UNIVERSITÀ DEGLI STUDI DI CAGLIARI

Dipartimento di Ingegneria Elettrica ed Elettronica
Corso di Laurea in Ingegneria Elettronica

Master graduation thesis

Collision Avoidance Using Decentralized Hierarchical Supervisory Control

Supervisors:
Prof. Alessandro Giua
Prof. Jörge Raisch

Student:
Michele Cau

Academic year
2011-2012

To my family, my girlfriend and my dogs,
hugs, kisses and cuddles had been necessary
to arrive here.
To my friends and colleagues,
our laughs have lighted my work.
To Berlin,
it taught me that is always possible
to improve ourselves.
Ich bin ein Berliner

Contents

Chapter 1

Introduction

The collision avoidance between robots is a classical issue to solve when there are some agents which have to share some resources. In literature there are many solutions to this problem, from a software approach to the construction of an unique supervisor for the whole field, but we want to follow a different strategy and to find a systematic method to control this kind of system without sharing all information in each robot.
Based on previous works made in Berlin at the *Technische Universität* by professor *Jörg Raisch*, it has been used a hierarchical supervisory control structure modelled as a framework to solve practical problems, such as example, concerning robots moving on the same area. This approach should show how a complex problem, as the example is, could be solved easily through some abstractions of the system.

1.1 The Multi-agent rendezvous problem

The example to test in this approach is showed in fig.1.1, and consists of a field into there are some agents (4 in the image) which want to arrive in their goal positions as soon as possible avoiding collisions. The system is described by the following features:

- the *number of agents* (n);
- the dimension of the field, in terms on x/y *length*;
- positions of goals, that are points to reach inside the field.

These *variables* are determined before the beginning of the agents movement. The main goal of this thesis is to create an approach which is not completely

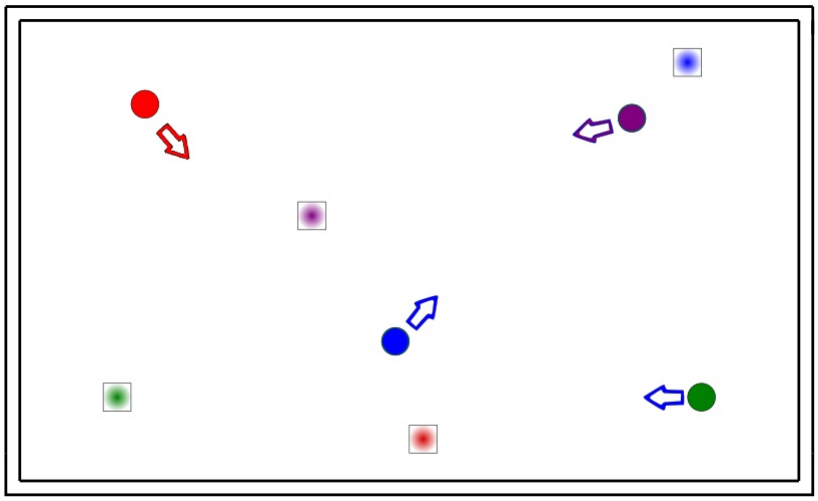


Figure 1.1: Generic system

dependent on these constraints, it means that the structure is not dependent on previous variables. *Agent point of view* does not know the exact movements of agents as information; that is, each agent is not interested in knowing and memorizing the exact movements of the other ones and its movement must allow for this *uncertainty* of the system. In [20] is shown a coordination strategy provided with a non-synchronous decentralised structure where a coordination strategy solution has been developed.
The practical example was implemented with a *Matlab-Simulink* model with a **GUI** [1] and based on a *hierarchical supervisory approach*.
It is supposed:

- all agents have a *common point of reference* in the field called *origin* (O);
- each car can determine its own position using *sensors* related to O;
- robots are able to exchange each others simple informations.

All these points may be better analysed in the realization step or better when real robots are chosen for a real application.

[1] GUI = graphic unit interface

1.2 Elements of theory

The chosen approach is the result of studies started by *P.J.G. Ramadge* and *W. M. Wonham* with discrete events [12] [11] to which has been extended afterwards to hybrid systems and their supervisory theory [13] [15] [16] [17]. We want to control a *multi-agent system* modelled as a set of continue equations for each agent through their individual admissible *behaviours* may be done. Each agent is *limited* into possible actions in a *abstract field* where agents have to work simultaneously, so a first *abstraction* of the plant creates a first level of discrete behaviour description. This procedure may be repeated in order to create a *hierarchical control structure* that builds a *decentralized supervisory control architecture* for the problem [18]. All these aspects are examined in Cap. 2. Following are briefly shown some theoretical concepts.

1.2.1 Behavioural point of view

The events are described in a mathematical language by answering to which set do the (unmodelled) events belong. The *universum* of events that are, in principle, possible is denoted by $\mathfrak{U}$. After studying the situation, the conclusion is reached that the events are constrained, that some laws are in force. Expressing this restriction leads to a *model*. Modelling therefore means that certain events are declared impossible, that they cannot occur. The possibilities that remain constitute the *behaviour* of the model, and is denoted by $\mathfrak{B}$. Every *dynamical system* (Σ) has a behaviour and it is described as:

$$\Sigma = (T, W, \mathfrak{B})$$

where:

- $T \subseteq \mathbb{R}$ is the *time set*,
- W is the *signal space*,
- $\mathfrak{B} \subseteq W^T$ is the *behaviour*, a set of permitted events called *trajectories*.

A generic trajectory (w) could be:

- **allowed** if $T \to W \in \mathfrak{B}$;
- **forbidden** if $T \to W \notin \mathfrak{B}$.

In section 2.2 are shown the main proprieties and how they can characterize a system, all notes are based on previously works [3] and [11].

1.2.2 Supervisory theory

The *supervisory control theory* focus on how to limit a system behaviour through *measurements* from the physical system. Measurements are made by *sensors* and *actions* are driven by *actuators* . Limiting actions means to know how all possible *evolutions* of a system are and to create a subset of allowed evolutions for each measurement done. Moreover, actions must be imposed and the system must react to them. This theory is well studied in [11] for *DES's systems* where a no-empty *controlled DES* is defined as:

$$\mathbf{G} = (Q, \Sigma, \delta, q_o, Q_m)$$

where:

- Q is the *state set* mostly countable,
- $\Sigma = \Sigma_c \bigcup \Sigma_m$ is the union of controllable and uncontrollable events called *alphabet*,
- $\delta : Q \times \Sigma \to Q$ is the *transition function*,
- q_0 is the *initial state*,
- Q_m is the set of *possible final states*, $Q_m \subseteq Q$.

A particular subset of events can be selected by specifying a subset of controllable events to be enabled. It is convenient to adjoin with this all the uncontrollable events as these are automatically enabled. Each subset of events is a *control pattern* , the set of them is defined as:

$$\Gamma = \gamma \in Pwr(\Sigma) \mid \gamma \supseteq \Sigma_u$$

A *supervisory control* for $\mathbf{G}$ is any map $V : L(\mathbf{G}) \to \Gamma$. The pair $(\mathbf{G}, V)$ will be written $V\backslash G$, to suggest that $\mathbf{G}$ is under the supervision of V. The *closed behaviour* of $V\backslash G$ is defined to be the language $L(V\backslash G) \subseteq L(\mathbf{G})$ and it could be used as new language if some proprieties are verified, i.e. $V\backslash G$ must be non-blocking for $\mathbf{G}$. In Section 2.2 are some interesting properties and how this theory is implementable trough an automata even if the starting system is a *hybrid system* [19]. The central topic of this work is to enforce different specifications to a multi-agent system so that *conjunction of supervisors* theory is approached.

1.3 Hierarchical Control Theory

Complexity represents the main obstacle in many control problems, and it is common engineering knowledge that suitable decomposition techniques form a necessary ingredient for any systematic treatment of complex control problems. Hierarchical approaches, where several control layers interact, are a particularly attractive way of problem decomposition as they provide an extremely intuitive control architecture. Complexity problems are especially pronounced for hybrid control synthesis problems, and this has motivated usefulness in multi agent system topic. This approach has been modelled at university of Berlin as a hierarchical control synthesis framework which is general enough to encompass both continuous and discrete levels. It works without any heuristic approach. Each layer in this framework is thought in order to encapsulate the lower layer but they must be designed with the engineer intuition. Fig. 1.2 shows an application of this general approach

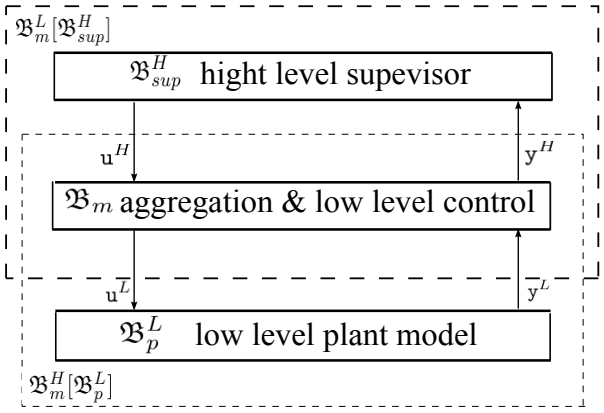


Figure 1.2: Example: two level modular architecture

where a middle interface is needed to link two levels. In chapter 3, it is reported the general idea to this approach and how is applicable to solve the problem through abstraction. There are many examples that show how physical systems could be abstracted, i.e. [21]. Beyond the generic theory, in sections 3.3 and 3.4 is explained how abstractions have been planned and how different supervisors work in the hierarchical structure.

1.4 The model

The developed model must respect specifications illustrated in chapter 3 so the whole model is easily decomposable in blocks where each one implements

a particular specification. The structure is modelled *agents-independently*:

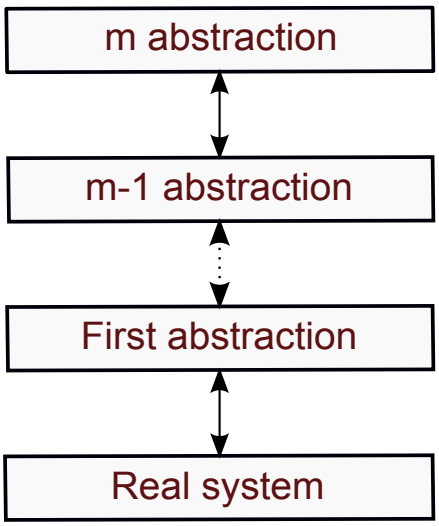


Figure 1.3: Modular project

it means that it is not important the number of agents in the system, even though there is a block for each agent with its own personal first layer abstraction. The implemented structure is logically planned as figure 1.3. The Model is realized in *Matlab-Simulink* environment, the logical part is realized with *Stateflow* (http://www.mathworks.de/help/toolbox/stateflow/index.html), this toolbox permits to solve easily *logical supervised problems*. Moreover, the model has a *GUI*, made with the *GUIDE Matlab graphical interface*, that is a particular tool capable to launch a Matlab-Simulink model, memorizing results and plotting them in tables, labels or graphics. Further informations can be found in [5], [6], [7], [8], [7] and [9]. Chapter 4 contains all the details about the way the model is built and what abstraction level the specifications belong to. A special focus is given on automaton which enforce specifications and on how agents movement is limited.

1.5 Test and results

The goal of this work is to test this new adopted approach in complex projects where number of agents and type of field is not totally fixed. This is the reason why tests that have been done are function of the 3-dimensions space to test. Clearly, this space is still composed of:

- number of agents,

- field dimension,
- goal positions.

Tests are reported in chapter 5 with a special focus on to the purpose they have been made. Lastly some tests about tricky situation are reported and they show the real behaviour of the model.

Chapter 2

Elements of Theory

The chosen approach in this work embraces many topics in automatic and computer science field, the purpose of this chapter is explaining basic concepts which are basilar in this work. Section 2.1 introduces the *multi-agent system* concept with emphasis on *no-synchronous* agents and on what the literature proposes. However, it is wanted to apply a supervisory policy based on the *behaviour* point of view witch properties are showed in section 2.2. At least, section 2.3 talks about how a supervisor works with the behaviour theory, some proprieties must be respected in order to have a realizable supervisor.

2.1 The rendezvous problem

Multi-agents systems (MAS) are a class of systems where different entities, called *agents*, work sharing resources and cooperating together to achieve a common goal. These systems catch our attention because are based on a general concept and are useful in many research fields such as robotics and sociology. An agent is basically an *independent* system with own describing laws and dynamic which has to solve/achieve a private goal, it has to interact with its working field through *sensors* and to take decisions. Moreover, typical advantages of MAS are:

- faster solution for complex problem through its decomposition;
- toughness to noise or fault;
- each agent is projected for each specific sub-problem.

The *multi-agent rendezvous problem* is to devise "local" control strategies, one for each agent, which without any active communication between agents,

it means to avoid communication between them so not that to cope with communication problem. A possible approach is given in [20], it consist of creating a *sensing region* around agents and studying strategies when a overlapping appends, figure 2.1 shows the situation faced in the paper.

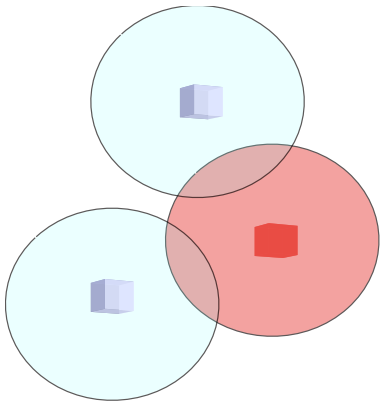


Figure 2.1: Overlapping example between three agents

Two types of strategies are possible for solving the problem. The first consists of agent strategies which are mutually synchronized in the sense that all depend on a common clock, the second consists of strategies which can be implemented independent of each other, without reference to a common clock. Obviously, sharing a clock signal let us consider a system as not truly distributed because each agent decision must be synchronize. Thus, it is interesting studying a *completely distributed system* with moving agents with *stop-and-go strategies*. Let us define:

- τ_{M_i} as *manoeuvring period*, it is the time of a complete movement from a stable position to another one;
- τ_D as *dwell time*, the time in witch an agent is stationary grater than τ_{M_i};
- τ_S as *sensitive time* or rather a brief piece of time when agents stops before to accomplished their work

In [20], it is supposed a stop-and-go policy for agent, it means each agent had decided previously the plan of its movement and if a next movement is not possible it has to stop and wait. According to this it is possible to have an asynchronous movement if $\tau_D and \tau_S$ are fixed parameter equal in every agent and it is satisfied

$$\tau_S \leq \frac{1}{2}(\tau_D - \tau_{M_i}) \tag{2.1}$$

Although all agents use the same dwell time, they function asynchronously in the sense that the time sequences t_1,t_1,$\cdots$,t_n are uncorrelated. Thus, each agent strategy could be implemented separately and independently.
Each agent has some *registered neighbour*that those agents in its sensitive area for at least τ_S seconds, sensors capture their positions and memorize them for purpose of calculation, evidently if an agent j is a sensitive neighbour for i even it will be true vice versa. Equation 2.1 imposes the maximum overlapping time for each pair of considered agents, agents must be in this critical situation for τ_S at least.
In [20] each agent studies the *evolution of the filed* through memorizing its neighbour list in each k_i^{th} manoeuvring period as

$$x_i(t_{k_i}) = x_i(t_{k_{i-1}}) + u_m(z_1, z_2, \ldots, z_m) \tag{2.2}$$

where:

- $x_i(t_{k_i})$ is the current position at t_{k_i} time period;
- $x_i(t_{k_{i-1}})$ is the past position at $t_{k_{i-1}}$ time period;
- $u_m(z_1, z_2, \ldots, z_m)$ is the movement, just done, function of the set of neighbours positions.

The main property of this is that $u_m(\bar{z})$ could be expressed as 2.2 for each neighbour and it is easy to elaborate information in order to achieve the personal agent goal. However, it is still present co-ordination issue, it is turned into a *communication* problem but it is possible to study the whole system trough 2.2 by DESs system theory, i.e. the graph theory.

2.2 Behaviour definition and proprieties

The concept of *behaviour* is basilar for the supervisory control approach, we can define a *mathematical model* $\mathfrak{M}$ as:

$$\mathfrak{M} \Longleftrightarrow (\mathfrak{U}, \mathfrak{B})$$

where:

- $\mathfrak{U}$ is the *universum of events*, it is the set of all possible events;
- $\mathfrak{B}$ is the *behaviour*, a subset of possible events such as $\mathfrak{B} \subseteq \mathfrak{U}$ or rather the outcomes which the original model allows.

If $\mathfrak{U}$ is a finite set, or strings of elements from a finite set, we speak about discrete event systems (DESs), if $\mathfrak{U}$ is a (subset of) a finite-dimensional real (or complex) vector space, we speak about continuous models and if $\mathfrak{U}$ is a set of functions of space and time, we speak about distributed parameter systems [3].

Definition 2.1 (Dynamical system definition)**.** In dynamical systems (Σ) *events* are maps from the *time domain* to the *signal space co-domain* and a behaviour is a set of events, for this it is convenient to distinguish these elements

$$\Sigma = (T, W, \mathfrak{B})$$

where:

- $T \subseteq \mathbb{R}$ is the *time set*,
- W is the *signal space*,
- $\mathfrak{B} \subseteq W^T$ is the *behaviour*, a set of permitted events called *trajectories*.

A generic trajectory (w) could be:

- **allowed** if $T \to W \in \mathfrak{B}$;
- **forbidden** if $T \to W \notin \mathfrak{B}$.

Based on T we can distingued sistems in

- *continuous time systems* with $T = \mathbb{R}$ o $T = \mathbb{R}_+$;
- *discrete time systems* with $T = \mathbb{N}$ o $T = \mathbb{Z}$;

2.2.1 Property of dynamical systems

Dynamical systems $\Sigma = (T, W, \mathfrak{B})$, such as all mathematic approch, has some propreties which defined some categories, in a row the most important are shown, a dynamical system is

Definition 2.2 (*Linear)***.**

$$w_1, w_2 \in \mathfrak{B} \text{ and } \alpha \in \mathbb{F}] \Rightarrow [w_1 + \alpha w_2 \in \mathfrak{B}]$$

if two trajectories in a field $\mathbb{F}$ are in the set of the behaviours than a linear combination of them is in also.

$$[w \in \mathfrak{B} \text{ and } t \in T] \Rightarrow [\sigma^t w \in \mathfrak{B}]$$

where σ^t is a *time-shift* defined as:

Definition 2.3 (*Time-invariant).*

$$\sigma^t w : T \to W, \sigma^t w(t\prime) := w(t\prime + t)$$

so the trajectory doesn't depend on the time of strarting.

Definition 2.4 (*Autonomus* system).

$$[w_1, w_2 \in \mathfrak{B} \text{ and } w_1(t) = w_2(t) \text{ for } t < 0] \Rightarrow [w_1 = w_2]$$

so the past evolution implies the future evolution.

Definition 2.5 (*Stability*). The dynamical system $\Sigma = (T, W, \mathfrak{B})$, with $T = \mathbb{R}$, $[0, \infty)$, $\mathbb{Z}$, or $\mathbb{N}$, and $\mathbf{W}$ a normed vector space (for simplicity) is *stable* if

$$[w \in \mathfrak{B}] \Rightarrow [w(t) \to 0 \text{ for } t \to \infty]$$

W is a normed vector, than all trajectories go to 0, it is called *asymptotic stability.*

Definition 2.6 (*Controllability*). Given $T = \mathbb{R}$ or $\mathbb{Z}$, Σ is *controllable* if for all w_1, $w_2 \in \mathfrak{B}$ there exist $T \in \mathbb{T}$, $T \geq 0$, and $w \in \mathfrak{B}$, such that

$$w(t) = \begin{cases} w_1(t) & \text{for } t < T \\ w_1(t-T) & \text{for } t \geq T \end{cases}$$

This property implies an important property: *contorllability* $\Longleftrightarrow$ *concatenability of trajectories after a delay*. A practical situation is shown in fig.2.2, a period of time is needed to switch from a behaviour to another one and it is a sensible point to analyse in a project.

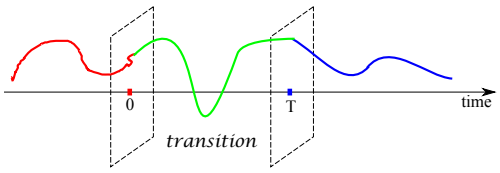


Figure 2.2: Pratical example of concatenability

Definition 2.7 (*Stabilizability*). The dynamical system Σ with $T = \mathbb{R}$ or $\mathbb{Z}$ is stabilizable when for all $w \in \mathfrak{B}$ there exist $w' \in \mathfrak{B}$, such that:

$$w'(t) = \begin{cases} w_1(t) & \text{for } t < 0 \\ \to 0 & \text{for } t \to \infty \end{cases}$$

Definition 2.8 (*Observability).* Let us consider the system

$$\Sigma = (\mathbb{T}, \mathbb{W}_1 \times \mathbb{W}_2, \mathfrak{B})$$

where W_1 is the set of observed behaviours and w_2 is the set of deduced behaviours. w_2 is said *observable* from w_1 in Σ if

$$[(w_1, w_2), (w_1', w_2') \in \mathfrak{B} \text{ and } w_1 = w_1'] \Rightarrow [w_2 = w_2']$$

Equivalently, it means that there exist a map $F : \mathbb{W}_1^{\mathbb{T}} \to \mathbb{W}_2^{\mathbb{T}}$ such that $[(w_1, w_2) \in \mathfrak{B}] \Rightarrow [w_2 = F(w_1)]$.

Definition 2.9 (*Detectability).* Consider $\Sigma = (\mathbb{T}, \mathbb{W}_1 \times \mathbb{W}_2, \mathfrak{B})$ with $\mathbb{T} = \mathbb{N}, \mathbb{R}, \mathbb{R}_+$ or $\mathbb{Z}$. A behaviour w_2 is said *detectable* from w_1 in Σ if

$$[(w_1, w_2), (w_1', w_2') \in \mathfrak{B} \text{ and } w_1 = w_1'] \Rightarrow [w_2(t) - w_2'(t) \to 0 \text{ for } t \to \infty]$$

Thus, w_2 can be asymptotically deduced from w_1.

2.2.2 Input-Output representation

Behaviours and systems can be described easily through properties shown in Par. 2.2.1. However, models are often create to replace unknown/complex set of equations and they must have a behavioural representation, different to the original equations plainly, called *behavioural equations*. This point of view supposes the real system as a *black box* and follows paradigms shown in [4] where equation representation is discussed.

Definition 2.10 (*Behavioural equation representation).* Let $\mathbb{U}$ be the universum, $\mathbb{E}$ an abstract set called the *equating spaces*, and $f_1, f_2 : \mathbb{U} \to \mathbb{E}$. The mathematical model $\mathfrak{M} =| (\mathbb{U}, \mathfrak{B})$ with $\mathfrak{B} = \{u \in \mathbb{U} \mid f_1(u) = f_2(u)\}$ is saied to be described by *behavioural equations* and $(\mathbb{U}, \mathbb{E}, f_1, f_2)$ is the *behavioural equation representation* of $(\mathbb{U}, \mathfrak{B})$.

Moreover, let $f_1(u) = f_2(u)$ be defined *equilibrium conditions* as particular conditions which many models are concerned to. There exist a huge class of systems described by *behavioural inequalities* where describing equations are $f_1(u) \leq f_2(u)$ and equilibrium conditions are their central topic. Even DESs are described through behavioural inequalities, a map f is set in order to have $\mathfrak{B} = \{y \in \mathbb{U} \mid f(u) = 0\}$ until a precise value is not reached. Clearly, a model described by a behaviour is more general than a real system, even though a behaviour is derivable from a set of equations, the opposite procedure is not possible. Thus, this approach has the main idea: it is the behaviour, the solution set of the behavioural equations, not the behavioural equations themselves, which is the essential result of a modelling procedure.

Given a system Σ described as 2.2 , the element $\mathbb{T}$ defines the instants in which events append and represents the time space in which the system produces values. The behaviour $\mathfrak{B}$ is a family of $\mathbb{W}-valuedtimetrajectories$ that defines the setting paired with $\mathbb{T}$ that describe the mathematization of the problem. Dynamical systems have some general properties to be reported in this framework, i.e. linearity.

Definition 2.11 (*linearity).* A dynamical system $\Sigma = (\mathbb{T}, \mathbb{W}, \mathfrak{B})$ is *linear* if $\mathbb{W}$ is a vector space, i.e. $\mathbb{F} = \mathbb{R}$, and $\mathfrak{B}$ is a linear subspace of $\mathbb{W}^{\mathbb{T}}$ even valid with addiction and multiplication by scalar. Thus, linear systems obey the *superposition principle* in his simple form:

$$\{w_1(\cdot), w_2(\cdot) \in \mathfrak{B};\ \alpha, \beta \in \mathbb{F}\} \Rightarrow \{\alpha w_1(\cdot) + \beta w_2(\cdot) \in \beta\}$$

Since time invariance will play an important role in the sequel, a definition is given:

Definition 2.12 (*Time invariant dynamical system).* A dynamical system $\Sigma = (\mathbb{T}, \mathbb{W}, \mathfrak{B})$ with $\mathbb{T} = \mathbb{Z}$ or $\mathbb{Z}$ is said to be *time invariant* if $\sigma^t\mathfrak{B} = \mathfrak{B}$ for all $t \in \mathbb{T}(\sigma^t$ denotes the backward t-shift: $(\sigma^t f)(t') := f(t' + t))$. If $\mathbb{T} = \mathbb{Z}$ then this condition is equivalent to $\sigma\mathfrak{B} = \mathfrak{B}$.

