Distributed Model Predictive Control for Large Scale Systems. *Proceedings of the IEEE American Control Conference*, (págs. 3142-3143). Arlington, VA, USA.

El Fawal, H. G. (1998). Optimal control of complex irrigation systems via descomposition-coordination and the use of augmented lagrangian. *IEEE Int. conference Systems, man and cybernetics, 4* (págs. 3874-3879). San Diego. CA.: IEEE (Ed.).

Ernst D., G. M. (2006). Model Predictive Control and Reinforcement Learning as a two complementary frameworks. *Proceedings of the 13th IFAC Workshop on Control Applications of Optimisation.*

Ernst, C. W. (2007). Reinforcement Learning Vs Model Predictive Control: A comparison on a power system problem. *IEEE Transactions on Power Systems , 22.*

Fambrini V., C. O.-M. (2009). *Modelling and decentralized Model Predictive Control of drinking water networks. Technical Report IRI-TR-04-09,*. Barcelona: Institut de Robòtica i Informàtica Industrial (CSIC-UPC).

Fambrini, C. O.-M. (2009). *Modelling and decentralized Model Predictive Control of drinking water networks. Technical Report IRI-TR-04-09,*. Barcelona: Institut de Robòtica i Informàtica Industrial (CSIC-UPC).

Farokhi, I. S. (2014). Distributed MPC Via Dual Decomposition and Alternative Direction Method of Multipliers. En J. M. Negenborn (Ed.), *Distributed Model Predictive Control Made Easy* (Vols. Intelligent Systems, Control and Automation: Science and Engineering 69, págs. 115-132). Springer Science+Business Media Dordrecht.

Ferramosca, A. (2014). Cooperative MPC with Guaranteed Exponential Stability. En J. M. Negenborn (Ed.), *Distributed Model Predictive Control Made Easy* (Vols. Intelligent Systems, Control and Automation: Science and Engineering 69, págs. 585-600). Dordrecht: Springer Science+Business Media .

Ferramosca, L. G. (2014). Cooperative Distributed MPC Integrating a Steady State Target Optimizer. En J. M. Negenborn (Ed.), *Distributed Model Predictive Control Made Easy* (Vols. Intelligent Systems, Control and Automation: Science and Engineering 69, págs. 569-584). Dordrecht: Springer Science+Business Media.

Gatti, C. (2015). *Design of experiments for Reinforcement Learning.* Springer.

Gómez, M. R. (1998). Decentralized adaptive control for water distribution. *IEEE International on systems, man and cybernetics* (págs. 1411-1416). San Diego. CA. USA: IEEE (ed.).

Hakansson, H. a. (2010). *Agent and Multi-agent Technology for Internet and Enterprise Systems.* Springer-Verlag.

Hartung, A. H. (2013). *Agent and Multi-Agent Systems in Distributed Systems – Digital Economy and E-Commerce.* Springer.

Hester, T. (2013). *TEXPLORE: Temporal Difference Reinforcement Learning for Robots and Time-Constrained Domains.* Springer.

Jaakkola, M. I. (1994). Q-learning. *Machine Learning, 8(X)*, 1185–1201.

Javalera V., B. M. (2010). Distributed mpc for large scale systems using agentbased reinforcement learning. *IFAC Symposium Large Scale Systems,*. Lille, France.

Javalera V., M. B. (2010). Negotiation and learning in distributed MPC of large scale systems. *ACC 2010.* Baltimore, USA: IEEE Press. Institute of Electrical and Electronics Engineers.

Javalera V., M. S. (2010). A multi-agent MPC architecture for distributed large scale systems. *INSTICC Press. Institute for Systems and Technologies of Information, Control and Communication.*

Jia, B. K. (2002). Min-max feedback model predictive control for distributed control with communications. *Proceedings of the IEEE American Control Conference*, (págs. 4507-4512). Anchorage, AK. USA.

Jurado, D. E. (2014). Cooperative Dynamic MPC for Networked Control Systems. En J. M. Negenborn (Ed.), *Distributed Model Predictive Control Made Easy* (Vols. Intelligent Systems, Control and Automation: Science and Engineering 69, págs. 357-373). Springer Science+Business Media Dordrecht.

Kapentanakis, K. (2002). Reinforcement Learning of coodination in cooperative multi-agents systems. *Eighteenth national conference on Artificial intelligence*, (págs. 326-331). Edmonton, Alberta, Canada.

Lauer, R. (2000). An algorithm for distributed Reinforecement Learning in cooperative Multi-Agent system. *Proceedings of the Seventeenth International Conference on Machine Learning*, (págs. 535-542).

Liu, D. M. (2014). Lyapunov-Based Distributed MPC Schemes: Sequential and Iterative Approaches. En J. M. Negenborn (Ed.), *Distributed Model Predictive Control Made Easy* (Vols. Intelligent Systems, Control and Automation: Science and Engineering 69). Springer Science+Business Media Dordrecht.

Lunze, J. (1992). *Feedback Control of Large Scale Systems.* Prentice Hall.

Maestre, D. M. (2014). Distributed MPC Based on Agent Negotiation. En J. M. Negenborn (Ed.), *Distributed Model Predictive Control Made Easy* (Vols. Intelligent Systems, Control and Automation: Science and Engineering 69, págs. 465-478). Dordrecht: Springer Science+Business Media.