The analog of this definition when the time axis is $\mathbb{Z}_+$ or $\mathbb{R}_+$ require $\sigma\mathfrak{B} \subseteq \mathfrak{B}$. A behaviour has an evolution, often independent of the time. Each occurred event has to be signal in the own behaviour-space, if it is verified the behaviour has the *completeness* property.

Definition 2.13 (*completeness property).* A behaviour $\mathfrak{B} \subseteq W^{\mathbb{N}_0}$ is *complete* if for all $w \in W^{\mathbb{N}_0} : w \in \mathfrak{B} \Longleftrightarrow \forall k \in \mathbb{N}_0 : w\mid_{[0.k]} \in \mathfrak{B}\mid_{[0,k]}$

However, a new characteristics must be add for modelling real systems, it is compulsory to add an *input - output behaviours* definition (basilar in [2], [4]).

Definition 2.14 (*I/- behaviours).* A behavioural $\mathfrak{B} \subseteq W^{\mathbb{N}_0}$ over the signal space $W = W_{in} \times W_{out}$ is an *I/- behaviour* if:

(i) the input is *free*, i.e. $\mathcal{P}_{in}\mathfrak{B} = W_{in}^{\mathbb{N}_0}$; and

(ii) the output *does not anticipate* the input, i.e. for alla $k \in \mathbb{N}_0, \tilde{w}, \hat{w} \in \mathfrak{B}$ with $\mathcal{P}_{in}\tilde{w} \mid_{[0,k]} = \hat{w} \mid_{[0,k]}$ there exists a $w \in \mathfrak{B}$ such that $\mathcal{P}_{out}w \mid_{[0,k]} = \mathcal{P}_{out}\hat{w} \mid_{[0,k]}$ and $\mathcal{P}_{in}w = \mathcal{P}_{in}\hat{w}$.

where $\mathcal{P}$ are natural projections from $W^{\mathbb{N}_0}$ to the input and output component respectively.

In control theory the *non-anticipating* condition is compulsory to be required unlike other subject as signal processing. A practical definition is given with focus on relationships between signals and time.

Definition 2.15 (*Non-anticipating system).* Consider the time-invariant dynamical system $\Sigma = (\mathbb{T}, \mathbb{W}_1 \times \mathbb{W}_2, \mathfrak{B})$ with $\mathbb{T} = \mathbb{Z}$ or $\mathbb{R}$. We will say w_2 does not anticipate w_1 if $\{(w_1', w_2'), (w_1'', w_2'') \in \mathfrak{B}$, and $w_1'(t) = w_1''(t)$ for $t \leq 0\} \Rightarrow \{\exists w_2$ such that $(w_1'', w_2) \in \mathfrak{B}$ and $w_2(t) = w_2'(t)$ for $t \leq 0\}$.

Moreover, the input must be *free*, it means an input signal could be any value and the system can follows any input trajectory; the system can be defined as follow.

Definition 2.16 (*free systems).* A time-invariant dynamical system $\Sigma = (\mathbb{T}, \mathbb{W}, \mathfrak{B})$ with $\mathbb{T} = \mathbb{Z}$ or $\mathbb{R}$ is said to be *locally free* if it is trim (i.e. $\forall w \in \mathbb{W}$, $\exists w \in \mathfrak{B}$ such that $w(0) = w$) and *memoryless*. It is said to be *free* if addition Σ is complete.

Hence, free implies no local constrains (trim), no memory (memoryless) and no constrains at $t = \pm\infty$ (complete)

2.3 Supervisory control theory

A behavioural approach makes easy a supervisory control for every systems which are described by a behaviour. Now the problem is turned into a behaviour limitation issue. Generally, plants are continues and [13] [22] but if we assume that this external interface is given, the problem of supervisory controller synthesis can be discussed in analogy to [12] and [11] as DESs. It possible to follow a well-thinking approach [2] for the supervisors design to adapt DES techniques to the typical input/output structure of continuous systems.

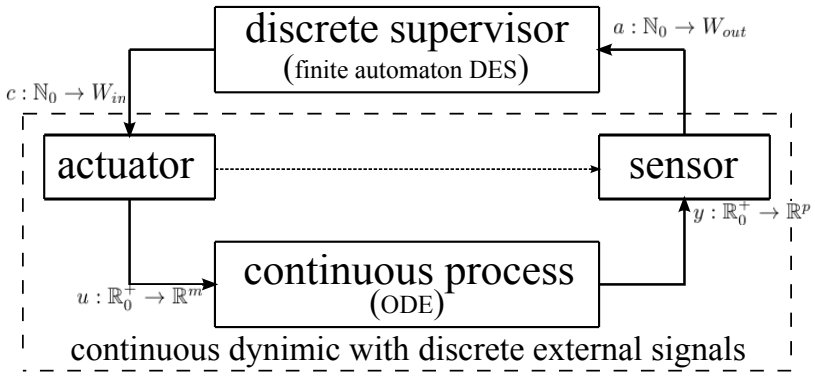


Figure 2.3: Hybrid closed-loop system

Definition 2.17. A Behaviour $\mathfrak{B}$ over a signal space W is now definite as a set of maps $w : \mathbb{N}_0 \to W$; i.e. $\mathfrak{B} \subseteq W^{\mathbb{N}_0}$

An example of a hybrid closed-loop configuration is depicted in Fig. 2.3, mathematical details are developed in [23] including configurations based on clock time (events occur at a fixed sampling rate) and logic time (events may occur at any time). It is assumed for this approach that the plant inherits the input/output structure from the underlying continuous system (i.e. cross critical thresholds). A dynamical system is a model of a phenomenon and a behaviour is the set of all trajectories on which the phenomenon can, according to the model, possibly evolve.

2.3.1 Supervisory control of I/- behaviours

Supervisory control theory starts from the *I/- behaviour* definition, def. 2.2.2, for witch Ramadge-Wonham theory [24] provides a mathematical framework for design and computations of the supervisory controller of discrete event systems DES). Two different of behaviours are involved from the real system, the *plant behaviour* $\mathfrak{B}_p \subseteq W^{\mathbb{N}_0}$ that are of trajectories the system can evolve without restrictions, and the set of acceptable signals, *specification*, denoted $\mathfrak{B}_{spec} \subseteq W^{\mathbb{N}_0}$. In analogy to the plant, the supervisor is represented by a behaviour $\mathfrak{B}_{sup} \subseteq W^{\mathbb{N}_0}$, denoting the set of external signals it can evolve on. The closed-loop behaviour is the intersection $\mathfrak{B}_{cl} = \mathfrak{B}_p \cap \mathfrak{B}_{sup}$ and the supervisor $\mathfrak{B}_{sup}$ is said to *enforce the specification* $\mathfrak{B}_{spec}$ if $\mathfrak{B}_{cl} \subseteq \mathfrak{B}_{spec}$.

However, in [13] are shown two *admissibility criteria* in terms of behaviour must be respected for the interconnection of $\mathfrak{B}_p$ and $\mathfrak{B}_{sup}$. The first condi-

tion on on behavioural interconnection ensures that the synchronisation of external signals is performed *locally on the time axis*: in every time instant, given a trajectory, plant and supervisor must agree in their future evolution. This requirement correspond to the non-conflicting languages in DES theory.

Definition 2.18 (*Non-conflicting languages).* Two behaviours $\mathfrak{B}_p, \mathfrak{B}_{\mathfrak{sup}} \subseteq \mathfrak{W}^{\mathbb{N}_0}$ are said to be *non-conflicting* if $\mathfrak{B}_p \mid_{[0,k]} \cap \mathfrak{B}_{sup} \mid_{[0,k]} = (\mathfrak{B}_p \cap \mathfrak{B}_{sup}) \mid_{[0,k]}$ for all $k \in \mathbb{N}_0$.

The second criteria is valid specifically for I/- behaviours, despite of the possibility to limit input can not impose restriction on the plant output.

Definition 2.19 (*Generically implementability).* A behaviour $\mathfrak{B}_{sup} \subseteq W^{\mathbb{N}_0}$, $W = W_{in} \times W_{out}$, is *generically implementable* if $k \in \mathbb{N}_0, w \mid_{[0,k]} \in \mathfrak{B}_{sup} \mid_{[0,k]}$, $\tilde{w} \mid_{[0,k]} \in W^{k+1}$, and $\tilde{w} \mid_{[0,k]} \approx_y w \mid_{[0,k]}$ implies $\tilde{w} \mid_{[0,k]} \in \mathfrak{B}_{sup} \mid_{[0,k]}$.

It is possible to give a formal definition to the problem ([2],[13]) that include previous definitions.

Definition 2.20 (*Supervisory control problem).* Given a plant behaviour $\mathfrak{B}_p \subseteq W^{\mathbb{N}_0}$, $W = W_{in} \times W_{out}$, and a specification $\mathfrak{B}_{spec} \subseteq W^{\mathbb{N}_0}$, we call the pair $(\mathfrak{B}_p, \mathfrak{B}_{spec})$ a *supervisory control problem.* Is is possible to say:

i a supervisor $\mathfrak{B}_{sup} \subseteq W^{\mathbb{N}_0}$ is *admissible* with the respect to the plant $\mathfrak{B}_p$ if $\mathfrak{B}_p$ and $\mathfrak{B}_{sup}$ are non-conflicting (def. 2.3.1) and $\mathfrak{B}_{sup}$ is generically implementable;

ii a supervisor $\mathfrak{B}_{sup} \subseteq W^{\mathbb{N}_0}$ *enforces the specification* $\mathfrak{B}_{spec}$ if $\mathfrak{B}_{cl} := \mathfrak{B}_p \cap \mathfrak{B}_{sup} \subseteq \mathfrak{B}_{spec}$;

iii a supervisor $\mathfrak{B}_{sup}$ is a *solution* of $(\mathfrak{B}_p, \mathfrak{B}_{spec})$ if $\mathfrak{B}_{sup}$ is admissible to $\mathfrak{B}_p$ and enforces $\mathfrak{B}_{spec}$;

iv a solution $\mathfrak{B}_{sup}$ in *non-trivial* if it imposes a non-trivial closed-loop behaviour $\mathfrak{B}_{cl} \neq \emptyset$ [1].

It is possible to give a proof concerning the presence of a solution if $\mathfrak{B}_{sup} \neq \emptyset$ than $\mathfrak{B}_p \cap \mathfrak{B}_{sup} \neq \emptyset$.

Proof. Pick $w \in \mathfrak{B}_{sup}$ and the input of $\mathfrak{B}_p$ is free, there exists $\tilde{w} \in \mathfrak{B}_p$ with $\mathcal{P}_{in}\tilde{w} = \mathcal{P}_{in}w$, and, hence, $\tilde{w} \mid_{[0,k]} \approx_y w \mid_{[0.0]}$. From generic implementability it is obtained $\tilde{w} \mid_{[0,0]} \in \mathfrak{B}_{sup} \mid_{[0,0]} \cap \mathfrak{B}_p \mid_{[0,0]}$. The non-conflicting property ensures that $\tilde{w} \mid_{[0,0]} \in (\mathfrak{B}_{sup} \cap \mathfrak{B}_p) \mid_{[0,0]} \neq \emptyset$. Hence, $\mathfrak{B}_{sup} \cap \mathfrak{B}_p \neq \emptyset$. □

[1] obviously it is the only possible trivial solution

Certainly the trivial solution is unacceptable in almost any application context, it facilitates a set-theoretic lattice argument that establishes the unique existence of a least restrictive solution. Given all possible solution of the problem $(\mathfrak{B}_p, \mathfrak{B}_{spec})$ there exists a *last restrictive solution* $\mathfrak{B}^{\uparrow}_{sup}$ and it is an upper semi-lattice with operators $\cup$ and $\subseteq$. Proof is given in [2], it explains that the least restrictive supervisor $\mathfrak{B}^{\uparrow}_{sup}$ contains all other solutions by $\cup$ operation [2].

If both $\mathfrak{B}_p$ and $\mathfrak{B}_{spec}$ are realised by finite automaton it is possible to compute a least restrictive solution $\mathfrak{B}^{\uparrow}_{sup}$ by modifying the standard theory [11]. Real applications have never a finite representation, so models approximate real systems and behaviours are obtained through *abstractions* in order to have a finite automaton with behaviour $\mathfrak{B}_{ca}$. Thus, $\mathfrak{B}_{ca}$ is an *abstraction* if $\mathfrak{B}_p \subseteq \mathfrak{B}_{ca}$ and the solution $\mathfrak{B}_{ca}$ carry over $\mathfrak{B}_p$. All behaviours realised by finite automata are *complete* (Def. 2.2.2), unfortunately completeness can not be always determined by a finite computational procedure so approximations are nearly compulsory even for a discrete complex behaviour. However, if both $\mathfrak{B}_p$ and $\mathfrak{B}_{sup}$ are complete (Def. 2.2.2) and generically implementable (Def. 2.3.1), then they are non-conflicting (Def. 2.3.1). A proof is give in [2].

Proof. Pick any $k \in \mathbb{N}_0, w \mid_{[} 0, k] \in \mathfrak{B}_p \mid_{[0,k]} \cap \mathfrak{B}_{sup} \mid_{[0,k]}$. Without loss of generality, we may assume $w \in \mathfrak{b}_{sup}$. Pick some $\tilde{w} \in \mathfrak{B}_p$ with $\tilde{w} \mid_{[0,k]} = w \mid_{[0,k]}$. Using the I/- property of B_p and the generic implementability of B_sup, we can construct $\tilde{w} \in W^{\mathbb{N}_0}$ with $\hat{w} \mid_{[0,k+1]} \in \mathfrak{B}_p \mid_{[0,k+1]} \cap \mathfrak{B}_{sup} \mid_{[0,k+1]}$ and $\hat{w} \mid_{[0,k]} = w \mid_{[0,k]}$. As we have started with an arbitrary $w \mid_{[} 0, k] \in \mathfrak{B}_p \mid_{[0,k]} \cap \mathfrak{B}_{sup} \mid_{[0,k]}$, we can iterate our construction to obtain a sequence of trajectories $(w^k)_{k \in \mathbb{N}_0}$, $w^k \in W^{\mathbb{N}_0}$, with $w^{k+1} \mid_{[0,k+\mathsf{k}]} = w^k \mid_{[0,k+\mathsf{k}]}$ for all $\mathsf{k} \in \mathbb{N}_0$. Hence, the sequence converges point-wise as the limit $w^\infty \in W^{\mathbb{N}_0}$ and then $w^\infty \mid_{[0,k]} \in \mathfrak{B}_p \mid_{[0,k]} \cap \mathfrak{B}_{sup} \mid_{[0,k]}$ for all $\mathsf{K} \in \mathbb{N}_0$. Completeness of $\mathfrak{B}_p$ and $\mathfrak{B}_{sup}$ implies $w^\infty \in \mathfrak{B}_p \cap \mathfrak{B}_{sup}$ and any trajectory could be seen as $w \mid_{[0,k]} = w^\infty \mid_{[0,k]} \in (\mathfrak{B}_p \cap \mathfrak{B}_{sup}) \mid_{[0,k]}$. □

Generic implementability does not depend on the particular plant, and it is a good point to start when an abstraction $\mathfrak{B}_{ca} \supseteq \mathfrak{B}_p$ is made. In [2] is shown the following theorem.

Theorem 1. Let $\mathfrak{B}_{ca} \subseteq W^{\mathbb{N}_0}$ be an abstraction of a plant $\mathfrak{B}_p \subseteq \mathfrak{B}_{ca}$ and let $\mathfrak{B}_{sup}$ be a complete and non-trivial solution of $(\mathfrak{B}_{ca}, \mathfrak{B}_{spec})$, $\mathfrak{B}_{spec} \subseteq W^{\mathbb{N}_0}$. If $\mathfrak{B}_p$ is a complete I/- behaviour, then $\mathfrak{B}_{sup}$ is a non-trivial solution of $(\mathfrak{B}_p, \mathfrak{B}_{spec})$.

[2] In particular, $\mathfrak{B}^{\uparrow}_{sup}$ is a non-trivial solution if and only if a non-trivial solution exists

Clearly, the theorem has a logical easy proof because $\mathfrak{B}_{sup}$ is projected in order to enforce the specification $\mathfrak{B}_{ca}$. Thus, $\mathfrak{B}_{sup}$ solves the problem $(\mathfrak{B}_p, \mathfrak{B}_{spec})$ because it was already solved by $\mathfrak{B}_{ca}$.

2.3.2 Modular supervisory control

Setting up an overall supervisor by combining a number of individual supervisors is referred to as *modular supervisory control*. There are two potential benefits from modular supervisors. First, it may turn out that the synthesis of individual supervisors and their combination is computationally cheaper than the direct synthesis of an overall supervisor. Second, given a plant, one may set up a library of supervisors and it is guarantee a sort of personalization to the problem. The modular control architecture we consider in this

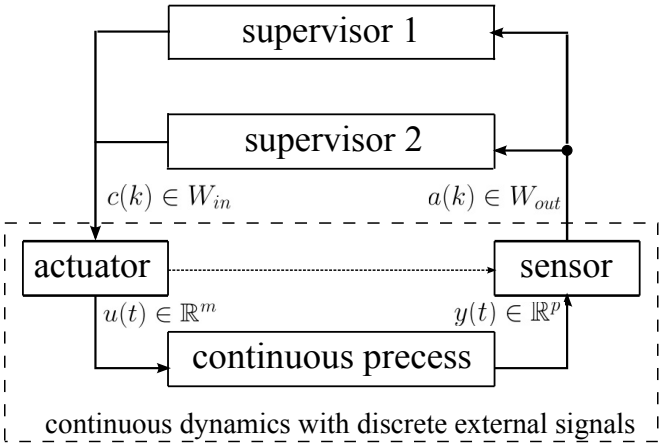


Figure 2.4: Modular control architecture

section is illustrated in Fig. 2.4 and corresponds to the concept modularity that has been proposed for DESs [11]. Is is expected that principle of behaviour theory are still valid for the *modularity* issue.
Given a plant $\mathfrak{B}_p$ now the control problem is required to be solved by two different specifications $\mathfrak{B}_{spec1}$ and $\mathfrak{B}_{spec2}$, both admissible for the plant. The natural idea is to run different supervisors in parallel and to have $\mathfrak{B}_{spec} := \mathfrak{B}_{spec1} \cap \mathfrak{B}_{spec2}$ but unfortunately the property of admissibility is not preserved because it is not guaranteed the *non-conflicting* propriety for the couple of supervisors $(\mathfrak{B}_{spec1}, \mathfrak{B}_{spec2})$ [15].

Definition 2.21 (Non-conflicting supervisor behaviours). Two behaviours $\mathfrak{B}_{spec1} \in \mathbb{N}_0$ and $\mathfrak{B}_{spec2} \in \mathbb{N}_0$ are *non-conflicting relative* to $\mathfrak{B}_p \in \mathbb{N}_0$ if for all $k \in \mathbb{N}_0$:$\mathfrak{B}_p \mid_{[0,k]} \cap \mathfrak{B}_{sup1} \mid_{[0,k]} \cap \mathfrak{B}_{sup2} \mid_{[0,k]} = \mathfrak{B}_p \mid_{[0,k]} \cap (\mathfrak{B}_{sup1} \cap \mathfrak{B}_{sup2}) \mid_{[0,k]}$

It is expected that a least restrictive supervisor will take *no action* outside the plant behaviour, it means every $w \in \mathfrak{B}_{sup} - \mathfrak{B}_p$ implies $w \in \mathfrak{B}_{sup}$. This characteristic because in [2] is proved that, given the behaviours $\mathfrak{B}_p, \mathfrak{B}_{sup1}, \mathfrak{B}_{sup2}$ the last restrictive solution of the supervisory problem $\mathfrak{B}^{\uparrow}_{sup}, \mathfrak{B}^{\uparrow}_{sup}$ are non-conflicting if and only if they are non-conflicting relative to $\mathfrak{B}_p$. If either supervisor are $\mathfrak{B}_{sup1}\mathfrak{B}_{sup2} \subseteq W^{\mathbb{N}_0}$ be generically implementable and non-conflicting then $\mathfrak{B}_{sup} = \mathfrak{B}_{sup2} \cap \mathfrak{B}_{sup2}$ is generically implementable.

It has been shown that different supervisors can cooperate in order to enforce a whole specification, a following problem to solve is adapting this concept to specification that are in different abstraction layer. Rendezvous problems are a perfect example of how different supervisor, one for each agent, must enforce a whole specification, i.e. avoiding collisions.

2.3.3 Decentralised supervisory control

Another step in supervisor theory must be done, the previous section has shown what are lines guide for building supervisors hierarchy, in contrast it is explained now how agents individual enforce specifications simultaneously. Individual supervisors may not share the same measurement information nor may they have the same set of controls at their disposal. The principle aim is to address applications in which communication constraints enforce a decentralised control architecture, although computational benefits for controller synthesis also play a role in the DES literature on this topic [25]. The whole discussion is based on Fig.2.5 and is an evolution of the monolithic case. Let two agents have separate interfaces where the input space is $W_{in} = W_{in1} \times W_{in2}$, the output space is $W_{out} = W_{out1} \times W_{out2}$, external supervisors signal spaces are $W_1 = W_{in1} \times W_{out1}$ and $W = W_{in2} \times W_{out2}$ respectively.
Given two specifications $\mathfrak{B}_{spec1} \subseteq W_1^{\mathbb{N}_0}$, $\mathfrak{B}_{spec2} \subseteq W_2^{\mathbb{N}_0}$ the aim is to design supervisors $\mathfrak{B}_{sup1} \subseteq W_1^{\mathbb{N}_0}$ and $\mathfrak{B}_{sup2} \subseteq W_2^{\mathbb{N}_0}$ which create the closed-loop behaviour

$$\mathfrak{B}_{cl} := \mathfrak{B}_p \cap (\mathfrak{B}_{sup1} \times_\pi \mathfrak{B}_{sup2})$$

where $\times_\pi$ is the product operator that usual product composition plus the obvious re-arrangement of components to fit the scheme imposed by $W = W_{in1} \times W_{in2} \times W_{out1} \times W_{out2}$. If the inclusion $\mathfrak{B}_{cl} \subseteq \mathfrak{B}_p \cap (\mathfrak{B}_{sup1} \times_\pi \mathfrak{B}_{sup2})$ is fulfilled the created supervisors enforces specifications. In this subsection the problem to solve is slightly different from before an it told *decentralized control problem*

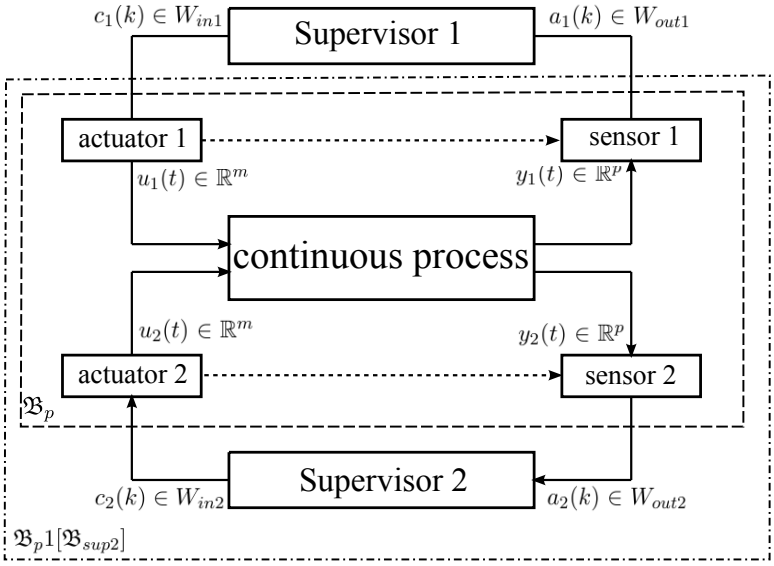


Figure 2.5: Decentralised control architecture

Definition 2.22 (*Decentralized control problem).* The triple $(\mathfrak{B}_p, \mathfrak{B}_{sup1}, \mathfrak{B}_{sup2})$ is said to be *decentralized control problem* where the I/- behaviour $\mathfrak{B}_p \subseteq W^{\mathfrak{N}_0}$ is the plant and $\mathfrak{B}_{spec1} \subseteq W_1^{\mathfrak{N}_0}$, $\mathfrak{B}_{spec2} \subseteq W_2^{\mathfrak{N}_0}$, are specifications for the respective signals. The pair of supervisors $(\mathfrak{B}_{sup1}, \mathfrak{B}_{sup2})$ is a *solution* of $(\mathfrak{B}_p, \mathfrak{B}_{spec1}, \mathfrak{B}_{spec2})$ if $\mathfrak{B}_{sup1} \times_\pi \mathfrak{B}_{sup2}$ solves the supervisory control problem $(\mathfrak{B}_p, \mathfrak{B}_{spec1} \times_\pi \mathfrak{B}_{spec2})$.

The first benefit of previous definition is that 2.3.1 is still valid. Moreover. $\mathfrak{B}_{sup} = \mathfrak{B}_{sup1} \times_\pi \mathfrak{B}_{sup2}$ is *generically implementable* if and only if $\mathfrak{B}_{sup1}, \mathfrak{B}_{sup2}$ are, the proof is logical since elementary properties of product composition and restriction operator do not change the property.
It is not simple understanding if a plant provided by each individual interface is I/- behaviour. Starting from the point of view of a supervisor, $\mathfrak{B}_{sup1}$ the seen environment is $\mathfrak{B}_{p1}[\mathfrak{B}_{sup2}] \subset W_1^{\mathbb{N}_0}$ includes all trajectories $w_1 \in W_1^{\mathbb{N}_0}$ which can possibly occur under the restriction imposed by $\mathfrak{B}_{sup2}$. Formally:

$$\mathfrak{B}_{p1}[\mathfrak{B}_{sup2}] := \{w_1 \in W_1^{\mathbb{B}_0} \mid \exists w_2 \in \mathfrak{B}_{sup2} : (w_1, w_2)_\pi \in \mathfrak{B}_p\}$$

In general, these operations are readily seen not to preserve the input/output structure of $\mathfrak{B}_p$; e.g., if the behaviour $\mathfrak{B}_{sup2}$ consists of one trajectory only, this imposes a rather restrictive condition on the measurement readings from sensor 2, which in turn restricts the input signals to $\mathfrak{B}_p$, such that the input of $\mathfrak{B}_{p1}[\mathfrak{B}_{sup2}]$ cannot be expected to be free. However, there are two conditions

with which $\mathfrak{B}_{p1}[\mathfrak{B}_{sup2}]$ is an I/- behaviour.
Let a simplified problem be considered in which the behavioural restriction arises only from a fixed input signal rather than an entire behaviour $\mathfrak{B}_{sup2}$. Let $\mathfrak{B}_p \subseteq W^{\mathbb{B}_0}$ be an I/- behaviour (w.r.t $W = W_{in} \times W_{out}$). Each possible input c_2 for the second supervisor define a new plant vision for the first supervisor

$$\mathfrak{B}'_{p1}[c_2] := \{(c_1, a_1, a_2) \in (W_{IN1} \times W_{out1} \times W_{out2})^{\mathbb{N}_0} \mid (c_1, c_2, a_1, a_2) \in \mathfrak{B}_p\}$$

Then $\mathfrak{B}'_{p1}[c_2]$ is an I/- behaviour and inherits the *completeness* property from $\mathfrak{B}_p$. This procedure is valid even the point of view of the first supervisor $\mathfrak{B}'_{p2}[c_1]$. If inputs are taken *free* now the behaviour $\mathfrak{B}'_{p1}[c_2]$ is still an I/- behaviour, proof is given in [2]. Starting from results already shown, it is possible to define a procedure that provides a solution to the decentralised control problem as a theorem.

Theorem 2. Given a decentralised control problem $(\mathfrak{B}_p, \mathfrak{B}_{spec1}, \mathfrak{B}_{spec2})$, where the plant $\mathfrak{B}_p$ is a complete I/- behaviour, let:

i $\mathfrak{B}'_{spec2} := \mathcal{P}_{in1}\mathfrak{B}_{spec2} \times W_{out2}^{\mathbb{N}_0}$;

ii $\mathfrak{B}_{sup1}$ solve $(\mathfrak{B}_{p1}[\mathfrak{B}'_{spec2}], \mathfrak{B}_{spec1})$;

iii $\mathfrak{B}_{sup2}$ solve $(\mathfrak{B}_{p2}[\mathfrak{B}'_{sup1}], \mathfrak{B}_{spec2})$.

If both $\mathfrak{B}_{sup1}$ and $\mathfrak{B}_{sup2}$ are non-trivial and complete, and if $\mathfrak{B}_{p1}[\mathfrak{B}_{sup2}]$ and $\mathfrak{B}_{p2}[\mathfrak{B}_{sup1}]$ are complete, then $\mathfrak{B}_{sup1} \times_\pi \mathfrak{B}_{sup2}$ solves $(\mathfrak{B}_p, \mathfrak{B}_{spec1}, \mathfrak{B}_{spec2})$.

Proof. Clearly, $\mathfrak{B}_{spec2} \subseteq \mathfrak{B}'_{spec2}$. Hence, $\mathfrak{B}_{sup2}$ solves $(\mathfrak{B}_{p2}[\mathfrak{B}_{sup1}], \mathfrak{B}'_{spec2})$. Clearly $\mathfrak{B}_{sup2} \subseteq \mathfrak{B}'_{spec2}$ and properties allow the writing $\mathfrak{B}_{p1}[\mathfrak{B}_{sup2}] \subseteq \mathfrak{B}_{p1}[\mathfrak{B}'_{sup2}]$ where $\mathfrak{B}_{sup1}$ solves $(\mathfrak{B}_{p1}[\mathfrak{B}_{sup2}], \mathfrak{B}_{spec1})$. Now pick any overall closed-loop trajectories $w = (w_1, w_2)_\pi) \in \mathfrak{B}_p \cap (\mathfrak{B}_{sup1} \times_\pi \mathfrak{B}_{sup2})$ and observe $w_1 \in \mathfrak{B}_{p1}[\mathfrak{B}_{sup2}] \cap \mathfrak{B}_{sup1} \subseteq \mathfrak{B}_{spec1}$ and $w_2 \in \mathfrak{B}_{p2}[\mathfrak{B}_{sup1}] \cap \mathfrak{B}_{sup2} \subseteq \mathfrak{B}_{spec2}$. Thus, $w \in \mathfrak{B}_{spec1} \times_\pi \mathfrak{B}_{spec2}$. Since completeness is retained under the cross product, we obtain from Proposition 2.3.1 admissibility of $\mathfrak{B}_{sup}$ with respect to $\mathfrak{B}_p$ □

Chapter 3

The hierarchical approach

Many *hybrid control problems* of practical interest can be decomposed in a hierarchy of control structure in order to easily design the whole control architecture. We refer to a structure decomposable in layers with their own time scale, input and output signals. Planned solutions are often "*ad hoc*" and made to avoid computational limitations of known methods for systematic hybrid systems design. The aim of this chapter is to explain some theory details about the hierarchical approach and to apply supervisory a control structure to solve the rendezvous problem. Below there are shown the main principles that form the hierarchical theory showing that hierarchical decompositions is provided by engineering intuition (Sec. 3.1) and that *decomposition* still permits an *overall solution* to the supervisory problem (Sec. 3.2). Results are applied to the rendezvous problem through *abstractions*: the first one of the continuous plant (Sec. 3.3) and a further second one of agents system (Sec. 3.4).

3.1 Basic theory

Real plants are mostly continuous systems where their continuous dynamics might be easily controlled by discrete event supervisors. A system created in this way is called *hybrid automata*. However, behavioural point of view is different and we talk over *hybrid plant*. There are many advantages of this approach, for example there are not any real-valued vector that typically evolves into a huge state space.
A real plant needs an *abstraction* to create its *external behaviour*, defined as the set $\mathfrak{B}_p$ of all sequences of pairs of input and output events that are compatible with the hybrid plant dynamics. Despite the abstraction operation creates approximations of the system it is possible to solve the original control problem through the *plant abstraction* $\mathfrak{B}_{ca} \supseteq \mathfrak{B}_p$. This is not a limit: having a larger set of possible trajectories does not mean that the real system evolves as them actually.
Behaviours are defined as Def. 2.3 through Willems' behaviour theory for the class of problem solvable by discrete events automaton [26]. The subclass of involved systems is described by *I/- behaviours* in Def. 2.2.2. The *closed-loop* behaviour is defined by $\mathfrak{B}_{cl} := \mathfrak{B}_p \cap \mathfrak{B}_{sup}$ and $\mathfrak{B}_{sup}$ is said to *enforce the specification* if $\mathfrak{B}_{cl} \subseteq \mathfrak{B}_{spec}$. Interconnections between continuous plant and discrete supervisor still follow the *admissibility criterion* in Def. 2.3.1 and 2.3.1; this leads to the *supervisory control problem* $(\mathfrak{B}_{\mathfrak{p}}, \mathfrak{B}_{\mathfrak{spec}})_{cp}$ in Def. 2.20.

In [17] it is shown by the following proposition that an admissibility supervisor is independent of the particular plant dynamics and provides that all involved behaviours are complete (Def. 2.2.2) by the following proposition.

Proposition 3.1. *Let $\mathfrak{B}_p \subseteq W^{\mathbb{N}_0}$ be a complete I/- behaviour and $\mathfrak{B}_{sup} \subseteq W^{\mathbb{N}_0}$ be complete and generically implementable. Then $\mathfrak{B}_p$ and $\mathfrak{B}_{sup}$ are non-conflicting.*

If only complete behaviours are considered it will be obtain the main result of [17] for abstraction-based supervisory control. The ensuring theorem represents the resulting deduction.

Theorem 3. Let $\mathfrak{B}_{ca} \subseteq W^{\mathbb{N}_0}$, $W = \mathcal{U} \times \mathcal{Y}$, be an abstraction of an I/-behaviour $\mathfrak{B}_p \subseteq W^{\mathbb{N}_0}$, et $\mathfrak{B}_{spec} \subseteq W^{\mathbb{N}_0}$ be a solution to the supervisory control problem $(\mathfrak{B}_{ca}, \mathfrak{B}_{spec})_{cp}$. If $\mathfrak{B}_p$ and $\mathfrak{B}_{sup}$ are complete then $\mathfrak{B}_{sup}$ is a solution of $(\mathfrak{B}_{ca}, \mathfrak{B}_{spec})_{cp}$.

Let a real plant $\mathfrak{B}_p^L$ be considered over the signal space $W_L = U_L \times Y_L$. It represents the *low level* of the future whole system and U_L, Y_L are input and output of that level.

The control objective can be represented by the set $\mathfrak{B}^L_{spec} \subseteq W_L^{\mathbb{M}_0}$ of all signals that correspond to the well-accepted system behaviour. However, a reasonably accurate *plant model* may be based only on the real mechanic of the low level. Only some *engineering intuitions* could select right measurement to include in the model. Once considering a particular class of problems it is possible to define a low level interface in order to plan an *upper layer*. In this second design step it is possible to realize the chosen supervisory policy.

The relationship between low-level and high-level is graphically shown in

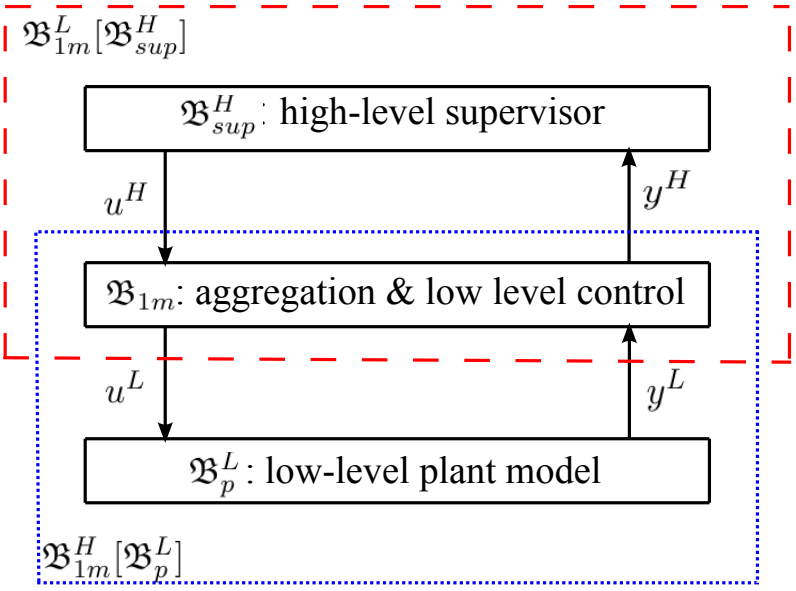


Figure 3.1: Plant (supervisor) perspective, dashed (dotted)

Fig 3.1. The *high-level supervisor* operates on a high level signal space $W_H = U_H \times Y_H$, where each signal can interact with the plant through the interface. This middle-layer is formally represented by a behaviour $\mathfrak{B}_{lm}$ over $W_H \times W_L = U_H \times Y_H \times U_L \times Y_L$.

3.2 Supervisory approach

From the perspective of the low-level plant $\mathfrak{B}_p^L$, the interconnection of the middle-level $\mathfrak{B}_{lm}$ with the supervisor $\mathfrak{B}_{sup}^H$ is seen as an huge unique supervisor $\mathfrak{B}_{im}^L[\mathfrak{B}_{sup}^H]$ over W_L where

$$\mathfrak{B}_{im}^L[\mathfrak{B}_{sup}^H] := \{w^L \mid (\exists w^H \in \mathfrak{B}_{sup}^H)[(w^H, w^L) \in \mathfrak{B}_{lm}]\}.$$

The problem must be now solvable by $\mathfrak{B}_{im}^L[\mathfrak{B}_{sup}^H]$ and in particular enforces the specification:

$$\mathfrak{B}_p^L \cap \mathfrak{B}_{im}^L[\mathfrak{B}_{sup}^H] \subseteq \mathfrak{B}_{spec}^L$$

Even the supervisor has a different perspective because of the interconnecting $\mathfrak{B}_{lm}$ with $\mathfrak{B}_p^L$:

$$\mathfrak{B}_{im}^h[\mathfrak{B}_{sup}^L] := \{w^L \mid (\exists w^L \in \mathfrak{B}_{sup}^L)[(w^H, w^L) \in \mathfrak{B}_{lm}]\}.$$

Although high-level specification are now not imposed, it is needed an *admissibility criteria* for systems interconnection: $\mathfrak{B}_{sup}^H$ must be generically implementable and both $\mathfrak{B}_{lm}^H[\mathfrak{B}_p^L]$ and $\mathfrak{B}_{sup}^H$ must be non-conflicting. Is is possible to define formally from [17]:

Definition 3.1. The pair $(\mathfrak{B}_{lm}, \mathfrak{B}_{sup}^H)_{tl}$ is a two-level *hierarchical solution* to the supervisory control problem $(\mathfrak{B}_p^L, \mathfrak{B}_{spec}^L)_{cp}$ if:

(i) $\mathfrak{B}_p^L \cap \mathfrak{B}_{lm}^L[\mathfrak{B}_{sup}^H]$, and

(iia) $\mathfrak{B}_{lm}^L[\mathfrak{B}_{sup}^H]$ is admissible to $\mathfrak{B}_p^L$, and

(iib) $\mathfrak{B}_{sup}^H$ is admissible to $\mathfrak{B}_{lm}^H[\mathfrak{B}_p^L]$.

The next step is to apply engineer intuition and to plan the *intended* relationship between high-level signals and low-level signals, the result is $\mathfrak{B}_{spec}^{HL} \subseteq (W_H \times W_L)^{\mathbb{N}_0}$. The middle-layer task is to transform $\mathfrak{B}_{spec}^{HL}$ into a understandable signal for the plant so $\mathfrak{B}_{lm}$ must enforce $\mathfrak{B}_{spec}^{HL}$ by the following inclusion:

$$\{(w^H, w^L) \in \mathfrak{B}^{lm} \mid w^L \in \mathfrak{B}_p^L\} \subseteq \mathfrak{B}_{spec}^{HL} \tag{3.1}$$

Let $\mathfrak{B}_{lm}$ be suppose in according to Eq. 3.1, the second step is to design the high level behaviour $\mathfrak{B}^H$ that represents the supervisor. Obviously the new abstraction $\tilde{\mathfrak{B}}_p^H$ is modelled from the *high-level plant* $\mathfrak{B}_{lm}^H[\mathfrak{B}_p^L]$ and the high-level specifications $\tilde{\mathfrak{B}}_{spec}^H$ expresses low-level specifications $\mathfrak{B}_{spec}^L$ in term of high-level signals. Both $\tilde{\mathfrak{B}}_p^H$ and $\tilde{\mathfrak{B}}_{spec}^H$ are obtained from Eq. 3.1:

$$\tilde{\mathfrak{B}}_p^H := \{w^H \mid (\exists w^L)[(w^H, w^L) \in \mathfrak{B}_{spec}^{HL}]\}; \tag{3.2}$$

$$\tilde{\mathfrak{B}}^H_{spec} := \{w^H \mid (\forall w^L)[(w^H, w^L) \in \mathfrak{B}^{HL}_{spec} \Rightarrow w^L \in \mathfrak{B}^L_{spec}]\} \tag{3.3}$$

Observe that the control problem $(\tilde{\mathfrak{B}}^H_p, \tilde{\mathfrak{B}}^H_{spec})_{cp}$ does not depend on the actual low-level plant under low-level control $\mathfrak{B}^H_{lm}[\mathfrak{B}^L_p]$, but only on the intended outcome $\mathfrak{B}^{HL}_{spec}$ of the preceding low-level design.
Examples to how easily solve the control problem efficiently are [22] and [19] through automaton but "ad hoc" solution must be still guided by engineering intuitions. However, it is still needed to proof the Def. 3.1. Concerning Def. 3.1.i it is possible to say:

Proposition 3.2. *Any solution* $\mathfrak{B}^H_{sup}$ *of* $(\tilde{\mathfrak{B}}^H_{sup}, \tilde{\mathfrak{B}}^H_{spec})_{cp}$ *satisfies* $\mathfrak{B}^L_p \cap \mathfrak{B}^L_{lm}[\mathfrak{B}^H_{sup}]$.

Supervisory approach shows an innate problem to discuss before to start designing: although the supervisory system is planned by continuous methods, it is technically realised by digital hardware. Even though it has been used a reasonably high sampling rate, it is necessary to coordinate supervisor and plant allowing each other for the inserted discretization.
A first possible setting consists of using an uniform time scale on all signals,

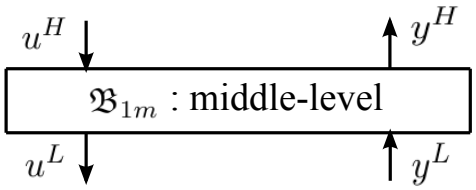


Figure 3.2: Middle Level I/O)

so a synchronous system is implemented. Thus, the middle layer $\mathfrak{B}_{lm}$ has an I/- behaviour, where informations get through vertically at the same time from supervisor to plant and vice versa. Natural candidates to be inputs are u^H and y^L in Fig. 3.2; y^H and u^L are outputs conversely. The supervisory system has to be composed only on I/- behaviours and the completeness property must be inherited by behaviours with the middle-level. Besides, a real problem is a finite set of signals and this peculiarity of the real solutions must not represent a problem. Completeness property is inherited as the following lemma says.

Lemma 3.1. *If* $\mathfrak{B}_{lm}$ *is a complete strict I/- behaviour with respect to* $(W_H \times Y_L, Y_H \times U_L)$*, and if* $\mathfrak{B}^L_p$ *is a complete I/- behaviour with respect to* (U_L, Y_L)*, then* $\mathfrak{B}^H_{lm}[\mathfrak{B}^L_p]$ *is a complete I/-behaviour with respect to* (U_H, Y_H)

The same criteria preserves the generic implementability of both $\mathfrak{B}^L_p, \mathfrak{B}^H_{lm}[\mathfrak{B}^L_p]$ and it is reported in the lemma 3.2.

Lemma 3.2. *If $\mathfrak{B}_{lm}$ is complete strict I/- behaviour with respect to $(W_H \times Y_L, Y_H \times U_L)$, and $\mathfrak{B}^H_{sup}$ is complete and generically implementable, then $\mathfrak{B}^L_{lm}[\mathfrak{B}^H_{sup}]$ is complete and generically implementable.*

It is possible now to admit that under the hypothesis of lemmas 3.1 and 3.2 and considering together with Prop. 3.1 that the admissibility criteria (iia) and (iib) in Def. 3.1, are satisfied. Thus, by 3.2 the pair $(\mathfrak{B}_{lm}, \mathfrak{B}^H_{sup})_{tl}$ is the *two level hierarchical solution* of $(\mathfrak{B}^L_p, \mathfrak{B}^L_{spec})_{cp}$.
The multi-agent scenario is hardly ever set up by agents which are driven by a common clock so it is necessary now to develop an internal structure for $\mathfrak{B}_{lm}$ that implements the non-synchronism. Mapping-functions are still *causal* in referring to different time scales for cause and effect, as [27] explains, if the operator is described as following.

Definition 3.2. Let $F : U^{\mathbb{N}_0} \to Y^{\mathbb{N}_0}$ and $T : U^{\mathbb{N}_0} \to \mathbb{N}_0^{\mathbb{N}_0}$. The operator is said *casual* if

$$\tilde{u} \mid_{[0,k]} = \hat{U} \mid_{[0,k]} \Rightarrow F(\tilde{u}) \mid_{[0,k]} = F(\hat{u}) \mid_{[0,k]} \tag{3.4}$$

for all $k \in \mathbb{N}_0, \tilde{u}, \hat{u} \in \mathbb{N}_0$. The operator T is said to be a *dynamic time scale* if T is strictly casual and if the *time transformation* $T(u) : \mathbb{N}_0 \to \mathfrak{N}_0$ is surjective [1] and monotone increasing for all $u \in U^{\mathbb{N}_0}$. The operator F is said to be *casual* w.r.t. T if T is a dynamic time scale and if

$$\tilde{u} \mid_{[0,j]} = \hat{u} \mid_{[0,j]} \Rightarrow F(\tilde{u}) \mid_{[0,k]} = F(\hat{u}) \mid_{[0,k]} \tag{3.5}$$

for all $k = T(\tilde{u})(j)$ and all $j \in \mathbb{N}_0, \tilde{u}, \hat{u} \in U^{\mathbb{N}_0}$.

Definitions above ensure that at any instant of time the transformation $T(u)$ only depends on the strict past of u. In the hierarchical structure, the middle-level has the input y_L and drives it with the time transformation $T(y^L)$. What is more the high-level measurement is generated by $y^H = F(y^L)$ where F is causal w.r.r. T. The easily way to take advantages of equations 3.4 and 3.5 is through an automaton that generates high-level events whenever the low-level measurement equals a given value or completes a given cycle.

In summary, the middle-layer has a behaviour constructed from the time scale T and the operator F:

$$\mathfrak{B}_{lm} := \{(u^H, y^H, u^L, y^L) \mid y^H = F(y^L) \text{ and } u^L = u^H \text{ o } T(y^L)\}. \tag{3.6}$$

Moreover, the behaviour $\mathfrak{B}_{lm}$ turns out to be complete:

[1] when the function f(x) has only a solution y in the codomain

Proposition 3.3. *Given a dynamic time scale* $T : Y_L^{\mathbb{N}_0} \to \mathbb{N}_0^{\mathbb{N}_0}$ *and an operator* $F : Y_L^{\mathbb{N}_0} \to Y_L^{\mathbb{N}_0}$ *that is casual w.r.t.* T*, define* $\mathfrak{B}_{lm}$ *by Eq. 3.6. Then* $\mathfrak{B}_{lm}$ *is complete.*

Analogous to the results to the uniform time scale solution, the 3.3 preserves the input/output structure of the plant and generic implementability of a supervisor. Thus, lemmas 3.1 and 3.2 have an equivalent enunciation in the non-uniform time scale as following.

Lemma 3.3. *Under the hypothesis of Proposition 3.3, and if* $\mathfrak{B}_p^L$ *is a complete I/- behaviour w.r.t.* (U_L, Y_L)*, it follows that* $\mathfrak{B}_{lm}^H[\mathfrak{B}_p^L]$ *is a complete I/- behaviour w.r.t.* (U_H, Y_H)*.*

Lemma 3.4. *Under the hypothesis of Proposition 3.3, and provided that* $\mathfrak{B}_{sup}^H$ *is complete and generically implementable, it follows that* $\mathfrak{B}_{lm}^L[\mathfrak{B}_{sup}^H]$ *is complete and generically implementable.*

Totally in agreement with the synchronous case, lemmas 3.3 and 3.4 show that if the intermediate specification $\mathfrak{B}_{spec}^{HL}$ is implemented through a behaviour $\mathfrak{B}_{lm}$ according to 3.6, the pair $(\mathfrak{B}_{lm}, \mathfrak{B}_{sup}^H)_{tl}$ is a two-level hierarchical solution of $(\mathfrak{B}_p^L, \mathfrak{B}_{spec}^L)_{cp}$.

3.3 Abstraction based synthesis

The practical problem to solve in this work has its theoretical basics in Sec. 2.1. Although it consists of a set of cooperating agents, the hierarchical approach allows to plan a solution structured onto levels where the first one depends on the single agent. Next step is to include interactions between agents through a decentralized approach.
Let the generic *i-th level* have the signal space $W_i = U_i \times Y_i$ that is interconnected with a intermediate layer $\mathfrak{B}_{lm}^{i,i-1}$ over $W_i \times W_{i-1}$ with $1 \leq i \leq m$ and a high-level supervisor $\mathfrak{B}_{sup}^m$. If the plant $\mathfrak{B}_p^0$ is a complete I/- behaviour the middle-level has characteristics discussed in Sec. 3.2 and it is possible to define the behaviour from the plant perspective

$$\mathfrak{B}_{sup}^i := \{w^i \mid \exists w^{i+1} \in \mathfrak{B}_{sup}^{i+1} : (w^{i+1}, w^i) \in \mathfrak{B}_{lm}^{i+1,i}\},$$

and from the generic i-th supervisor

$$\mathfrak{B}_p^i := \{w^i \mid \exists w^{i-1} \in \mathfrak{B}_{sup}^{i-1} : (w^i, w^{i-1}) \in \mathfrak{B}_{lm}^{i,i-1}\}.$$

Climbing the hierarchical structure the information gets more abstracted step-by-step and high levels have totally abstracted state space that could

correspond to logical abstracted situations. However, level by level must be verified implementability and completeness, so it is possible to define the *hierarchical solution* [17] as following.