Martinez E., D. P. (2003). Control Inteligente de Procesos usando aprendizaje por interacción. *XXIV Jornadas de Automática.* León, Spain.

Maturana, S. K. (2005). Metodologies and tools for intelligent agents in distributed control. *IEEE Intelligent Systems* , 42-49.

Milan Vidakovic, M. I. (2013). Extensible Java EE-Based Agent Framework – Past, Present, future. Springer-verlag.

Morcego B, j. V. (2014). Distributed MPC Using Reinforcement Learning Based Negotiation: Application to Large Scale Systems. En J. M. (eds.), *Distributed Model Predictive Control made easy* (págs. 517-534). Dordrech: Springer Science+Business Media.

Morge, J. M. (2013). Argumentative Agents for Service-Oriented Computing. En M. G. Jain, *Multiagent Systems and Applications,* (Vol. I, págs. 217-256). Berlin, Heidelberg: Springer-Verlag.

Negenborn, D. S. (2004). *Multi-Agent model predictive control: A survey.* The Netherlands: Delf University of Technology, Delf center for systems and control.

Negenborn, R. R. (2008). Multi-Agent Model Predictive Control with applications to power networks. *Engineering Applications of Artificial Intelligence. 21.*, 353-366.

Novak, K. e. (2013). Simulated Multi-robot Tactical Missions in Urban Warfare. En M. G. Jain, *Multiagent Systems and Applications.* Springer-Verlag.

Ocampo, V. P.-d.-O. (2014). Multi-layer Decentralized MPC of Large-scale Networked Systems. En J. M. Negenborn (Ed.), *Distributed Model Predictive Control Made Easy* (Vols. Intelligent Systems, Control and Automation: Science and Engineering 69, págs. 495-516). Dordrecht: Springer Science+Business Media.

Ocampo-Martinez C., B. D. (2011). Hierarchical and decentralised model predictive control of drinking water networks: Application to the barcelona case study. *IET Control Theory & Applications.*

Pannocchia, S. J. (2014). On the Use of Suboptimal Solvers for Efficient Cooperative Distributed Linear MPC. En J. M. Negenborn (Ed.), *Distributed Model Predictive Control Made Easy* (Vols. Intelligent Systems, Control and Automation: Science and Engineering 69, págs. 553-568). Dordrecht: Springer Science+Business Media.

Pokahr A., L. B. (2013). The Jadex Project: Programming Model. En M. G. Jain, *Multiagent Systems & Applications* (págs. 21-53). Berlin Heidelberg: Springer-Verlag.

Quevedo, V. P. (2010). Validation and reconstruction of flow meter. *Control Engineering Practice*, 640-651.

Rahman, S. M. (2001). *Internet Commerce and Software Agents: Cases, Technologies and Opportunities.* Hershey, P.A.: IGI Global.

Rawlings, S. (2008). Coordinating multple optimization-Based controllers: New opportunities and challenges. *Journal of process control (18)*, 839-845.

Richards, P. A. (2014). Cooperative Tube-based Distributed MPC for Linear Uncertain Systems Coupled Via Constraints. En J. M. Negenborn, *Distributed Model Predictive Control Made Easy* (Vols. Intelligent Systems, Control and Automation: Science and Engineering 69, págs. 57-72). Springer Science+Business Media Dordrecht.

Scattolini, R. (2009). Architectures for distributed and hiearical Model Predictive Control- A Review. *Journal of Process Control* , 723-731.

Stan, G. (1996). Is it an agent or just a program?: A taxonomy of autonomous agents. *Third International workshop on Agent theories, architectures and lenguages* . Springer-Verlag.

Sutton, B. (1998). *Reinforcement Learning, An introduction.* London, England: MIT Press Cambridge Massachussetts.

Tatara, Ç. T. (2007). Control of complex distributed systems with distributed intelligent agents. *Journal of process control* , 415-427.

Tesauro. (2003). Extending Q- Learning to General Adaptive Multi-gent System. *Advances in Neural Information Processing Systems.*

Tsitsiklis, J. (1994). Asynchronous stochastic approximation and q-learning. . *Machine Learning*, 16(X):185–202.

Valencia, J. D. (2014). Bargaining Game Based Distributed MPC. En J. M. Negenborn (Ed.), *Distributed Model Predictive Control Made Easy* (Vols. Intelligent Systems, Control and Automation: Science and Engineering 69, págs. 41-56). Dordrecht: Springer Science+Business Media.

Van Breemen, T. D. (2001). Design and implementation of a room thermostat using an agent based approach. *Control Eng. Practice*, , 233-248.

Venkat, A. N. (2005). Stability and Optimality of distributed Model Predictive Control. *IEEE Conference on Decision and Control / IEE European.* IEEE/IEE European.

Watkins. (1989). *Learning from Delayed Rewards. Doctoral dissertation.* Cambridge, United Kingdom: University of Cambridge.

WIDE - 224168 - FP7-ICT-2007-2. (2007). *Proyect final report.* Recuperado el 18 de 03 de 2015, de WIDE - 224168 - FP7-ICT-2007-2: http://cse.lab.imtlucca.it/hybrid/wide/index.php?p=deliverables

WIDE - 224168 - FP7-ICT-2007-2. (2009). *Decentralized Wireless Control of Large-Scale Systems.*

Woolridge, J. (1995). Agent theories, architectures, and languages: a survey. *ECAI-94 Workshop on Agent Theories.* Architectures and Languages Eds.