Proposition 3.4. *The tuple* $(\mathfrak{B}_{lm}^{1,0}, \mathfrak{B}_{lm}^{2,1}), \cdots, \mathfrak{B}_{lm}^{m,m-1}, \mathfrak{B}_{sup}^{m}$ *is said to be an* (m+1)-level hierarchical solution *to the complete problem* $(\mathfrak{B}_{p}^{0}, \mathfrak{B}_{spec}^{0})_{cp}$ *if*

(i) $\mathfrak{B}_{p}^{0} \cap \mathfrak{B}_{sup}^{0} \subseteq \mathfrak{B}_{spec}^{0}$, *and*

(ii) for all i, $0 \leq i \leq m$, $\mathfrak{B}_{sup}^{i}$ *is admissible to* $\mathfrak{B}_{p}^{i}$.

An iterative bottom-up design is proposed in [17], afterwards being decided each layer signal space for n levels, such that for all i, $0 \leq i \leq n$:

(a) $\mathfrak{B}_{p}^{i} \subseteq \tilde{\mathfrak{B}}_{p}^{i}$,

(b) $\{(w^i, w^{i-1}) \in \mathfrak{B}_{lm}^{i,i-1} \mid w^{i-1} \in \tilde{\mathfrak{B}}_{p}^{i}\} \subseteq \mathfrak{B}_{spec}^{i,i-1}$,

(c) $w^i \in \mathfrak{B}_{spec}^{i}$ and $(w^i, w^{i-1}) \in \mathfrak{B}_{spec}^{i,i-1} \Rightarrow w^{i-1} \in \mathfrak{B}_{spec}^{i-1}$

In Fig. 3.3 it is shown the general architecture of [18] proposed as decentralized solution. The coordinating layer has a particular architecture composed by two parts: a *conjunctive* and *disjunctive* one. The "*And*" and "*Or*" boxes implement logic of enable and disable events respectively. The point of view

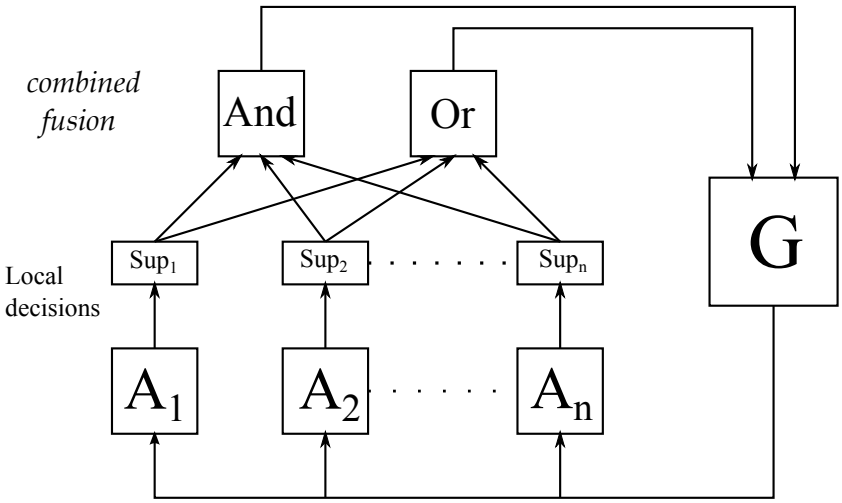


Figure 3.3: Classic Conjunctive-Disjunctive Architecture

of this work is slightly different from this architecture aim, although that kind of structure could preserve I/- behaviour properties and being a supervisory

approach such Def. 2.20. In contrast with [18] the decentralized architecture of a multi-agent system does not allow a unique solution in high level in order to coordinate agents. The whole system is not completely determinate, the number of agents is a requested degree of freedom and the solution must be adaptive to this. A possible solution of this problem is to establish a com-

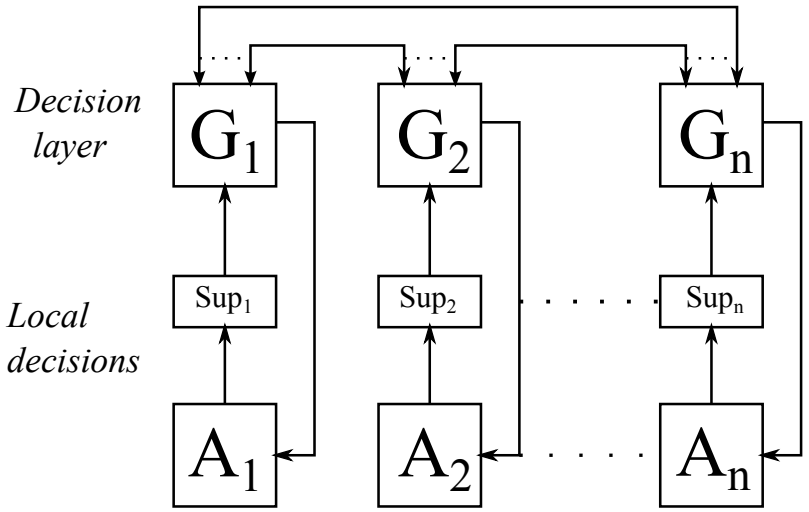


Figure 3.4: Considered Architecture

munication protocol between high level supervisors in order to implement a private second layer for each agent.

Let us consider a generic second level supervisor $\mathfrak{B}^2_{sup} \subseteq \mathfrak{B}^2_{spec}$ that has signal space $W^2 = W^2_{in} \times W^2_{out}$. If it is expected a short communication, inputs are not only from the underneath level but even from other sources. Hence, the input is $W^2_{in} = W^2_{in1} \times W^2_{in2} \times \cdots W^2_{inn}$ and is necessary to find some condition for an uniform communication.
If each high supervisory level $\mathfrak{B}^2_{sup} \subseteq \mathfrak{B}^2_{spec}$ has different W_{in}, a particular case of Def. 2.22 will be set where supervisors could take different decisions even with the same *behavioural policy*. If each supervisor implements a *disjunction policy*, a local set of rules could be set to disable unwanted behavioural evolutions. A right-thinking policy could be set as following.

Proposition 3.5. *Given two second level supervisors which have* $\mathfrak{B}^2_{sup1}, \mathfrak{B}^2_{sup2} \subseteq \mathfrak{B}^2_{spec}$ *and* $W_1 = W_{in1} \times W_{out1}$ *and* $W_2 = W_{in1} \times W_{in2} \times W_{out2}$ *where* $W_{in1} \cap W_{in2} \neq \emptyset$ *then* $\mathfrak{B}^2_{sup2} \subseteq \mathfrak{B}^2_{sup1} \subseteq \mathfrak{B}^2_{spec}$

Prop 3.5 could be easily implemented with as Fig. 3.5, it is also possible:

- to include a new agent run-time if it is projected such Prop. 3.5;

- to satisfy Prop. 3.4 because are still valid $\mathfrak{B}^1_{sup} \cap \mathfrak{B}^2_{sup}$ and $\mathfrak{B}^2_{sup}$ are admissible to $\mathfrak{B}^1_{sup}$ for each agent

- to permit a conservative behaviour if $\mathfrak{B}^2_{sup1} \cap \mathfrak{B}^2_{sup2} \cap \cdots \mathfrak{B}^2_{supn} = \mathfrak{B}^2_{spec}$ for an agent.

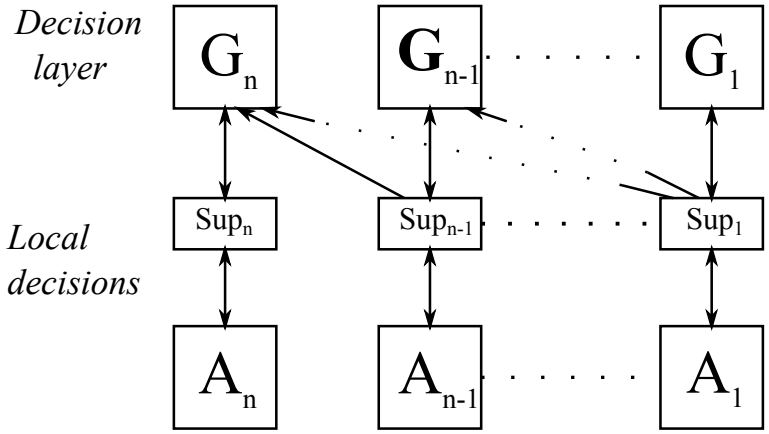


Figure 3.5: Planned Architecture

In Fig. 3.4 it is supposed a communication between high supervisors machine without any limitation of links and directions and links between last layers mean that. If communication is formed only by forwarded signals, even a subset of them is permitted, the Fig. 3.5 is a particular solution where the vertical logical communication is preserved.

3.4 Modular supervisory control of a multi-agent system

Decentralized approach in a hierarchical structure depends on the engineer intuition ad the framework explained before provides guide lines to follow. Examples of the engineer work are [15], [22] and [28] where supervisory approach is planned with and without framework. However, the main step up of this work is creating a second supervisory layer of the framework and examples had not been done yet.

Let us consider a real and likely example of agent with the following

general dynamic:

$$\begin{cases} \dot{x}_1 = v\cos(\theta) \\ \dot{x}_2 = v\sin(\theta) \\ \dot{v} = u_1 \\ \dot{\theta} = u_2 \end{cases} \tag{3.7}$$

Obviously real values could be estimated only using real robots but there are some certain limitations due to the physical implementation:

- $\theta \in [0, 2\pi]$;
- $v \in [0, v_{max}]$;
- $u_1 \in [\dot{v}_{min}, \dot{v}_{max}]$;
- $u_2 \in [\dot{\theta}_{min}, \dot{\theta}_{max}]$.

Therefore, angle and acceleration variations are not instantaneous, angles are periodic and velocity has a maximum value.

In analogy with [17] and [29] agents/robots must be driven in a working field from a *starting position* to a *goal position* avoiding collisions between them.

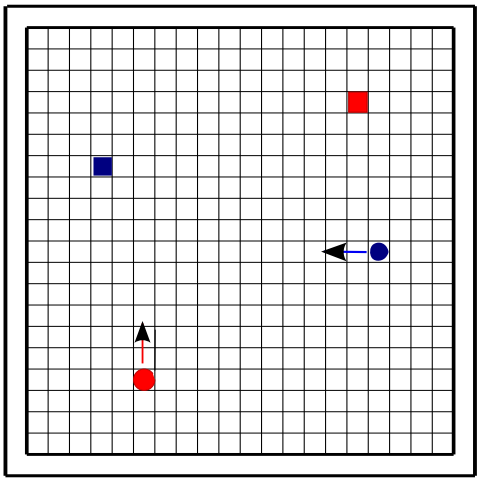


Figure 3.6: Small boxes panorama

A first abstraction of the problem is shown in Fig. 3.6: the field is divided into *small boxes*, each one with a couple of values (x_{sm}, y_{sm}), and agents cross some of them. Hence, a first supervisory layer has to guide the agent trough the field and it is described by a behaviour $\mathfrak{B}^{L1}_{sup} \subseteq \mathfrak{B}^{L1}_{spec}$ where the

specifications are a set of *possible trajectories* that solve the problem. Signal space is composed here as $\mathbf{W}_1 = \mathbf{U} \times \mathbf{Y}$ where $\mathbf{U} = \mathbf{U}_1 \times \mathbf{U}_2$ and $\mathbf{Y} = Y_1 \times Y_2$ are inputs/outputs from/to overhanging and underlying layers. In detail, signals from the upper layer represent which trajectories are permitted any longer and from the layer below arrives the small box where the agent is. In summary, the global panorama of this supervisory layer is composed by:

- a couple of coordinates $\mathbf{U}_1 = \{(x_{SB}^0, y_{SB}^0), \cdots, (x_{SB}^{j-1}, y_{SB}^{j-1})\}$ where j is the number of agents;
- the map of *conflicting big areas* of the field $\mathbf{U}_2 = \{C^{box_0}, \cdots, C^{box_{p-1}}\}$ with p the fixed number of them;
- the outputs toward the plant $Y_1 = \{d_0, \cdots, d_{m-1}\}$ through $m$agents can be driven;
- the outputs toward the second level supervisor $Y_2 = /C^{box_0}, \cdots, C^{box_{p-1}}/$ which indicate big boxes still full.

However, Y_2 and U_2 have the same signal space but it is only a formal way to express what type of shared information is. In a real implementation, signals between the two supervisory levels are translated by a middle layer. From

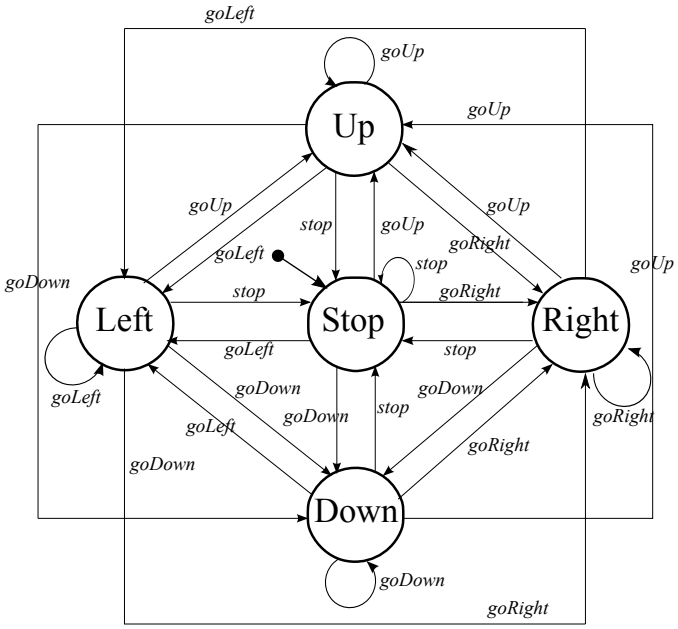


Figure 3.7: First Level Supervisory Automata

the point of view of the single agent the discrete supervisory behaviour is described by [30]

$$\mathbf{G} = (X, E, \delta, x_0, X_m)$$

where:

- X is the state space $\{Stop, Left, Right, Up, Down\}$;
- E is the alphabet $\{stop, goLeft, goRight, goUp, goDown\}$;
- $\delta : X \times E \rightarrow X$ is the *transition function* that describes which state is reached through the event e;

δ	**Stop**	**Left**	**Right**	**Up**	**Down**
Stop	stop	goLeft	goRight	goUp	goDown
Left	stop	goLeft	goRight	goUp	goDown
Right	stop	goLeft	goRight	goUp	goDown
Up	stop	goLeft	goRight	goUp	goDown
Down	stop	goLeft	goRight	goUp	goDown

- $x_0 \in X$ is the initial state, $x_0 = \{Stop\}$;
- $X_m \subseteq X$ are the final states, $X_m = \{Stop\}$.

Events $e \in E$ are input from the upper layer where the coordination policy is implemented and represent what is the next small box to reach.

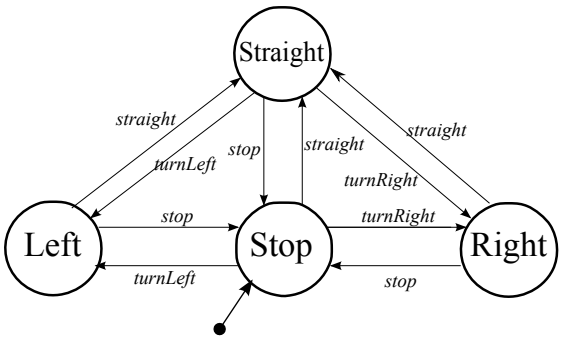


Figure 3.8: Supervisory Translation Policy to the Plant

The Fig. 3.7 describes how a first level supervisor limits the agent movement by a DES automata with a discrete interface where the state represent the output for the agent. However, it is necessary to translate the output to be

understandable to the agent, so the first supervisor commands are thought over a $\mathfrak{B}_{cl}$ that gives abilities to limit the agent movement.

The Fig. 3.8 shows how the agent dynamic is logically limited: each state has the outputs u_1 and u_2 towards the agent and switching values can guide it. In example, given Eq. 3.7, the event *turnLeft* means to create a new angle $\theta' = \theta + \frac{\pi}{2}$ through the output u_2 and the event *stop* means to bring the velocity to 0 through the input u_1. The joining between the dynamical equations and DES movement abstraction makes up the *hybrid closed-loop system* that is supervised by $\mathfrak{B}_{sup1}$.

The main improvement of this framework is introducing a new layer of

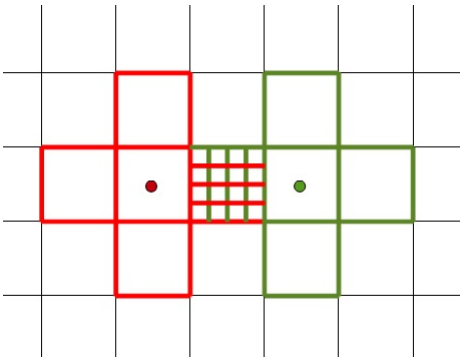


Figure 3.9: Agents on the field

abstraction that represents a collaborative aim. This layer is projected such it was a big areas map of the field that agent could reach. It is important to highlight that in this layer the inputs are the small boxes position of each agent without any information about goals positions. Fig. 3.9 represents the abstract idea of this layer: two agent that share a *next possible big box* must know a-priori who can get into between them. A simple solution is given with a priories hierarchy: each agents can control only *more important* agents signal in order to disable unwanted events. This solution has two interesting characteristics: the communication is not full connected and a new agent could be inserted runtime giving it the lowest priority.

The last point to clarify about the model concerns condition with witch a first level supervisory can decide its possible next small boxes. Fig. 3.10 shows the scenario: the forbidden big boxes map arrives from the upper layer and through the agent and goal positions is possible to planning a movement strategy of this level. In detail, each agent has a free area around it and if it cross a different big box the supervisor will decide the next manoeuvre. The *overall behaviour* $\mathfrak{B}_{spec} := \mathfrak{B}_{spec1} \times_{\pi} \mathfrak{B}_{spec2}$ is ultimately implemented as the

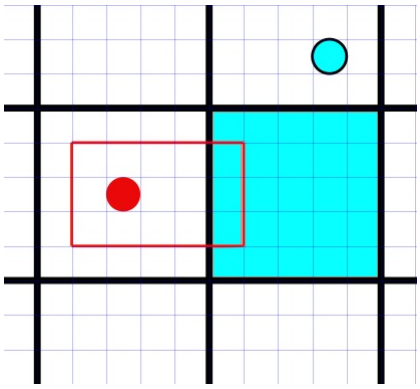


Figure 3.10: Agents on the field

set of possible next manoeuvres limited by second layer supervisory policy.

The Fig. 3.11 is given such summary to underlying that the hierarchical

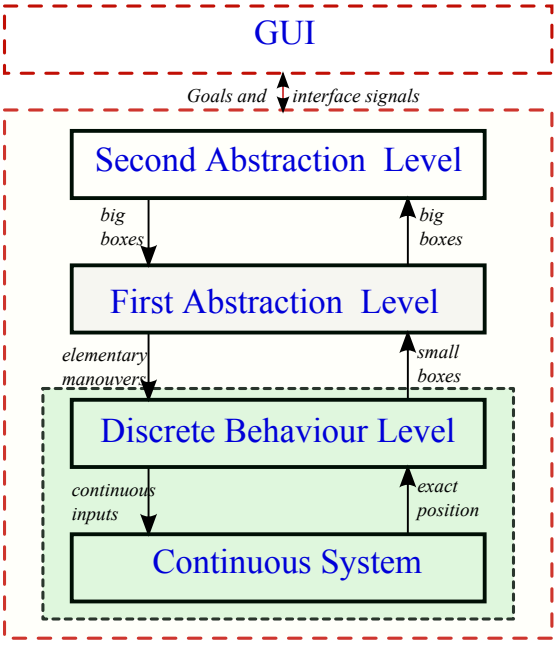


Figure 3.11: Agents on the field

approach permits the solution of the multi-agent system only through abstractions of single agents. The GUI [1] layer is only a way to include possible

[1]Graphic Unit Interface

higher layers, i.e. for planning a series of goals to reach.

Chapter 4

Model

This chapter has the aim to explain the Matlab model that implements the project seen in previous chapter.Following, the *four agents* solution is described, even though the number of them is totally unimportant. However, the most complicated situation is set with 4 robots and more robots do not increase the complexity.

The top view is given in Fig. 4.1, four agents are represented vertically with their own hierarchical structure horizontally. Some functions are repeated in different blocks, the whole structure follows the theoretical approach and each block represents a small part of the reasoning. On the left side is given the continuous dynamical systems presented in Sez. 3.7 with their *middle-layer*, details will be given in 4.1 for each sub-block. On the right side there are high level abstractions and the supervisory policy that are discussed in Sez. 4.2 and 4.3.

Some *display components* are inserted, they do not implement any particular function but are useful for a numerical viewing and they are not enough convenient to analyse the whole project. A GUI was implemented in order to create a plain interface and to observe agents movement during the simulation, a brief description of it will be given at the end of this chapter in Sez. 4.4.

A clarification must be done before to start explaining every single block: every simulation is parametrized with some permanent features and they can not be changed runtime. In particular it is necessary fixed *small boxes step quantization* and *big boxes step quantization*. At the opposite extreme are *goal positions*, to change them means to change a direction that is a common procedure in this work. Moreover, each agent is described by starting conditions: *starting* x and y exact positions and the *starting angle* of the movement.

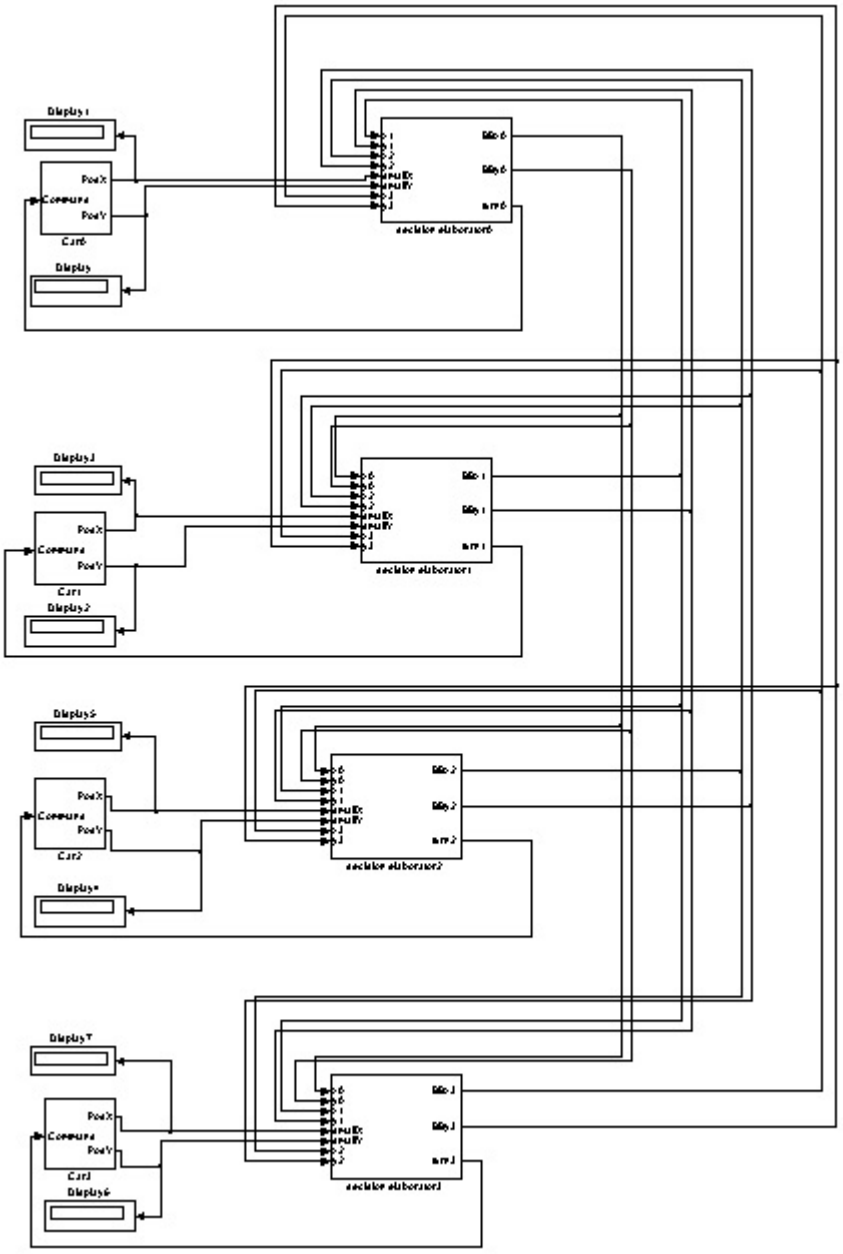


Figure 4.1: Global View

4.1 The physical model and the field

In this section are given some details about the agent dynamic and its limited movement in the model. Let us start from the top-viewing of Fig. 4.2, the box is interfaced with the first supervisory level through input and output ports. The port *Command* is the input from the supervisor that indicates where the *next small box* is, that is a numerical value that means one of $\{up, down, left, right\}$. The ports $PosX$ and $PosY$ are outputs, still to the first level supervisor, that indicate in which small box is the agent in terms of coordinate. Therefore, outputs has already changed from the exact point on the plain to the small box indication.

A more detailed viewing is given in Fig. 4.3. The continuous dynamic is

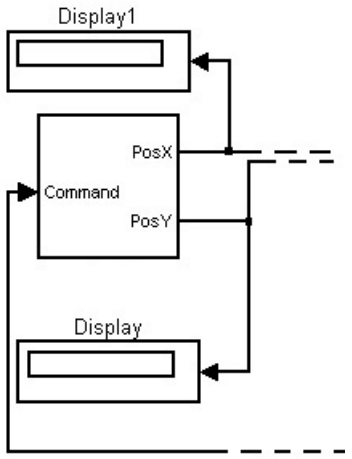


Figure 4.2: Agent Viewing

contained in the box called "*Agent*". Outputs of this box are x and y exact positions, the *velocity* and the *exact angle* θ; all are values correlated to a common point called *origin*. Inputs of the agent are the commands u_1 and u_2 from the *middle-layer* already presented in the last two equations of the system 3.7 that modify the velocity and the angle. The complete dynamic is fully built in Fig. 4.4, one can here notice that the initial angle θ in input of *Sine* and *Cos* blocks.

The middle layers is implemented in the block *direction changer* of Fig. 4.3 through the Matlab tool *State Flow* [7] [8], it receives *velocity* and the

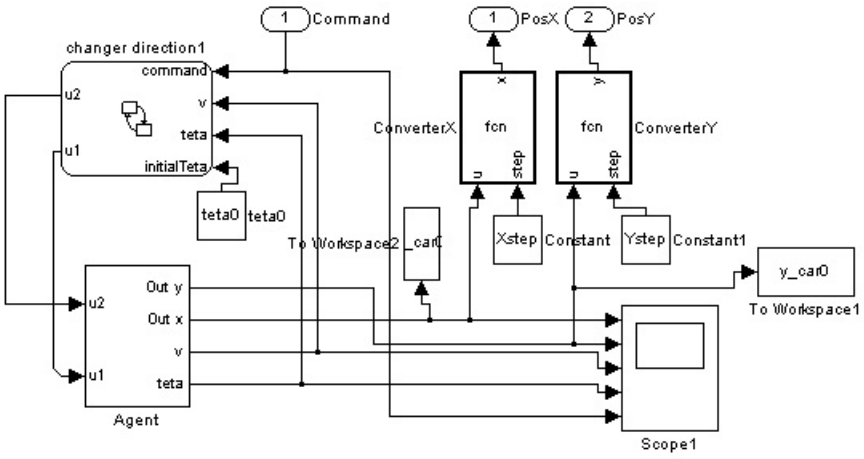


Figure 4.3: Agent Box Viewing

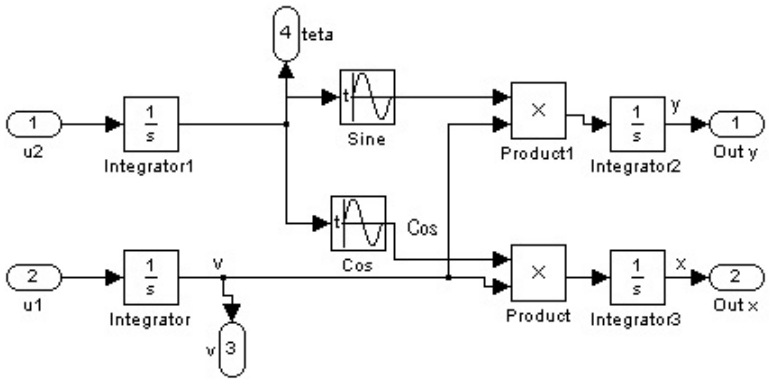


Figure 4.4: Agent Continuous Dynamical Model

angle from the agent and the command to translate from the first supervisor. Outputs are u_1 and u_2. Middle-layers functionalities are not simple and need to be discussed carefully. The general schema is given in Fig. 4.5, the model is planned with starting idle agents and this layer drives an agent to satisfy requested manoeuvres.

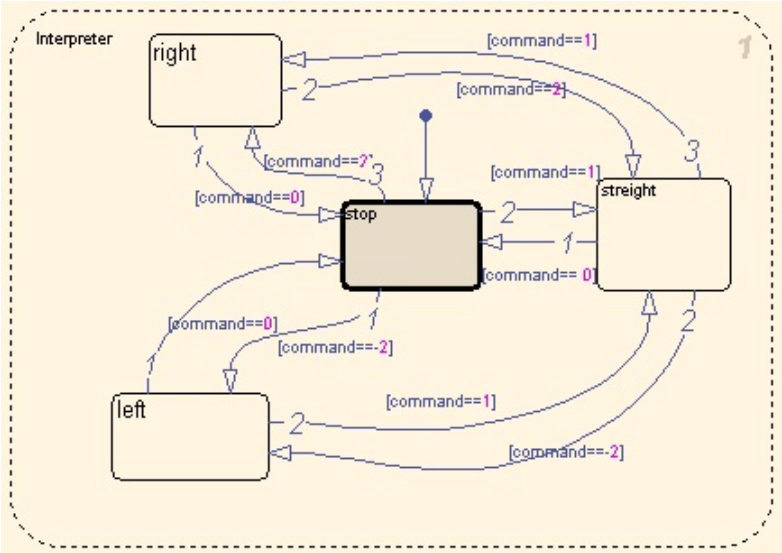


Figure 4.5: Middle-Layer General Schema

Some small complications are included in this layer:

- events are not clocking;
- the starting angle θ may be every value;
- the reversing movement must be implemented.

The first problem will entail not complete dynamical evolutions if commands switching are fast, so each state must recognize when the evolution is ended and when in being done. The second problem is avoided partially giving only a small set of possible starting angle, they will be modified only with $\pm\frac{\pi}{2}$. However, this limitation has highlighted come computational problems of *Matlab* with numerical operations, especially with angles. The third one is cleverly solved through a series of movements in this middle layer that are two turning to the same direction. Despite the complications, this level is logically described with states *stop*, *right*, *left* and *straight*.

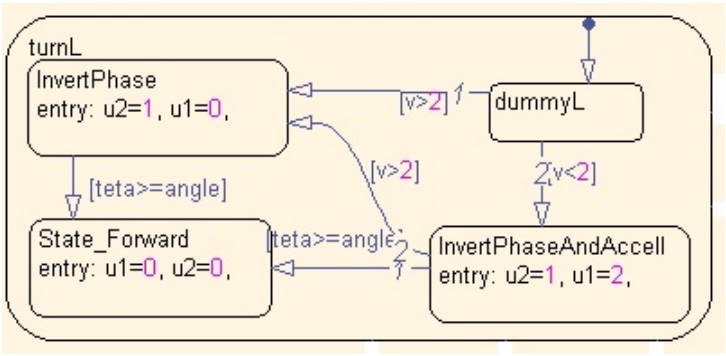


Figure 4.6: Turn Left Schema

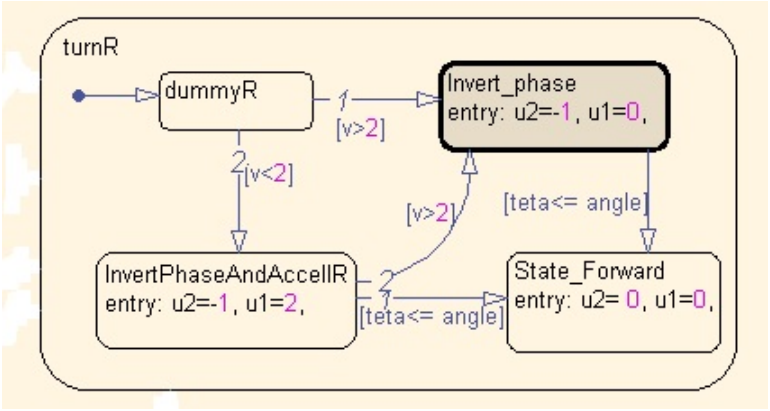


Figure 4.7: Turn Right Schema

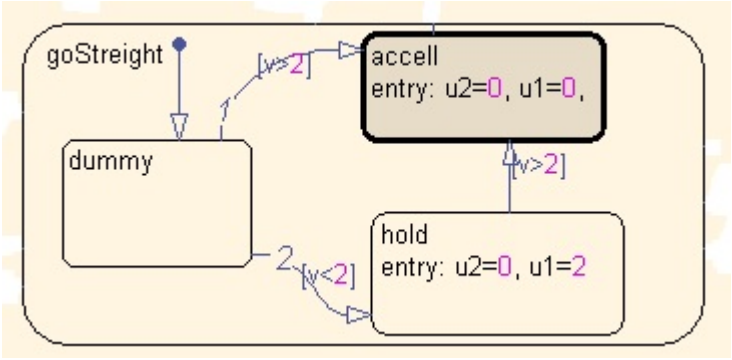


Figure 4.8: Go Straight Schema

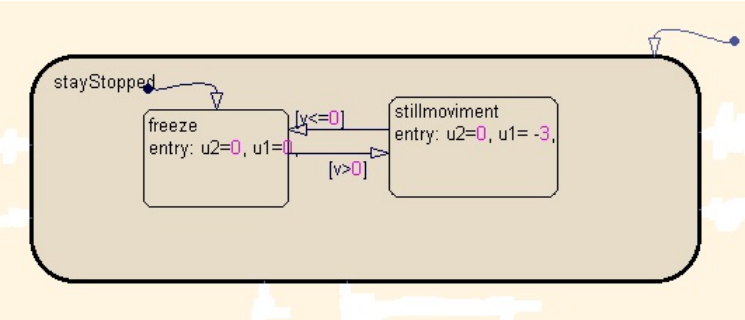


Figure 4.9: Stop Schema

Figures 4.6, 4.7, 4.8 and 4.9 represent all sub-procedure for each state. In details *turn left* and *turn right* are built with the same logic but with opposite values. The stating procedure is called *Dummy* because has no output and switch immediately trough the input velocity. The two different velocity mean if the system had already reached the maximum speed on not. Angle are calculated by the schema in Fig. 4.10 because of the better State Flow synchronism. The *go straight procedure* start with the same logic of

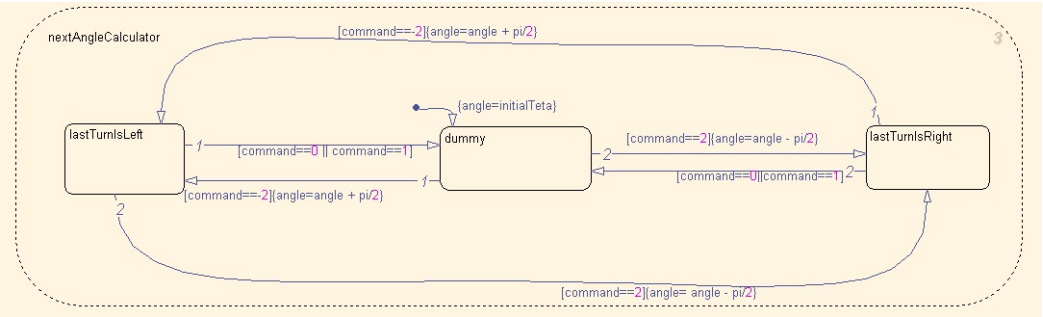


Figure 4.10: Angle Calculator

blocks before and control if an angle movement have been interrupted. If the angle movement is interrupted it will complete it. The last two blocks to explain in Fig 4.2 are *ConverterX* and *ConverterY*. They do simply the small boxes quantization of the field through the *floor* of a division.

4.2 First layer supervisor

The aim of this section is to give the first level supervisor panorama already explained in Cap. 4 as theory. The Fig. 4.11 contains both first and second supervisory functions, although the biggest part forms only the first supervisory policy.
The purposes of this level are:

- to generate the best next small boxes;
- to understand when the work is over;
- to translate all the reasoning into command for the underlying level;
- to forward position informations to the higher level.

Obviously all purposes can be planned in different ways. Let us consider the best next small boxes studying, there are not any restraints, so any policy could be implemented. The policy could be implemented differently for each agent. In this system, choosing the next small box means to compute away of thinking by two steps: decide a set of possible actions ordered by benefits and after decide one of them, according to commands from the second supervisory level. These two blocks are shown in Fig. 4.12, "*Priority*" orders possible actions and "*Best action*" chooses the small boxes. The implemented policy of best choose is simply based on the *Manhattan distance* in this example and it do not represent a very important topic in this model. The real movement is chosen in the block *Best action* through the method described in Fig. 3.9, close big boxes are valued before to avoid to enter in to. For each direction there are parameters to calculate as the Fig. 4.13 shows, all parameters are compared with the the other and a final decision is taken.

Fortunately both operations, understanding when the the goal is reached and forwarding position informations to the higher level, are easy operations. The first one is done through comparing the actual position with the goal one. The second one is implemented only such a translation by a *floor* of a division.

The last purpose is to analyse the translating the decision into command for the underlying level. This operation is done by the State Flow block "*Interpreter*". Its goal is to permit movements on the plain using the only understandable commands for the plant interface. The logic is shown in Fig. 4.14, the idea is analogue to Fig. 4.5 but now we have four directions and reverse movements must be planned.

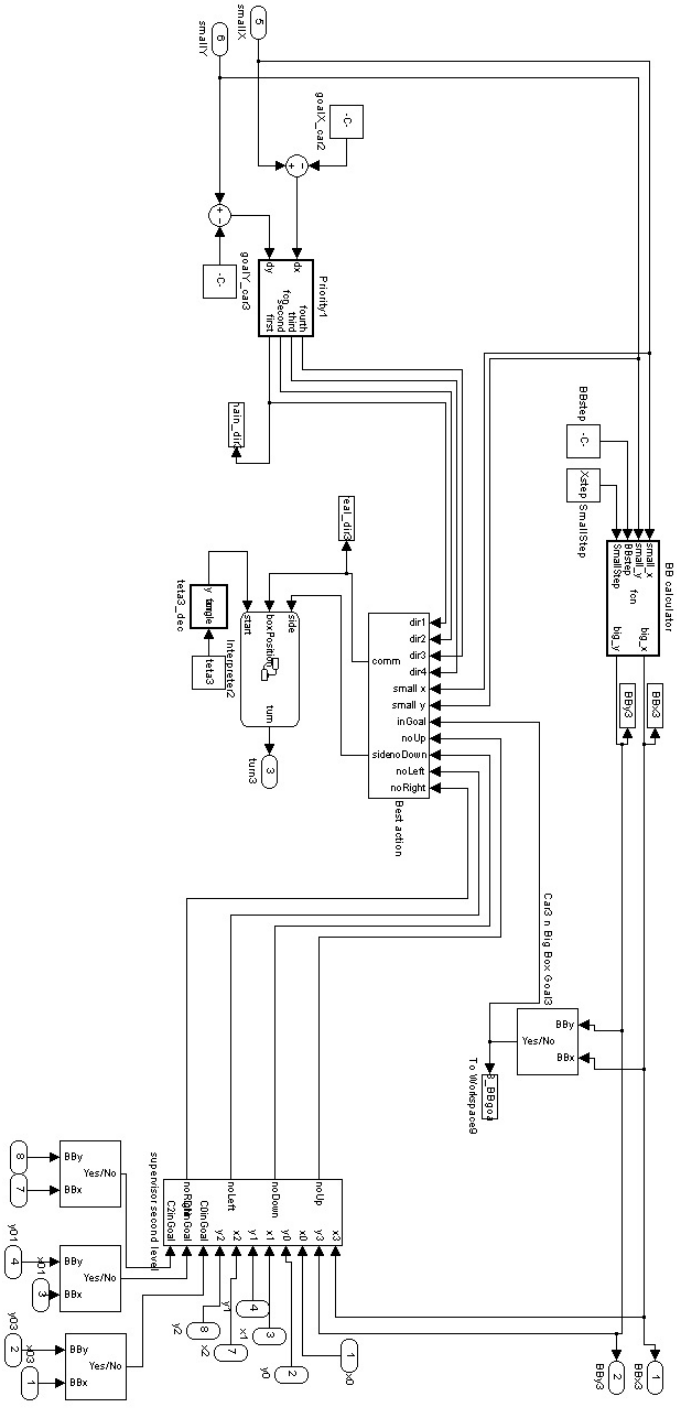


Figure 4.11: Supervisory System Top Viewing

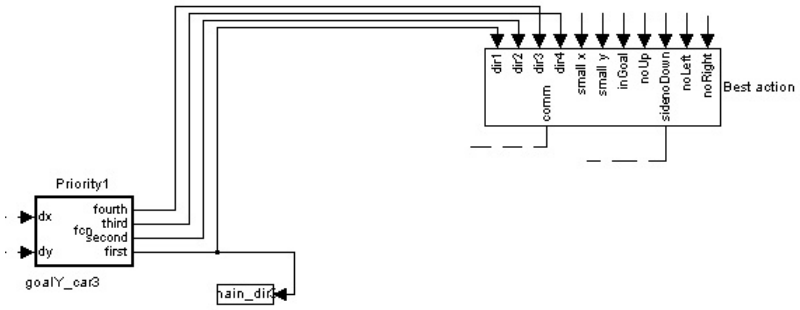


Figure 4.12: Direction Decision System

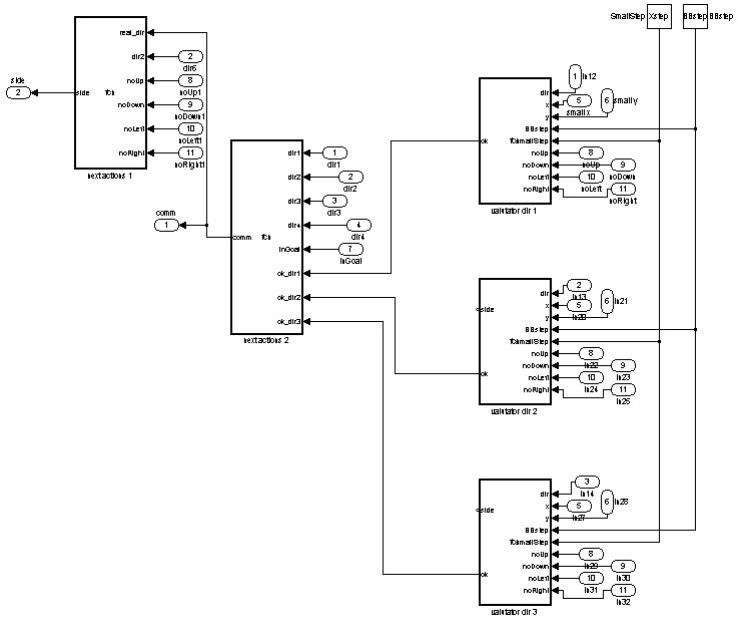


Figure 4.13: Inside the Best Action Block

In this layer an important point of uncertainty is the starting point: angles and exact positions can be every value and the first manoeuvre is always *constrained* to starting conditions. The adopted solution is still shown in Fig. 4.14 through the state *Init*. Moreover, changing small box means to give a set of commands to the plant, generated on each transition, so the all states called *Wait* implement this mechanism. A special focus on transitions

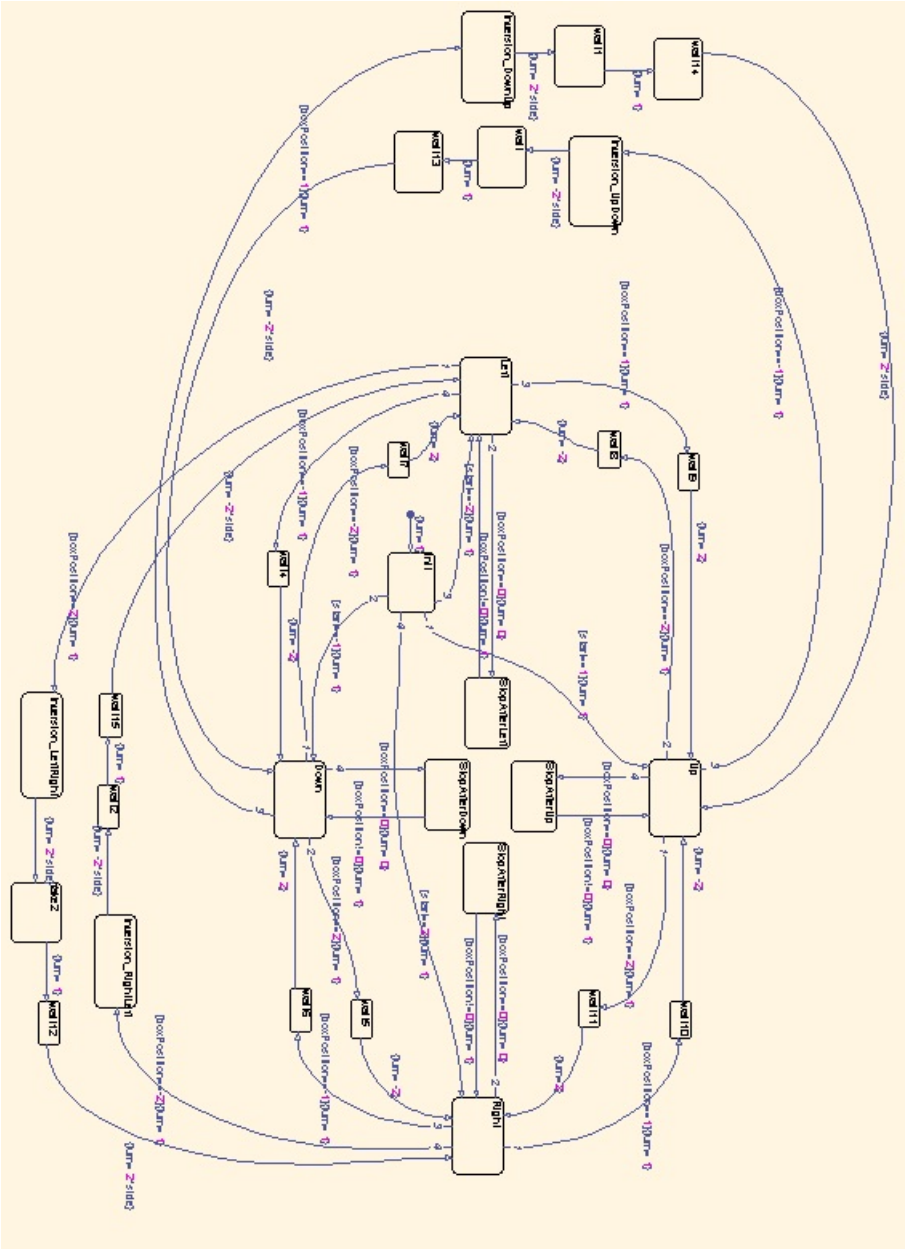


Figure 4.14: Translator of the first supervisor

is now given as last observation: the logical state *Stop* is divided into 4 parts because it necessary to remember what the previous command was when a movement is interrupted but not completed. The typical example of this necessity is an agent that stops its movement because another one, with

higher priority should use its next big box.

4.3 Decentralized supervisor

The last part of the model is the *second supervisory level* where the multi-agents policy is implemented. In contrast to previously sections each agent has a private implementation different to the others. The bock called *supervisor second level*, zoomed Fig. 4.15, implements the policy. The planned

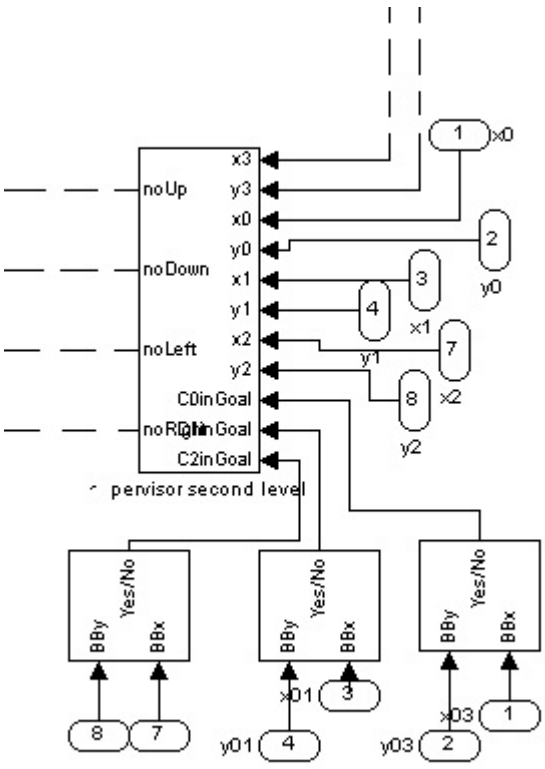


Figure 4.15: Second Supervisory Level

logic is limiting behavioural evolutions of underlying levels, so outputs represent disabled directions. Movements are studied through the field division on *big boxes*, it is known the *big box position* for each agent and outputs are based only on this.

Let us consider the particular implementation for this model: each agent has a priority and processes the informations differently if are from a higher o lower priority structure. The system is thought in order to allow a later

introduction of an agent, so the priority level is unique for each agent. Each second level supervisor disable a possible trajectory if they have a big box in common at least. The Fig. 4.16 shown easily what big boxes must be

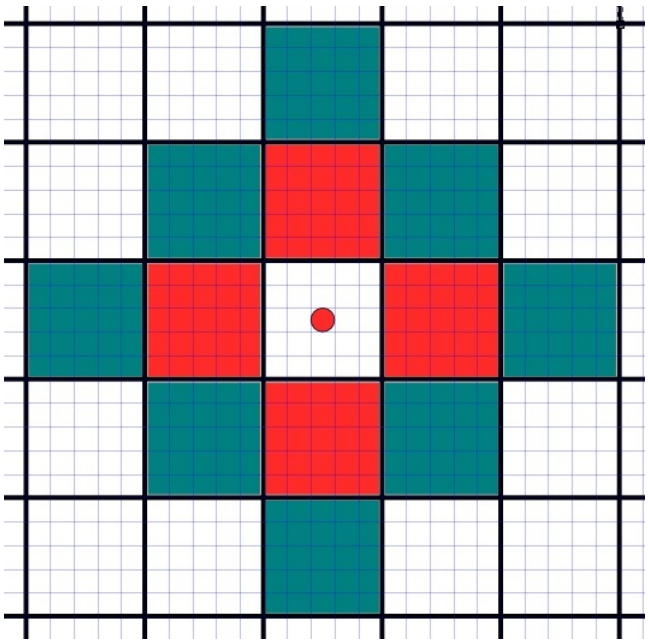


Figure 4.16: Different Big Boxes

analysed. The red ones represent some boxes conditions where agent are placed side by side, in this case it is not important if the movement is still in that direction and the agent with lower priority must move immediately according to this. If two agents has a possible red box in common, it means not placed side by side yet, the manoeuvre is still analysed function of the actual movement. On the other hand, blue boxes are the first not important boxes around an agent.

4.4 Graphic Unit Interface

The complete model is set with discrete and continuous dynamics insomuch as it is tricky studying the complete evolution only through display and scope components. A graphic interface has been planned by Matlab functions [9], it is shown in Fig. 4.17 the version about four agents.

The interface allows to interact with the system, two different logical part are included: the first one allows to insert initial informations and the second

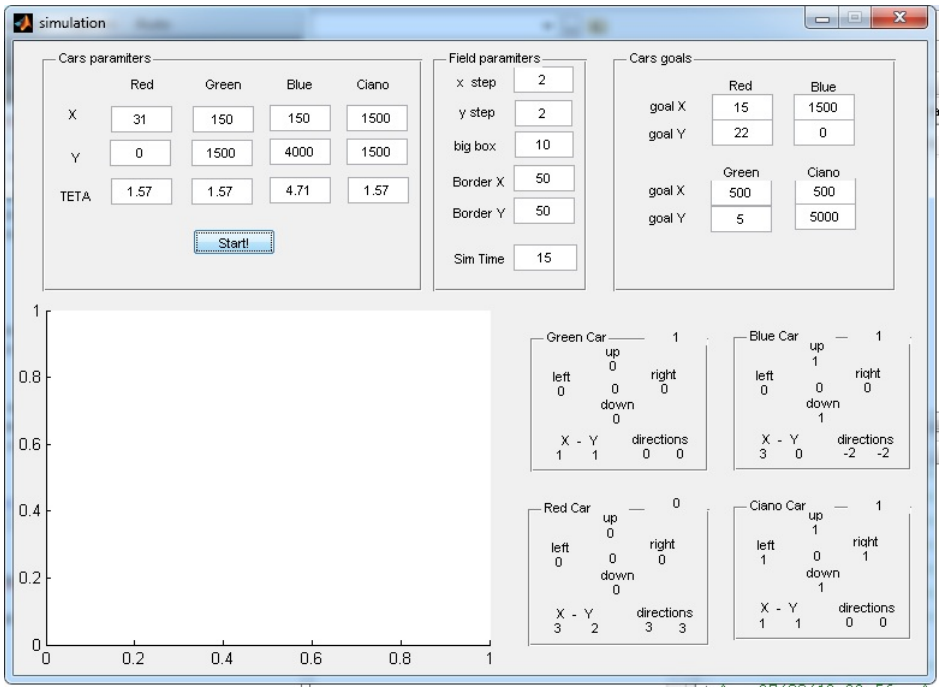


Figure 4.17: Graphics Interface

one allows to analyse the movement.
Earlier than simulating, it is necessary to insert the starting values for each agent, some information of the field and the goal positions. Obviously agents informations are x/y starting point and initial θ whereas field parameters are field dimensions ($Border\ X/Y$), small boxes step quantization ($x/y\ step$) and the big boxes quantization.
The visualization part is divided into two parts: the first one is the box on the lower left side and shows agents movements on the discrete field, the second one shows decisions of supervisory approach. A brief panorama of the movement is given with this second part, it contains which the *main* and *real* movements are in order to understand when the supervisory policy is applied for an agent. Besides, labels *left*, *right*, *up* and *down* represent how the instant picture is around each agent. The central label *here* marks when the actual big box might be used by other agents and if it is possible it will move out.
This system is only an easier instrument to study the system. However, no runtime actions are possible through it. When the simulation starts it will not be possible to modify it and the graphic interface will show results reading results on the workspace.

Chapter 5

Tests and results

The purpose of this chapter is to give practical examples about the model behaviour. Many test could be done because the number of possible situations is huge but following the most representative ones are shown. In each section of this chapter is provided picture of a particular situation in order to show some details of the architecture behaviour. The sections order is thought because of a simple viewing. The first section shows the most simple movement whereas the last one shows the most complicate situation with four agents that want to cross the same big box.
Conditions of experiments are:

- a big box can contain only an agent;
- a big box can contain only a goal small box;
- the agents movements must start with position and angle that do not permit to cross the field perimeter.

The field is contained in a perimeter modelled as forbidden big boxes out of the viewing. Each example is repeatable and might be complicated as it is desired but, obviously, a precise idea is given only seeing complete movements with the GUI.
Each simulation is described by numbers and labels and they will be highlighted example to example. As brief summary is given the following table, it contains the *first supervisor* possible signal values. The command "*continue*" indicates that the previous command is still valid

stop	up	down	left	right	continue
0	1	-1	-2	2	3

5.1 Start and stop manoeuvres of an agent

The first given example represents the easiest case of an agent that must reach its goal position only going straight.
If the GUI interface is filled with precise parameters that permit the starting alignment between starting and goal positions it will be possible to observe the *goStraight* movement with the *acceleration* and *deceleration* manoeuvres. Let us consider some possible values of the interface in Fig. 4.17:

- the field has dimension $(BorderX, BorderY) = (50, 50)$;
- small boxes have step $(xstep, ystep) = (2, 2)$;
- big boxes have step $bigbox = 10$, so are compost by 5×5 small boxes;
- the simulation time is enough, i.e. "$Sim\ Time$" $= 30^{\text{sec}}$
- an agent, the *red* one for example, has starting point $(X, Y, \theta) = (21, 0, 1.51)$;
- the goal is the small box $(goalX, goalY) = (10, 22)$;
- all the other agents are out of the field, it means for example setting all starting and goal parameters out of the field.

The obtained movement is shown in Fig. 5.1 an it is divided into 3 steps:

1 : acceleration until the maximum velocity is reached;

2 : movement with maximum velocity;

3 : deceleration when the goal is reached.

When the movement starts the only forbidden manoeuvre is going down, so the label down in the red car box is set to 1 whereas the others are 0. Any obstacle are in the field so label *main* and *real*, refereed to the movement, are always equal.

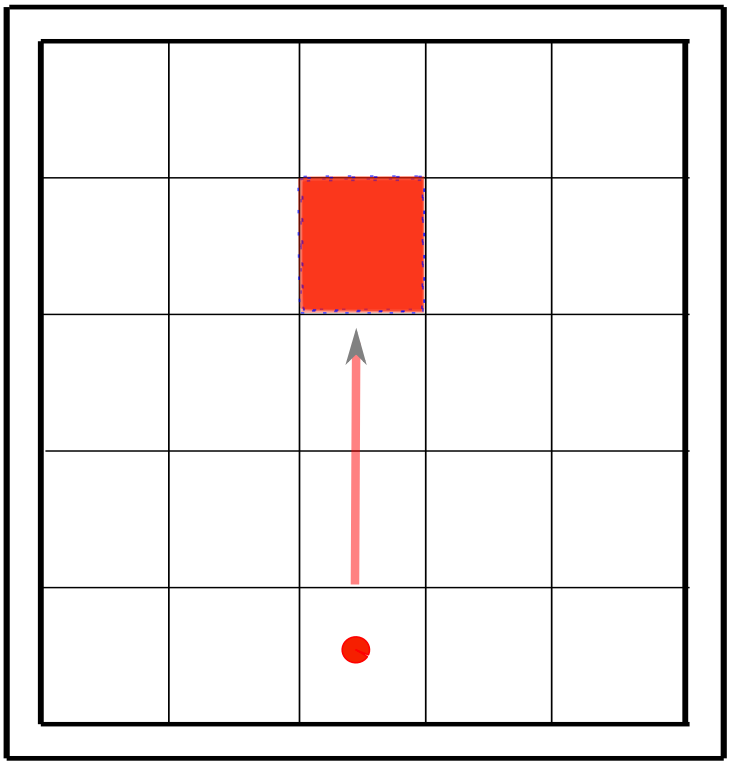


Figure 5.1: Example 1

5.2 Agent turn manoeuvre

The second example wants to show how a free curved path is made through the Manhattan policy of the first supervisor. The real path depends on starting and angle values. Thus, two big boxes paths are possible, they are shown in Fig. 5.2 by different coloured arrows.
The differences from the previous test are starting and goal positions. To give an example the parameters could be set as $(X, Y, \theta) = (2, 2, 0)$ and $(goalX, goalY) = (21, 21)$. The evolution of the example is shown in the

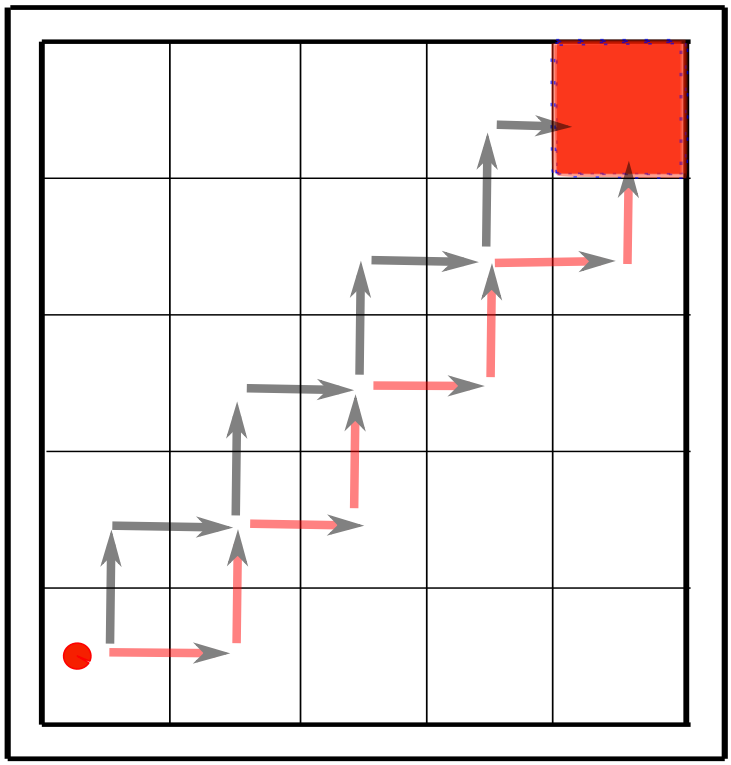
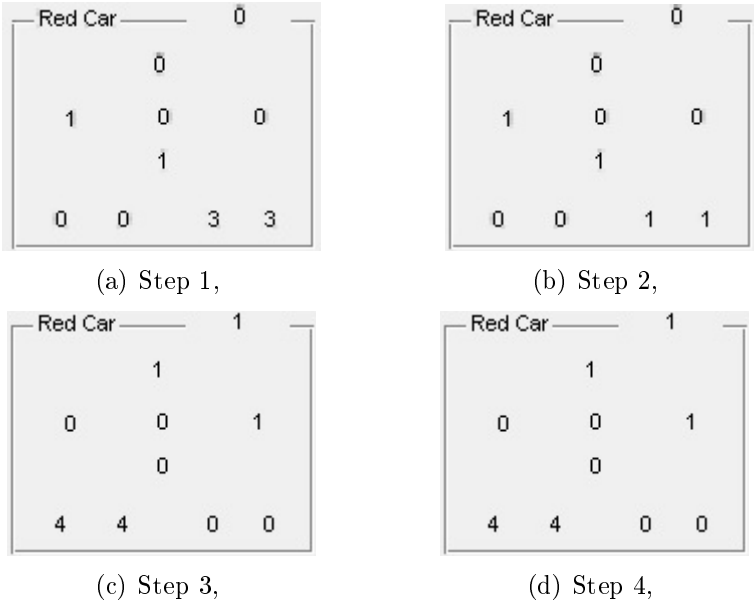


Figure 5.2: Example 2

Fig. 5.3 by the numerical screen. The subfigure 5.3(a) represents the initial movement, the agent is on the left down corner and can not follow both left and down directions. Initial value is not captured because it is valid only for a moment and it is represent only the holding command. Fig.s 5.3(b) and 5.3(c) represent movements while the goal is not reached, in this practical example switching direction are always *up* and *right*. The last image displayed, Fig. 5.3(d), represents the agent in the goal small box, it is possible observing that the last command is *stop* and the label in the high right side corner is

set to 1.



(a) Step 1, (b) Step 2,

(c) Step 3, (d) Step 4,

Figure 5.3: Displayed Screen

5.3 Two agents in the most simple conflicting path

The third example wants to show how a conflicting situation is solved. Let us consider the Fig.5.4, two agents must switch their position and their best path have a common big box to cross contemporaneously. A priority hierarchy is given: the red agent is more important than the blue one.
The movement development is partially shown in Fig.5.5. The simulation starts such the previous example following the their best path. When both are near to the central big box, Fig.5.5(a), the blue car notices about the red one but do not change its behaviour because of the small boxes studying. The behavioural changing is shown in Fig. 5.5(b), the blue agent can not follow in its best direction and the first supervisor chooses to stop it. The situation is numerical described by the grey panels, labels that indicate real and best direction have different values.
The red one is only globally limited by the central big box already taken but it is still moving. During the red robot moving, a path is unlocked for the blue one and it starts again as Fig.5.5(c). The movements follow on their new best path without any other obstacles.

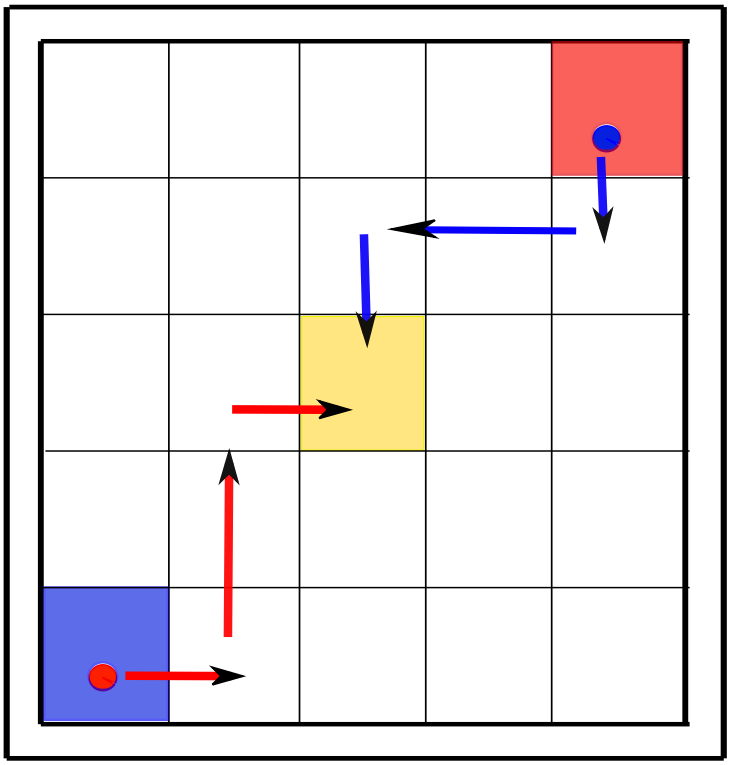
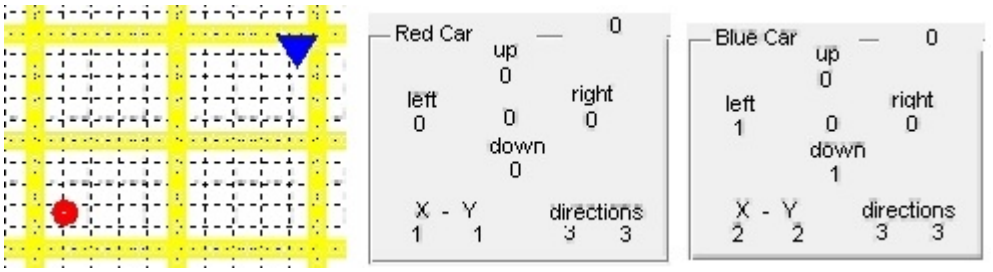
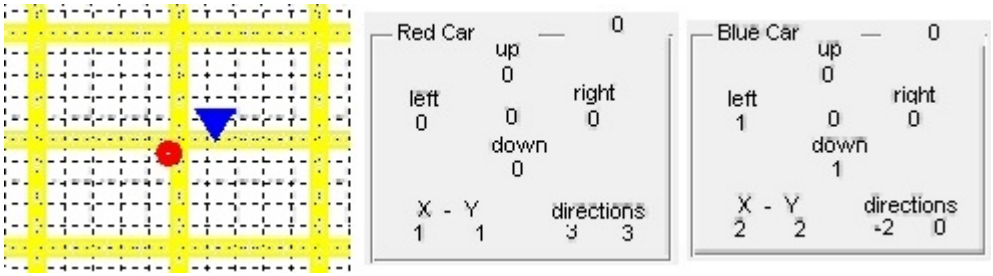


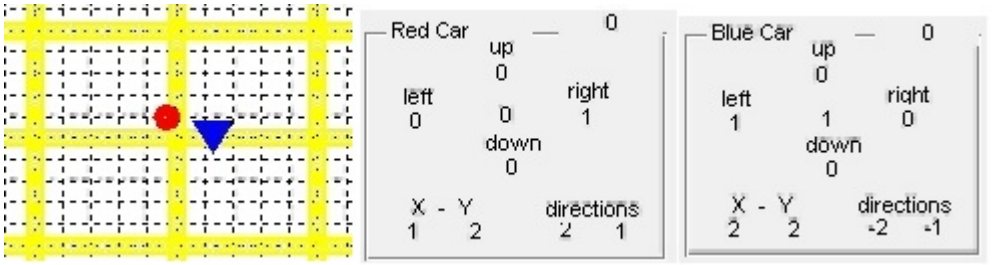
Figure 5.4: Example 3



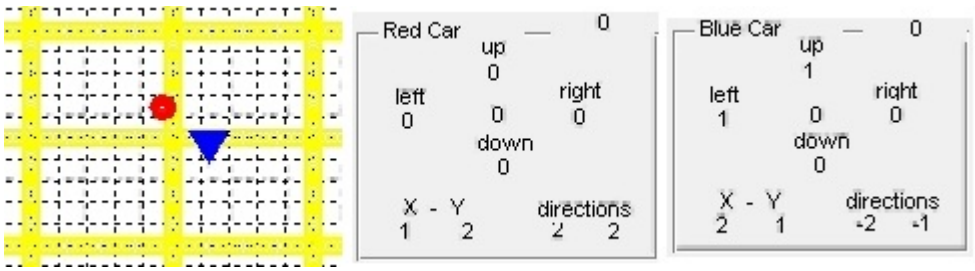
(a) Before,



(b) Stop,



(c) Getting free,



(d) Normal path,

Figure 5.5: Movement Development

5.4 Three agents with crossed trajectories

The fourth test wants to show how the system works if another agent is added. The situation shown in Fig. 5.6 is compost by three agent with the priority order: reg agent, green agent and blue agent. Let us consider as starting condition:

- red agent: $(X; Y, \theta) = (1, 1, 0)$ and $(goalX, goalY) = (20, 20)$;
- green agent: $(X; Y, \theta) = (4, 40, 0)$ and $(goalX, goalY) = (20, 1)$;
- blue agent: $(X; Y, \theta) = (40, 40, 0)$ and $(goalX, goalY) = (1, 1)$.

with the same field parameters of previous tests.
The analogies with the previous example are the movements of red and blue robots. However, the green agent is filling a big boxes that was used by the red one in the previous example. Thus, red and blue agents behaviours must change.

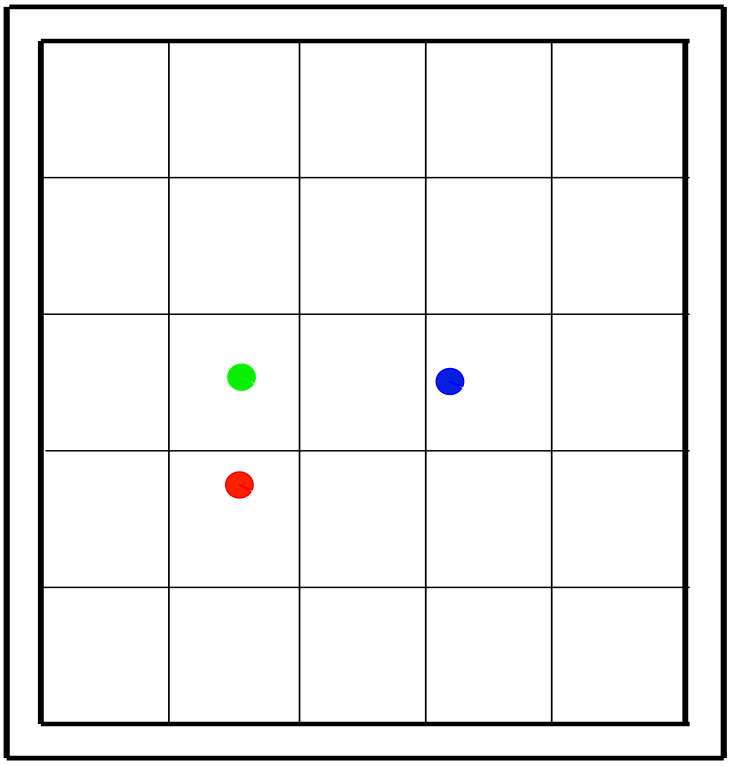
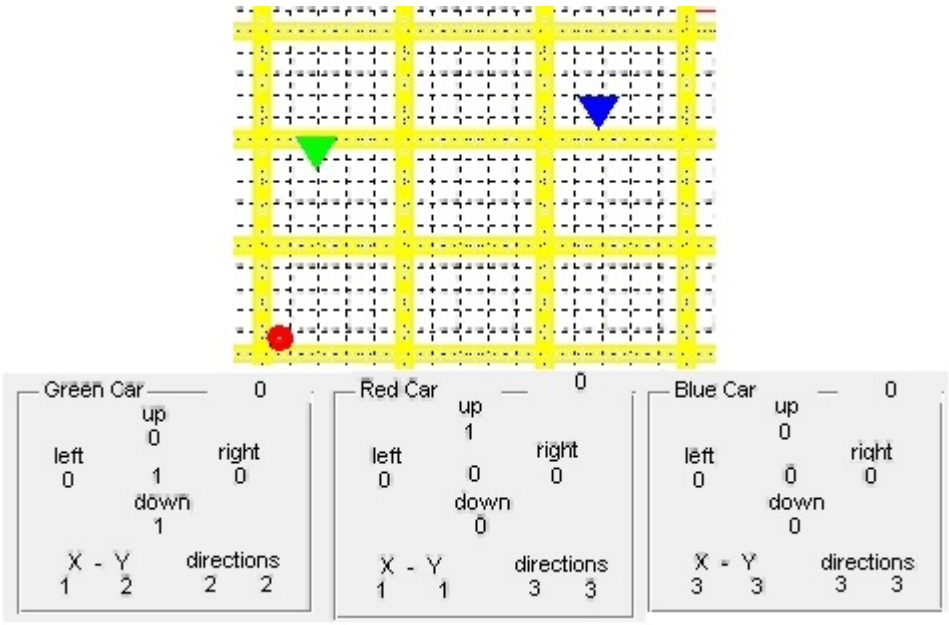
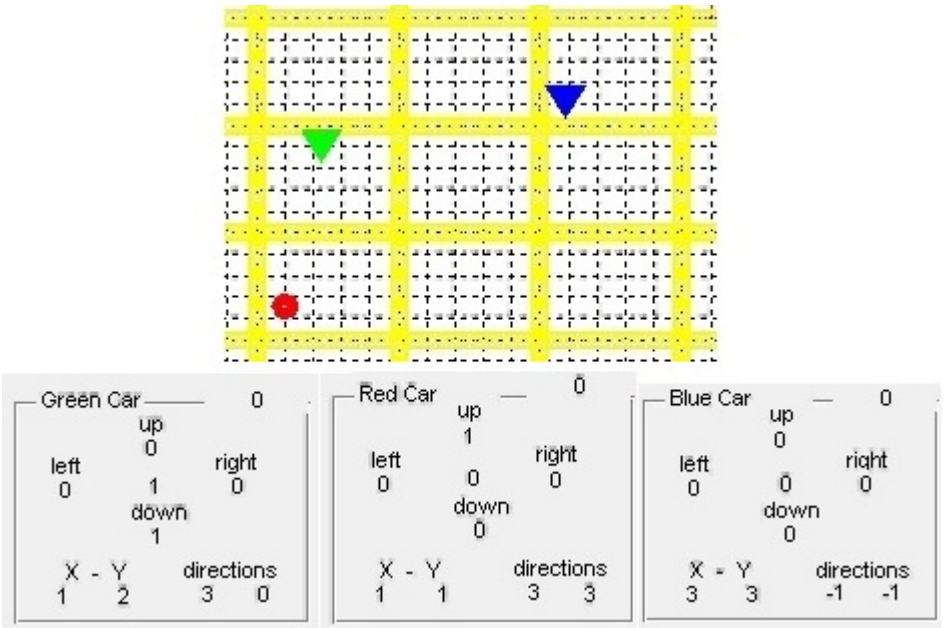


Figure 5.6: Example 4

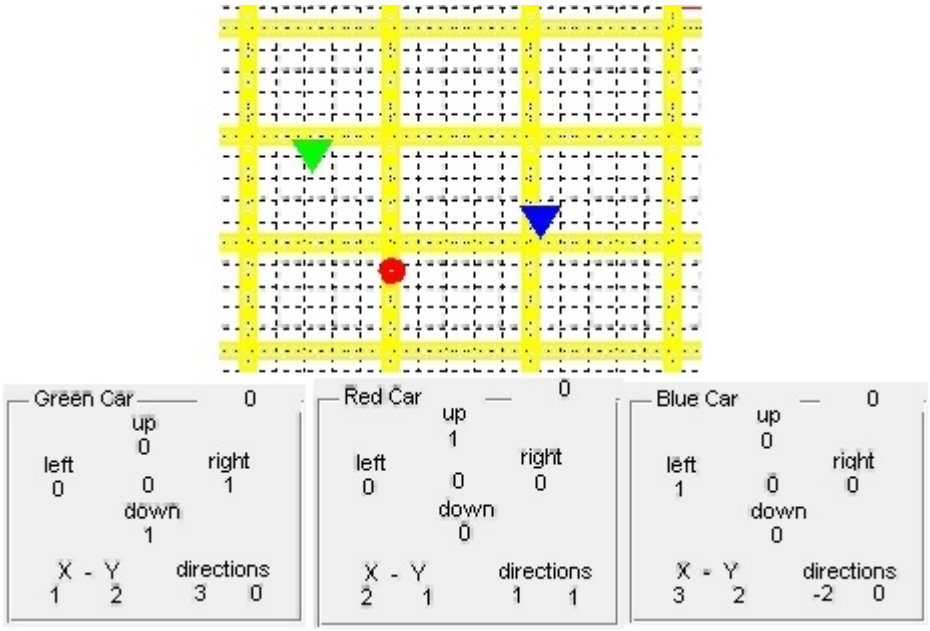


(a) Before,

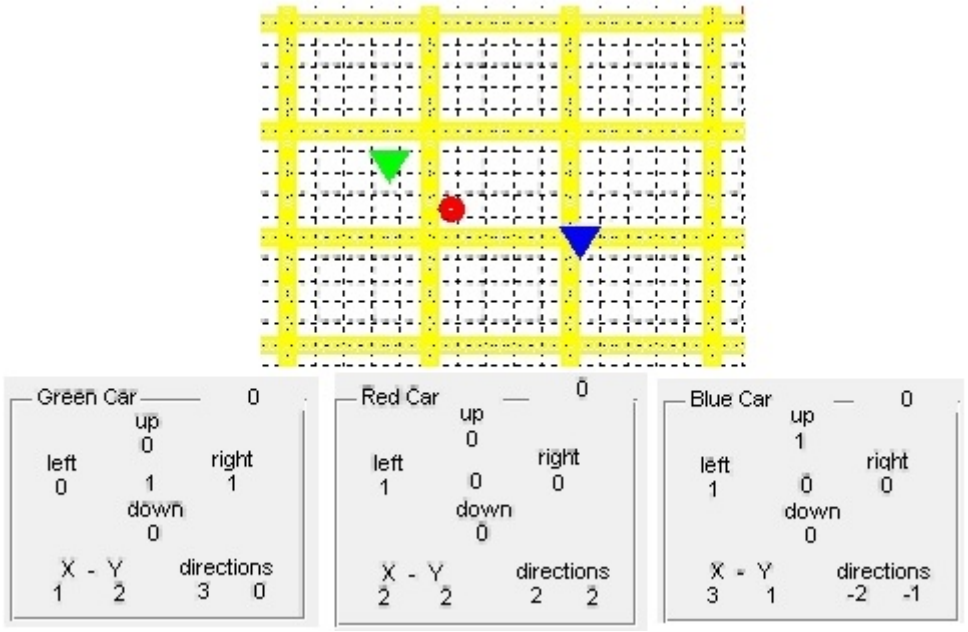


(b) First notice,

Figure 5.7: Movement Development with three agents



(a) Blue changing,



(b) Red Trajectory,

Figure 5.8: Movement Development with three agents

The movement development is shown in Fig. 5.8, the first part, Fig. 5.7(a), shows the movement before the conflicting knowledge. The second part, Fig. 5.7(b), shows that green agent notices that another agent could use its actual big box and changes its behaviour. Blue agent has the lower priority and can not enter in the central big box, so its behaviour changes as Fig. 5.8(a). The red robot has the highest priority and can choose all boxes not still empty, its evolution brings it to cross the central big boxes as Fig.5.8(b) shows. Moreover, the red agent movement allows the blue down to going down. Every agents can move now and the goals are reached by standard movements.

5.5 Four agents moving along the perimeter

This section wants to describe a new cases class compost by movements along the field perimeter. Let us consider following values for *four* agents on the field:

- red agent: $(X; Y, \theta) = (1, 1, 0)$ and $(goalX, goalY) = (2, 20)$;
- green agent: $(X; Y, \theta) = (4, 40, 0)$ and $(goalX, goalY) = (3, 2)$;
- blue agent: $(X; Y, \theta) = (40, 40, 0)$ and $(goalX, goalY) = (20, 1)$;
- cyan agent: $(X; Y, \theta) = (40, 1, 0)$ and $(goalX, goalY) = (20, 20)$;

The simulation is longer than the others and $SimTime = 50^{\text{sec}}$ whereas field parameter are still $(xstep, ystep) = (2, 2)$, $bigbox = 10$ and $(BorderX, BorderY) = (50, 50)$. The modelled movements is compost by pairs of agent moving along the perimeter with opposite direction as Fig. 5.12. The purpose of this set is showing how agent move when a direction is always forbidden by an obstacle. The priority hierarchy of agents is: red - green - blue - cyan.

The simulation starts as Fig. 5.10(a), robots move one toward another one but the only behavioural limitation is given by the perimeter presence. When agents are enough close, some agents will notice a new forbidden big box. This new situation is shown in Fig. 5.10(b) where green and cyan agents can not going straight now and must change direction. Obviously, green and cyan agent can move to the internal part of the field, Fig. 5.11(a) shows this movement and their display panels. This trajectories changing is done while red and blue agents keep following their best path.
The presented situation is different from previous examples: agents with lower priority must free their boxes because they are parts of the higher

priority agents path. The critical manoeuvres continues as Fig 5.11(b), agents are in assisted boxes and movements *up* and *down* are permitted. It means agents keep moving by those commands and when conflict is solved they will follow their best trajectories.

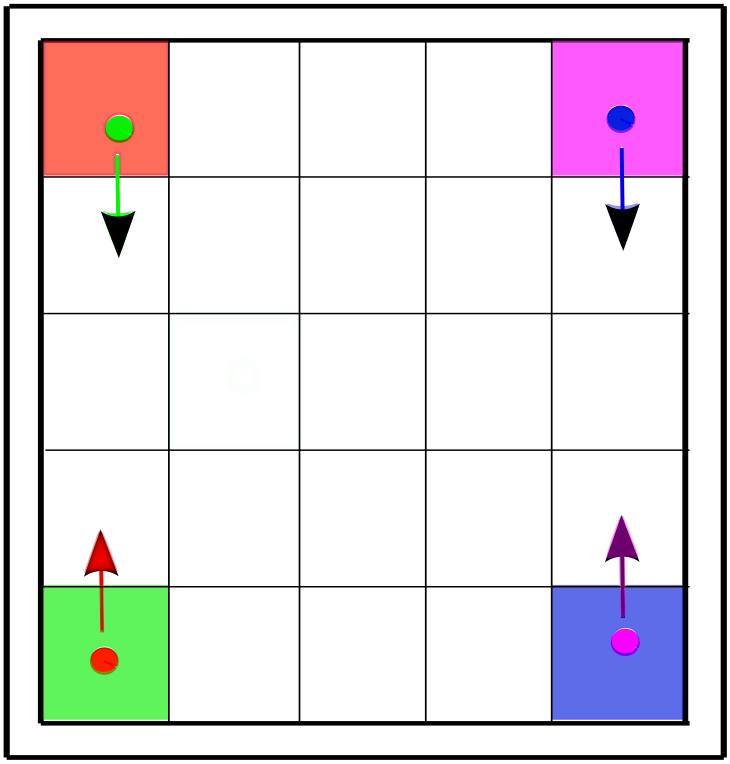
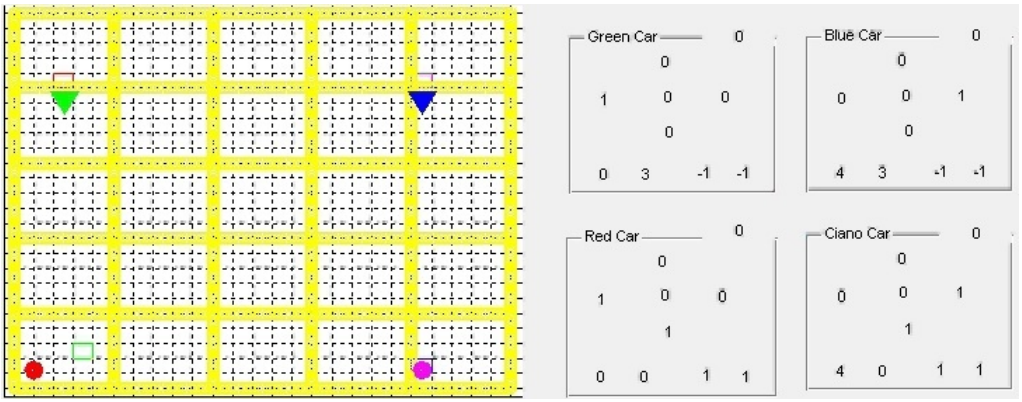
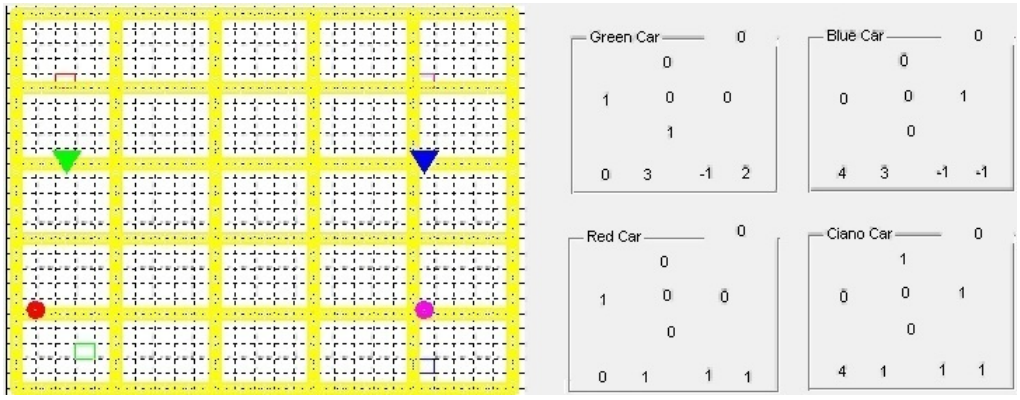


Figure 5.9: Example 5

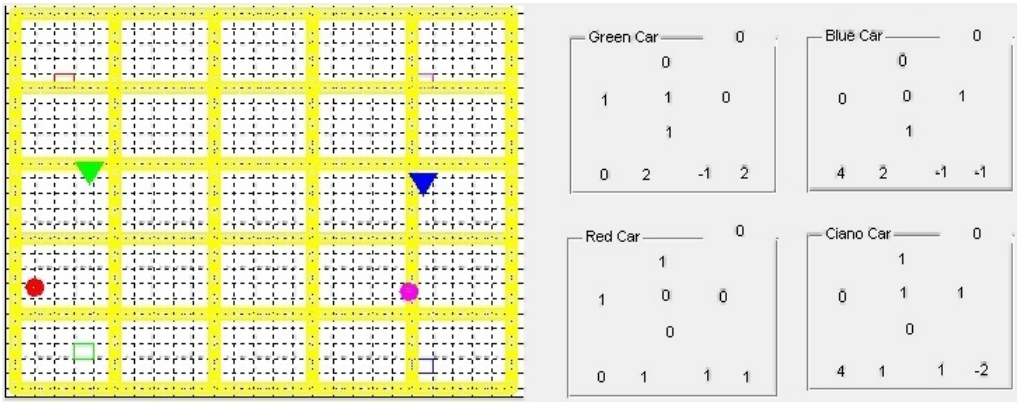


(a) Normal trajectories,

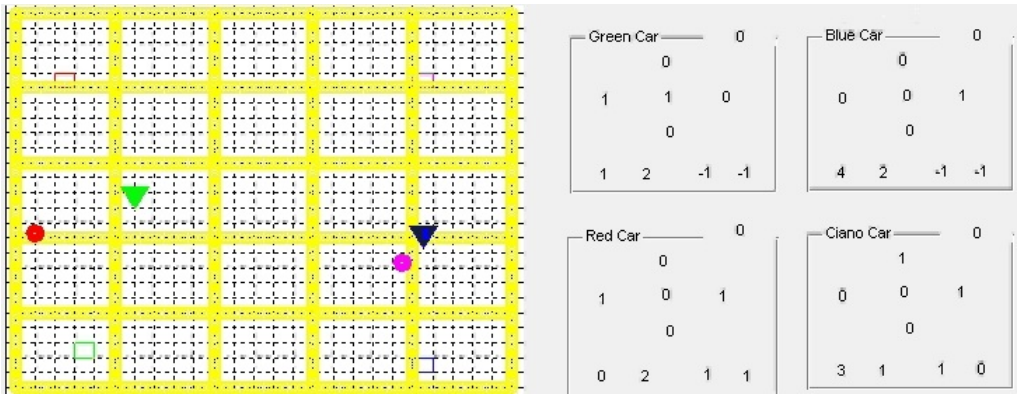


(b) Common big boxes notice,

Figure 5.10: Movement Development along the perimeter



(a) Agents large movements,



(b) Standard trajectories,

Figure 5.11: Movement Development along the perimeter

5.6 Four agents that cross their trajectories

This last section want to test the most complex possible situation. Even adding other agents entails to solve a set of situations already tested. Thus, this represents the high complexity of the rendezvous problem.
Let us consider the situation of Fig. 5.12. Four agents move on the field crossing their trajectories. The central big box is the common box of all paths because of their starting and goal positions. Following parameters has been used:

- red agent: $(X; Y, \theta) = (1, 1, 0)$ and $(goalX, goalY) = (20, 20)$;
- green agent: $(X; Y, \theta) = (4, 20, 0)$ and $(goalX, goalY) = (20, 10)$;
- blue agent: $(X; Y, \theta) = (40, 40, 0)$ and $(goalX, goalY) = (20, 10)$;
- cyan agent: $(X; Y, \theta) = (44, 20, 0)$ and $(goalX, goalY) = (2, 10)$.

The execution needs $SimTime = 50^{\text{sec}}$ and field parameters are always the same as previous section. The priority order is: red - green - blue - cyan agents.

The starting situation in Fig. 5.13(a) represents a beginning with conflicts because agents are still too close. However, ideal movement are not in according with conflicts and they start to follow their ideal path. The first conflict happens when the cyan agent wants to use a forbidden big boxes an in Fig. 5.13(b). Its new choose is *going down* in order not to fill the common big box. However the dynamic of the movement bring it to enter and to leave it immediately, so this example is affected with the most dangerous case. The structure of the field in boxes is important as minimums distance between agents without changing behaviour. The situation is still becoming complicated because of the green agents. Fig 5.14(a) shows what happens now: green agents keep going straight to its goal box and the blue agent must wait because its best paths are busy. The evolution of the movements is shown in Fig. 5.14(b), the cyan agent leaves the conflicting box and now the red one is allowed to enter into. Red agent has the highest priority and its behaviour is limited only when the close boxes are full.
The conflict is mainly solved but blue agent still changes its behaviour because of the red presence, Fig 5.15(a). When the red agent is faraway the blue robot still moves, it is now a bit larger than its best global trajectory but it plans a new best path and follows it. Fig. 5.15(b) shows a new generated conflict, similar to previous examples. In this case cyan agent waits the blue movement, after that restart the work and reach its goal. All agents are in their goal boxes now.

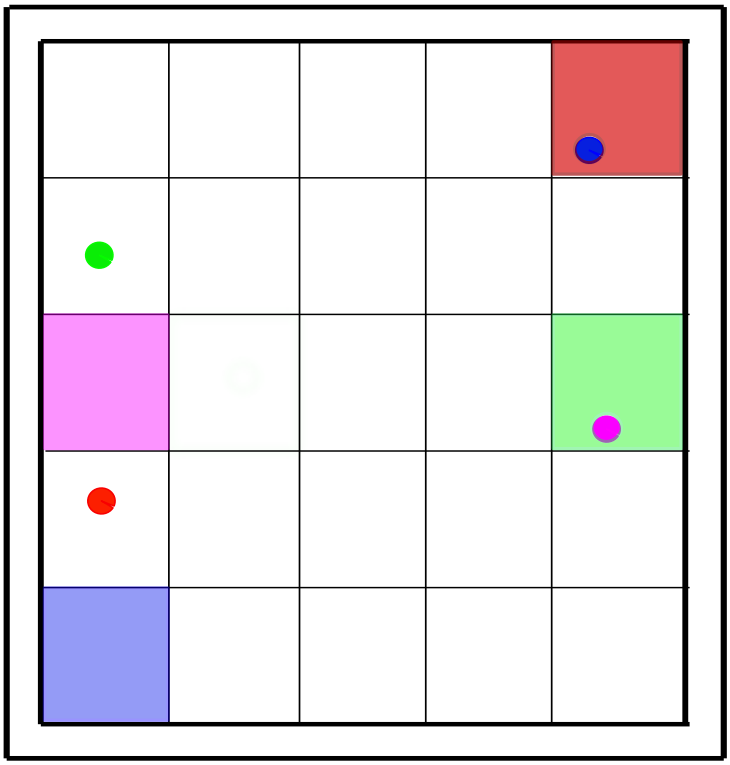
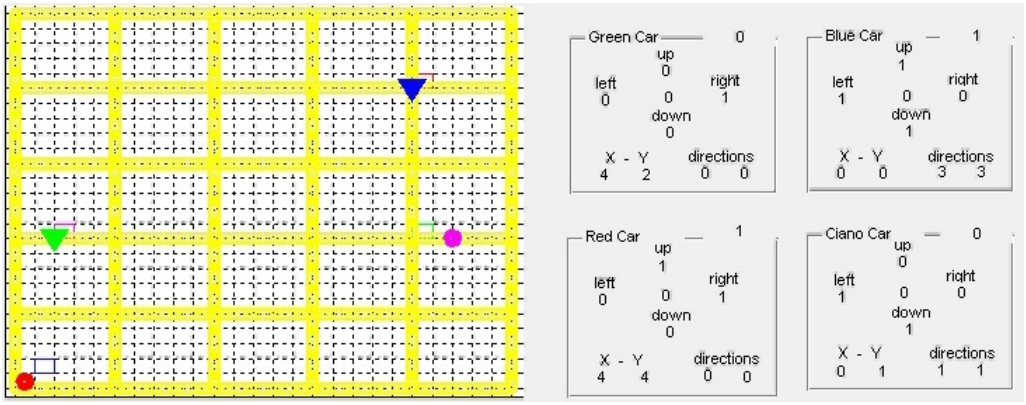
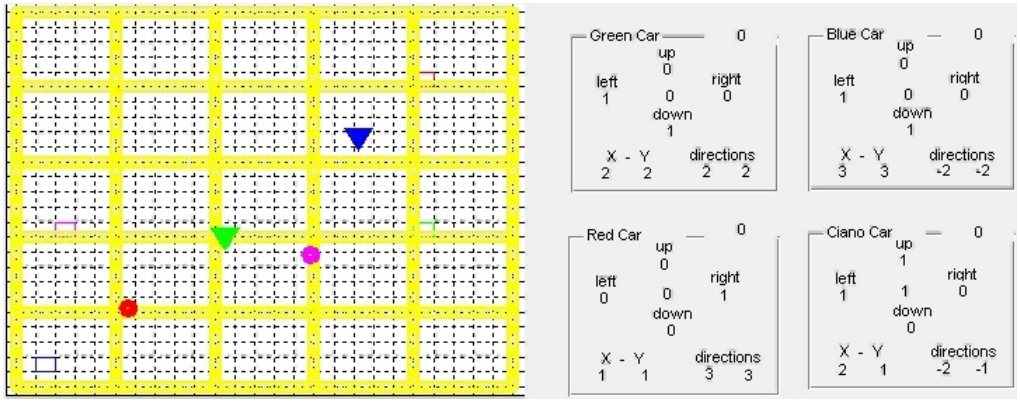


Figure 5.12: Example 6

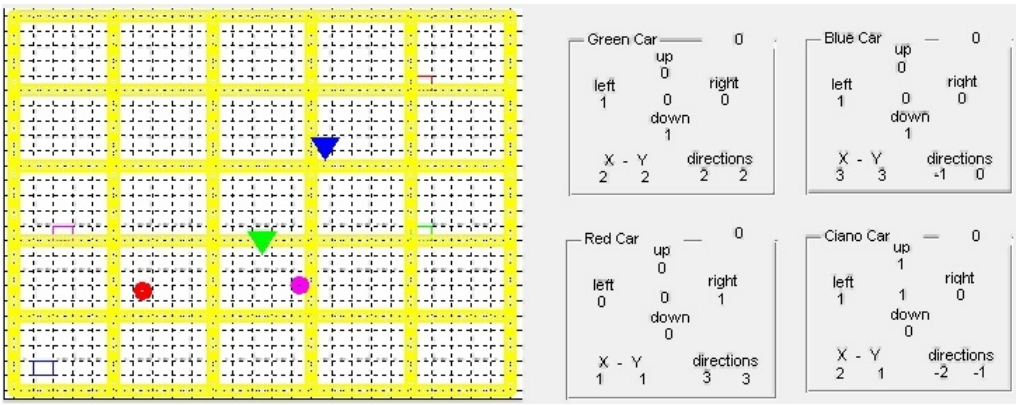


(a) Starting boxes,

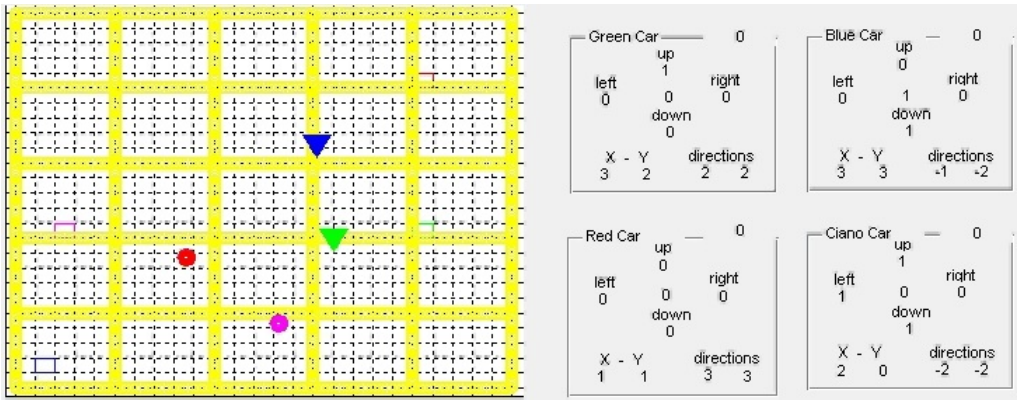


(b) First immediate conflict,

Figure 5.13: Second conflict

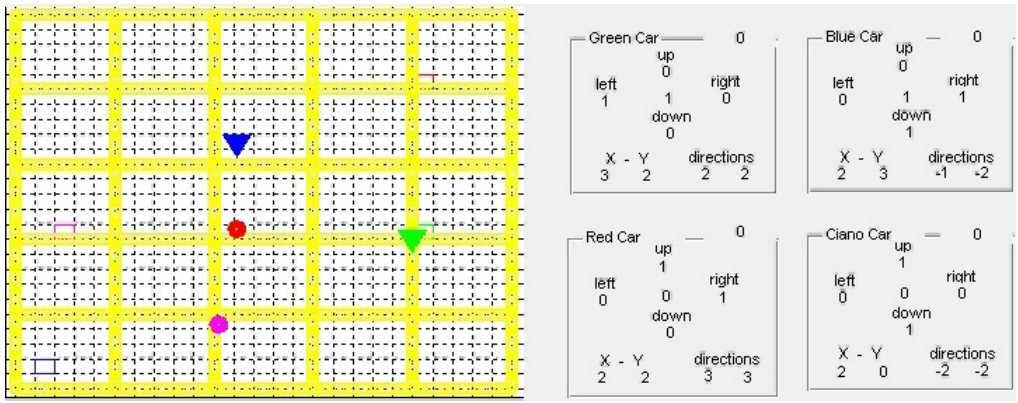


(a) Cyan agent critical movement,

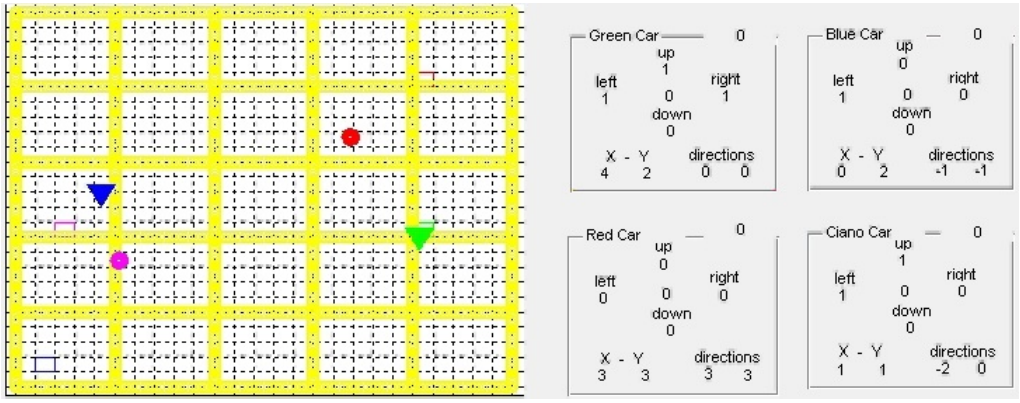


(b) Solved conflict,

Figure 5.14: Movement Development along the perimeter



(a) Toward goals,



(b) Last conflict,

Figure 5.15: Movement Development along the perimeter

Chapter 6

Conclusions

This chapter represents a brief summary of the work and results achieved. It has been addressed the rendezvous problem of a multi agent-system through a methodical approach based on a behavioural description. If each system is described by independent equations it is possible to study step by step each agent and therefore to plan a global strategy. The applied approach is based on the solution of supervisory controller synthesis for continuous plan with a discrete control interface. System descriptions are given as the Willem's behavioural theory. Previous works use this approach for practical situations, i.e. using a distillation column, but any multi-agents problems have never been solved.

Starting from works made at *Technische Universität* it is possible to show that exist a class of problems easily treated by abstractions. Each step make the system more general; information and signal change step by step. However, this approach allows us to use all known results from DES theory and it is easy to plan a supervisory approach.
The distinguishing feature of this setting is an input/output structure based on a product composition of the respective components. This is the normal case for continuous control systems and it is motivated by the considered class of plant models.

Previous works with this behavioural approach show simple continuous systems without any interactions with systems not controlled by the same supervisor. If we want to solve a complex multi agent issue it will be necessary to create a hierarchical structure and to decentralize decisions in this structure. A normal problem could be solved by a single supervisor, or a set of them that operate, on the same signal state space but not in multi agents systems. The main purpose is to derive a second level abstraction, to proof its theoretical implementability and admissibility. A condition has been proposed under which the second level supervisors, each enforcing its

own specification, will have an admissible the hierarchical composition for this class of problems. The second level supervisory rules allow combination between "fusion by intersection" and "fusion by union" behaviours. It is show that a proper combination of this rules with corresponding local decision rules results helpful in finding a global strategy for the whole system

The given framework is totally not dependent on the real implementation of each level. A set of rules and a behavioural description are given in order to have a well-thinking behavioural approach. It means that it is not important how each level is compost by, but the main thing is to respect some behavioural rules. It bring us to use this framework as easy planning instrument.

The second part of this master thesis work concerns to proof the framework correctness through an example. The chosen problem to solve concerns a set of agents that move in the same limited area. Thus, a coordination problem has been modelled by the framework. Framework is modelled to avoid the problem of not practicable when a huge number of agents are insert in the field. The hierarchical structure thought in this work allows us to solve the problem as numerical independent. Each agent is coordinate function of a given priority and it easy to add one runtime.
The planned model has proved the correctness of the hierarchical approach and a easy planning has been though through its rule.

Many development ideas are possible: other abstractions could be theorized in order to solve more complex. Examples are permitting set of goal decided runtime of studying particular coordination for agents in order to use cooperating agents. Even this model is easily got complicated by using particular representation of the supervisors or using particular powerful behavioural representations, i.e. Petri's nets.

Bibliography

[1] Michel Goossens,Frank Mittelbach and A. Samarin. *The LaTeX Companion*. Addison-Wesley, 1994.

[2] T. Moor and J. Raisch. *Think continuous, act discrete: DES techniques for continuous systems*. Proc. 10th Mediterranean Conference on Control and Automation, July 2002.

[3] J.C. Willem. *Notes on "Models and Behaviours"*. HYCON-EECI Graduate School on Control, Spring 2010.

[4] J.C. Willems. *Paradigms and puzzles in the theory of dynamic systems*. IEEE Transactions on Automatic Control, 1991.

[5] G. Caiabutto. *Manuale Matlab*.

[6] A. Pisano. *Guida introduttiva a Matlab-Simulink*.

[7] O. Barbarisi. *Corso introduttivo a Stateflow*.

[8] S. Di Cairano. *Modellare e simulare DES in Stateflow*.

[9] Y. Moshe. *Advanced MATLAB Graphics and GUI*. Department of Electrical Engineering, Technion - IIT Signal and Image Processing Laboratory, 2010.

[10] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. *Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems*. Cornell University - computer science technical report, 1993.

[11] W.M. Wonham. *Notes on control of discrete event systems*. Technical report, Department of Electrical & Computer Engineering, University of Toronto, 1999.

[12] Peter J. Ramadge and Willam M. Wonham. *The control of discrete event systems.* Proceedings of the IEEE, 1989.

[13] T.Moor and J.Raisch. *Supervisory control of hybrid systems within a behaviour framework.* Systems and Control Letters, 1999.

[14] T.Moor, J.M Davoren and B.D.O. Anderson. *Robust hybrid control from a behavioural perspective.* Technical report RSISE, 2002.

[15] T.Moor, J.M Davoren and B.D.O. Anderson. *Modular supervisory control of a of a class of hybrid systems in a behavioural framework* . European Control Conference. Porto, Portugal, 2001.

[16] T.Moor, J.M Davoren and J. Raisch. *Computational Advantages of a Two-level Hybrid Control Architecture.* Proceedings of the 40th IEEE Conference, 2001.

[17] T.Moor, J.M Davoren and J. Raisch. *Admissibility criteria for a hierarchical design of hybrid control system.* Conference on Analysis and Design of Hybrid Systems, Malo 2003.

[18] T.S. Yoo and S. Lafortune. *A General Architecture for Decentralized Supervisory Control of Discrete-Event Systems.* Discrete Event Dynamic Systems, 2003.

[19] T.Moor, J. Raisch. *Supervisory control of hybrid systems within a behavioural framework.* Systems and control letters, 1999.

[20] J. Lin, A.S. Morse and B.D.0.Anderson. *The Multi-Agent Rendezvous Problem.* 42nd IEEE Conference on Decision and Control, Maui(USA) 2003.

[21] T. Moor and J. Raisch. *Abstraction Based Supervisory Controller Synthesis for High Order Monotone Continuous Systems.* Analysis, and Design of Hybrid Systems, 2002.

[22] T. Moor, J. Raisch and S.D. *O'*Young. *Discrete supervisory control of hybrid systems based on l-complete approximations* .Journal of Discrete Event Dynamic Systems, 2002.

[23] T. Moor, J. Raisch and S.D. *O'*Young. *Discrete Approximation and Supervisory Control of Continuous Systems.*IEEE TRANSACTIONS ON AUTOMATIC CONTROL, 1998

[24] P.J. Ramadge and W.M. Wonham. *Supervisory control of a class of discrete event process.*SIAM J. Control and Optimization, 1987

[25] K. Rudie and W.M. Wonham. *Think globally, act locally: decentralized supervisory control.*IEEE Transactions on Automatic Control, 1992

[26] T. Moor, J. Raisch and J.M. Davoren. *Computational advantages of a two-level hybrid control architecture.*IEEE Conf. Decision and Control, 2001

[27] K.H. Khalil. *Nonlinear Systems.* Prentice-Hall. Second edition, 1996

[28] T. Moor and J. Raisch. *Abstraction Based Supervisory Controller Synthesis for High Order Monotone Continuous Systems.* Modelling, Analysis, and Design of Hybrid Systems, 2002.

[29] J. Liu and H. Darabi. *Ramadge-Wonham supervisory control of mobile robots: lessons from practice.* Robotics and Automation conference - ICRA, 2002.

[30] A. DiFebbraro and A. Giua. *Sistemi ad eventi discreti.* McGraw hill, 2002

Appendix A

Priority decision Matlab code

```
function [fourth, third, second, first] = fcn(dx,dy)
if (abs(dx)==0 && abs(dy)==0)
   first=0;
   second=0;
   third=0;
   fourth=0;
elseif (abs(dx) > abs(dy))
   if (dx < 0)
      first=2;    if (dy < 0)
      second=1;
   else
      second=-1;
   end
   third= - second;
   fourth=0;
   else
   first=-2;    if (dy < 0)
      second=1;
   else
      second=-1;
   end
   third= - second;
   fourth=0;
end
```

```
elseif (abs(dx) < abs(dy))
   if (dy < 0)
      first=1;
   if (dx < 0)
      second=2;
   else
      second=-2;
   end
   third= - second;
   fourth=0;
   else
      first=-1;
      if (dx < 0)
         second=2;
      else
         second=-2;
      end
   third= - second;
   fourth=0;
end
else
   first=3;
   second=3;
   third=3;
   fourth=3;
end
```

Appendix B

Border controller Matlab code

```
function [noUp, noDown, noLeft, noRight] = fcn(xCar, yCar, borderX, borderY, BBstep)
  noUp=0;
  noDown=0;
  noLeft=0;
  noRight=0;
  maxBBx= floor(borderX/BBstep);
  maxBBy=floor(borderY/BBstep);
  if (xCar==0)
    noLeft=1;
  end
  if (yCar==0)
    noDown=1;
  end
  if (xCar==maxBBx-1)
    noRight=1;
  end
  if (yCar==maxBBy-1)
    noUp=1;
  end
end
```

Appendix C

Big boxes controller Matlab code

```
function [noUp, noDown, noLeft, noRight, noHere] =
    fcn(x1, y1, x0, y0, x2, y2, x3, y3, x4, y4)
  noUp=0;
  noDown=0;
  noLeft=0;
  noRight=0;
  noHere=0;
  if (x0==x1 && y0+1==y1)
    noUp=1;
  end
  if (x0==x1 && y0-1==y1)
    noDown=1;
  end
  if (x0-1==x1 && y0==y1)
    noLeft=1;
  end
  if (x0+1==x1 && y0==y1)
    noRight=1;
  end
```

```
if (x0==x2 && y0+1==y2)
   noUp=1;
end
if (x0==x2 && y0-1==y2)
   noDown=1;
end
if (x0-1==x2 && y0==y2)
   noLeft=1;
end
if (x0+1==x2 && y0==y2)
   noRight=1;
end
if (x0==x3 && y0+1==y3)
   noUp=1;
end
if (x0==x3 && y0-1==y3)
   noDown=1;
end
if (x0-1==x3 && y0==y3)
   noLeft=1;
end
if (x0+1==x3 && y0==y3)
   noRight=1;
end
if (x0==x4 && y0+1==y4)
   noUp=1;
end
if (x0==x4 && y0-1==y4)
   noDown=1;
end
if (x0-1==x4 && y0==y4)
   noLeft=1;
end
if (x0+1==x4 && y0==y4)
   noRight=1;
end
```

Appendix D

Single direction valuation

```
function ok = fcn(dir, x, y, BBstep, smallStep, noUp, noDown, noLeft, noRight)
  actual_x = floor(x*smallStep/BBstep);
  actual_y = floor(y*smallStep/BBstep);
  switch (dir)
    case 0
      ok = 1;
    case 1
      y_max = y+2;
      max_y = floor(y_max*smallStep/BBstep);
      if (max_y == actual_y)
        ok=1;
      else
    if (noUp==1)
      ok=0;
    else ok=1;
    end
  end
  case -1
    y_min = y-2;
    min_y = floor(y_min*smallStep/BBstep);
    if (min_y == actual_y)
      ok=1;
    else
    if (noDown==1)
      ok=0;
    else ok=1;
    end
  end
```

```
    case 2
      x_max = x+2;
      max_x = floor(x_max*smallStep/BBstep);
      if (max_x == actual_x)
        ok=1;
      else
      if (noRight==1)
        ok=0;
      else ok=1;
      end
    end
    case -2
      x_min = x-2;
      min_x = floor(x_min*smallStep/BBstep);
      if (min_x == actual_x)
        ok=1;
      else
      if (noLeft==1)
        ok=0;
      else ok=1;
      end
    end
    otherwise
      ok=0;
end
```

Appendix E

GUI main function

```
function varargout = simulation(varargin)
function start_pushbutton_Callback(hObject, eventdata, handles)
axes(handles.axes1)
x0=str2num(get(handles.x0_edit,'String'));
y0=str2num(get(handles.y0_edit,'String'));
teta0=str2num(get(handles.teta0_edit,'String'));
x1=str2num(get(handles.x1_edit,'String'));
y1=str2num(get(handles.y1_edit,'String'));
teta1=str2num(get(handles.teta1_edit,'String'));
x2=str2num(get(handles.x2_edit,'String'));
y2=str2num(get(handles.y2_edit,'String'));
teta2=str2num(get(handles.teta2_edit,'String'));
x3=str2num(get(handles.x3_edit,'String'));
y3=str2num(get(handles.y3_edit,'String'));
teta3=str2num(get(handles.teta3_edit,'String'));
x4=str2num(get(handles.x4_edit,'String'));
y4=str2num(get(handles.y4_edit,'String'));
teta4=str2num(get(handles.teta4_edit,'String'));
Xstep=str2num(get(handles.xStep_edit,'String'));
Ystep=str2num(get(handles.yStep_edit,'String'));
BBstep=str2num(get(handles.bigbox_edit,'String'));
borderX =str2num(get(handles.BorderX_edit,'String'));
borderY =str2num(get(handles.BorderY_edit,'String'));
simTime =str2num(get(handles.SimTime_edit,'String'));
goalX_car0=str2num(get(handles.car0goalX_edit,'String'));
goalY_car0=str2num(get(handles.car0goalY_edit,'String'));
goalX_car1=str2num(get(handles.car1goalX_edit,'String'));
goalY_car1=str2num(get(handles.car1goalY_edit,'String'));
goalX_car2=str2num(get(handles.car2goalX_edit,'String'));
goalY_car2=str2num(get(handles.car2goalY_edit,'String'));
```

```
goalX_car3=str2num(get(handles.car3goalX_edit,'String'));
goalY_car3=str2num(get(handles.car3goalY_edit,'String'));
goalX_car4=str2num(get(handles.car4goalX_edit,'String'));
goalY_car4=str2num(get(handles.car4goalY_edit,'String'));
cla(handles.axes1,'reset');
options = simset('SrcWorkspace','current');
sim('CarsInThePlane',[0 simTime],options);
c0x=x_car0(1,1);
c0y=y_car0(1,1);
c1x=x_car1(1,1);
c1y=y_car1(1,1);
c2x=x_car2(1,1);
c2y=y_car2(1,1);
c3x=x_car3(1,1);
c3y=y_car3(1,1);
c4x=x_car4(1,1);
c4y=y_car4(1,1);
axis([-1 borderX+1 -1 borderY+1]);
title('Cars');
set(gca,'XTick',0:Xstep:borderX);
set(gca,'YTick',0:Ystep:borderY);
set(gca,'XTickMode','manual')
grid on;
hold on;
up1=c1_up;
down1=c1_down;
left1=c1_left;
right1=c1_right;
here1=c1_here;
up0=c0_up;
down0=c0_down;
left0=c0_left;
right0=c0_right;
here0=c0_here;
up2=c2_up;
down2=c2_down;
left2=c2_left;
right2=c2_right;
here2=c2_here;
```

up3=c3_up;
down3=c3_down;
left3=c3_left;
right3=c3_right;
here3=c3_here;
up4=c4_up;
down4=c4_down;
left4=c4_left;
right4=c4_right;
here4=c4_here;
c1InGoal=c1_BBgoal;
c0InGoal=c0_BBgoal;
c2InGoal=c2_BBgoal;
c3InGoal=c3_BBgoal;
c4InGoal=c4_BBgoal;
c0_BBx=BBx0;
c1_BBx=BBx1;
c2_BBx=BBx2;
c3_BBx=BBx3;
c4_BBx=BBx4;
c0_BBy=BBy0;
c1_BBy=BBy1;
c2_BBy=BBy2;
c3_BBy=BBy3;
c4_BBy=BBy4;
md0=main_dir0;
md1=main_dir1;
md2=main_dir2;
md3=main_dir3;
md4=main_dir4;
rd0=real_dir0;
rd1=real_dir1;
rd2=real_dir2;
rd3=real_dir3;
rd4=real_dir4;

```
plot(goalX_car0*Xstep:.1:goalX_car0*Xstep + Xstep,
goalY_car0*Ystep,'r-', 'LineWidth', 1 );
plot(goalX_car0*Xstep:.1:goalX_car0*Xstep +
   Xstep,goalY_car0*Ystep + Ystep,'r-', 'LineWidth', 1 );
plot(goalX_car0*Xstep,goalY_car0*Ystep:.1:goalY_car0*Ystep +
   Ystep,'r-', 'LineWidth', 1 );
plot(goalX_car0*Xstep + Xstep,goalY_car0*Ystep:.1:goalY_car0*Ystep +
   Ystep,'r-', 'LineWidth', 1 );
plot(goalX_car1*Xstep:.1:goalX_car1*Xstep + Xstep,
   goalY_car1*Ystep,'g-', 'LineWidth', 1 );
plot(goalX_car1*Xstep:.1:goalX_car1*Xstep +
   Xstep,goalY_car1*Ystep + Ystep,'g-', 'LineWidth', 1 );
plot(goalX_car1*Xstep,goalY_car1*Ystep:.1:goalY_car1*Ystep +
   Ystep,'g-', 'LineWidth', 1 );
plot(goalX_car1*Xstep + Xstep,goalY_car1*Ystep:.1:goalY_car1*Ystep +
   Ystep,'g-', 'LineWidth', 1 );
plot(goalX_car2*Xstep:.1:goalX_car2*Xstep + Xstep,
   goalY_car2*Ystep,'b-', 'LineWidth', 1 );
plot(goalX_car2*Xstep:.1:goalX_car2*Xstep + Xstep,goalY_car2*Ystep +
   Ystep,'b-', 'LineWidth', 1 );
plot(goalX_car2*Xstep,goalY_car2*Ystep:.1:goalY_car2*Ystep +
   Ystep,'b-', 'LineWidth', 1 );
plot(goalX_car2*Xstep + Xstep,goalY_car2*Ystep:.1:goalY_car2*Ystep +
   Ystep,'b-', 'LineWidth', 1 );
plot(goalX_car3*Xstep:.1:goalX_car3*Xstep + Xstep,
   goalY_car3*Ystep,'m-', 'LineWidth', 1 );
plot(goalX_car3*Xstep:.1:goalX_car3*Xstep + Xstep,goalY_car3*Ystep +
   Ystep,'m-', 'LineWidth', 1 );
plot(goalX_car3*Xstep,goalY_car3*Ystep:.1:goalY_car3*Ystep +
   Ystep,'m-', 'LineWidth', 1 );
plot(goalX_car3*Xstep + Xstep,goalY_car3*Ystep:.1:goalY_car3*Ystep +
   Ystep,'m-', 'LineWidth', 1 );
plot(goalX_car4*Xstep:.1:goalX_car4*Xstep + Xstep,
   goalY_car4*Ystep,'c-', 'LineWidth', 1 );
plot(goalX_car4*Xstep:.1:goalX_car4*Xstep + Xstep,goalY_car4*Ystep +
   Ystep,'c-', 'LineWidth', 1 );
plot(goalX_car4*Xstep,goalY_car4*Ystep:.1:goalY_car4*Ystep +
   Ystep,'c-', 'LineWidth', 1 );
plot(goalX_car4*Xstep + Xstep,goalY_car4*Ystep:.1:goalY_car4*Ystep +
   Ystep,'c-', 'LineWidth', 1 );
```

```
for indexX = 0:BBstep:borderX
plot(indexX, 0:.4:borderY,'yo', 'LineWidth', 1 );
end
for indexY = 0:BBstep:borderY
plot(0:.4:borderX, indexY,'yo', 'LineWidth', 1 );
end
h0 = plot(c0x,c0y, 'ro', 'LineWidth', 5, 'XDataSource','c0x', 'YDataSource', 'c0y');
h1 = plot(c1x,c1y, 'gv', 'LineWidth', 5, 'XDataSource','c1x', 'YDataSource', 'c1y');
h2 = plot(c2x,c2y, 'bv', 'LineWidth', 5, 'XDataSource','c2x', 'YDataSource', 'c2y');
h3 = plot(c3x,c3y, 'mo', 'LineWidth', 5, 'XDataSource','c3x', 'YDataSource', 'c3y');
h4 = plot(c4x,c4y, 'cv', 'LineWidth', 5, 'XDataSource','c4x', 'YDataSource', 'c4y');
kend = size (x_car0);
for k = 1:500:kend(1,1)
   c0x=x_car0(k,1);
   c0y=y_car0(k,1);
   c1x=x_car1(k,1);
   c1y=y_car1(k,1);
   c2x=x_car2(k,1);
   c2y=y_car2(k,1);
   c3x=x_car3(k,1);
   c3y=y_car3(k,1);
   c4x=x_car4(k,1);
   c4y=y_car4(k,1);
   refreshdata(h0, 'caller')
   refreshdata(h1, 'caller')
   refreshdata(h2, 'caller')
   refreshdata(h3, 'caller')
   refreshdata(h4, 'caller')v     drawnow;
   set(handles.c1_up_text,'String',num2str(up1(k,1)));
   set(handles.c1_down_text,'String',num2str(down1(k,1)));
   set(handles.c1_left_text,'String',num2str(left1(k,1)));
   set(handles.c1_right_text,'String',num2str(right1(k,1)));
   set(handles.c1_here_text,'String',num2str(here1(k,1)));
   set(handles.c0_up_text,'String',num2str(up0(k,1)));
   set(handles.c0_down_text,'String',num2str(down0(k,1)));
   set(handles.c0_left_text,'String',num2str(left0(k,1)));
   set(handles.c0_right_text,'String',num2str(right0(k,1)));
   set(handles.c0_here_text,'String',num2str(here0(k,1)));
```

```
    set(handles.c2_up_text,'String',num2str(up2(k,1)));
    set(handles.c2_down_text,'String',num2str(down2(k,1)));
    set(handles.c2_left_text,'String',num2str(left2(k,1)));
    set(handles.c2_right_text,'String',num2str(right2(k,1)));
    set(handles.c2_here_text,'String',num2str(here2(k,1)));
    set(handles.c3_up_text,'String',num2str(up3(k,1)));
    set(handles.c3_down_text,'String',num2str(down3(k,1)));
    set(handles.c3_left_text,'String',num2str(left3(k,1)));
    set(handles.c3_right_text,'String',num2str(right3(k,1)));
    set(handles.c3_here_text,'String',num2str(here3(k,1)));
    set(handles.c4_up_text,'String',num2str(up4(k,1)));
    set(handles.c4_down_text,'String',num2str(down4(k,1)));
    set(handles.c4_left_text,'String',num2str(left4(k,1)));
    set(handles.c4_right_text,'String',num2str(right4(k,1)));
    set(handles.c4_here_text,'String',num2str(here4(k,1)));
    set(handles.c0_inGoal_text,'String',num2str(c0InGoal(k,1)));
    set(handles.c1_inGoal_text,'String',num2str(c1InGoal(k,1)));
    set(handles.c2_inGoal_text,'String',num2str(c2InGoal(k,1)));
    set(handles.c4_inGoal_text,'String',num2str(c3InGoal(k,1)));
    set(handles.c4_inGoal_text,'String',num2str(c4InGoal(k,1)));
    set(handles.BBx0_text,'String',num2str(c0_BBx(k,1)));
    set(handles.BBy0_text,'String',num2str(c0_BBy(k,1)));
    set(handles.BBx1_text,'String',num2str(c1_BBx(k,1)));
    set(handles.BBy1_text,'String',num2str(c1_BBy(k,1)));
    set(handles.BBx2_text,'String',num2str(c2_BBx(k,1)));
    set(handles.BBy2_text,'String',num2str(c2_BBy(k,1)));
    set(handles.BBx3_text,'String',num2str(c3_BBx(k,1)));
    set(handles.BBy3_text,'String',num2str(c3_BBy(k,1)));
    set(handles.BBx4_text,'String',num2str(c4_BBx(k,1)));
    set(handles.BBy4_text,'String',num2str(c4_BBy(k,1)));
    set(handles.mainDir1_text,'String',num2str(md1(k,1)));
    set(handles.mainDir0_text,'String',num2str(md0(k,1)));
    set(handles.mainDir2_text,'String',num2str(md2(k,1)));
    set(handles.mainDir3_text,'String',num2str(md3(k,1)));
    set(handles.mainDir4_text,'String',num2str(md4(k,1)));
    set(handles.realC0_text,'String',num2str(rd0(k,1)));
    set(handles.realC1_text,'String',num2str(rd1(k,1)));
    set(handles.realC2_text,'String',num2str(rd2(k,1)));
    set(handles.realC3_text,'String',num2str(rd3(k,1)));
    set(handles.realC4_text,'String',num2str(rd4(k,1)));
    guidata(hObject, handles);
end
```