Università di Pisa

FACOLTÀ DI INGEGNERIA
Corso di Laurea in Ingegneria Elettronica

Distributed Intrusion Detection for Secure Cooperative Multi–Agent Systems

Candidato:
Gianni Valenti
Matricola 213457

Relatori:
Prof. A. Bicchi
Prof. A. Balestrino
Ing. A. Fagiolini

Anno Accademico 2006–2007

I never even thought about
whether or not they understand
what I'm doing...
The emotional reaction is all
that matters as long as there's
some feeling of communication,
it isn't necessary that it be
understood.

John W. Coltrane

Abstract

In this thesis we propose a solution for the problem of *detecting intruders* in an open set of *cooperative agents*. An agent can perform a finite set of maneuvers and is modeled by a *hybrid system* whose state is a continuous and a discrete part, representing the agents' physical evolution and logical variables respectively. Each agent plans its behavior and chooses the appropriate maneuver to perform following a common set of shared rules designed to ensure the safety of the entire system. Since the number of agents is unknown, and since these agents have a limited knowledge of their neighborhood, they can make decisions based only on their own position, and on the configuration of a limited number of surrounding agents. Such a planning strategy is said to be *decentralized*.

The expounded solution is an *Intrusion Detecting System* (IDS), based on a decentralized monitoring strategy, performed by several common local monitor modules running on–board each agent. This module tries to evaluate the behavior of neighboring agents by estimating the occurrence of the logical events described in the shared rule set. Since each monitor has a limited vision of its neighbors, in many cases it can remain uncertain about the correctness of the monitored agent's behavior. In order to solve this problem we developed a *distributed consensus algorithm* which, by introducing communication between agents, enhances the intrusion detection capabilities of single monitors. The effectiveness of our solution has been proved by in-depth simulations and a theoretical demonstration of the convergence of the consensus algorithm.

Sommario

In questa tesi viene proposta una soluzione al problema dell'*individuazione di intrusi* all'interno di un insieme aperto di *agenti cooperativi*. Un agente può effettuare un insieme finito di manovre ed è modellato tramite un *sistema ibrido* il cui stato è una combinazione di una parte continua, che rappresenta l'evoluzione fisica dell'agente, e una parte discreta, che rappresenta le sue variabili logiche. Ciascun agente pianifica il proprio comportamento e sceglie la manovra appropriata da eseguire seguendo un insieme comune di regole condivise progettate per garantire la sicurezza dell'intero sistema. Dato che il numero di agenti è sconosciuto e che questi agenti hanno una conoscenza limitata del loro intorno, possono prendere decisioni basandosi solamente sulla loro configurazione e sulla posizione di un numero limitato di agenti circostanti; una strategia di questo tipo è detta *decentralizzata*.

La soluzione proposta è un *sistema di individuazione di intrusi* (Intrusion Detection System), basato su una strategia di monitoraggio decentralizzata, effettuata da vari moduli monitor locali installati su ogni agente. Questo modulo cerca di valutare il comportamento degli agenti vicini stimando l'occorrenza degli eventi logici descritti nell'insieme di regole. Dato che ogni monitor ha una visione limitata dei suoi vicini, in molti casi può rimanere incerto riguardo la correttezza del comportamento dell'agente monitorato. Per risolvere questo problema abbiamo sviluppato un *algoritmo di consenso distribuito* che aumenta le capacità di decisione dei singoli monitor. L'efficienza della nostra soluzione è stata provata sia con approfondite simulazioni, sia con dimostrazioni teoriche e del tutto generali della convergenza dell'algoritmo di consenso.

Acknowledgments

There are a few people that I would like to thank for helping me during this work.

Prof. Antonio Bicchi, for his support and because he gave me the opportunity to work in such a stimulating environment.

Eng. Adriano Fagiolini, for having carefully supervised and supported me during my whole work.

Dr. Lucia Pallottino, for her help with mathematics and theorem formulation.

Msc. Felipe Augusto Weilemann Belo, for his clever ideas about rectangle clipping algorithms and for his great moral support.

Eng. Riccardo Schiavi, for his help with whatever I needed.

Doc. Laura Maley for her help with my doubts about english.

Doc. Marco Pellinacci, for his collaboration in the development of our Multi–Agent System Simulator and for all our precious conversations.

This thesis was entirely developed in the Interdepartmental Research Center "E. Piaggio" of the University of Pisa (*http://www.piaggio.ccii.unipi.it*).

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Multi–agent systems (MAS) are more and more often employed in various application areas to obtain secure and robust solutions. For example, collaborative agents may work together in robotic applications to explore places where human presence could be unsafe. Robot teams can cooperate to patrol a place or follow a target or even travel without collisions in an automated transport system. Moreover, in many networked systems, sensors or other autonomous devices can communicate to jointly monitor an environment variable or a network parameter.

In particular, our attention is focused on decentralized control strategies for autonomous vehicles planning their motion based only on their configuration and the ones of their visible neighbors. In such situations agents may perform different tasks, but they all must follow a cooperation protocol because a robot acting independently from the other ones could perform maneuvers that might compromise the safety of the whole system. Therefore a cooperation protocol, i.e. a set of shared decision rules, is needed to describe behaviors and interactions between agents. For example, one of the most studied collaborative tasks is the avoidance of collisions between robots. In fact, several collision avoidance strategies for MAS have been presented in the literature, with different application domains and different sets of decentralized rules [1, 2, 3, 4]. Two collision avoidance strategies will also be the "running example" to which we will apply our ideas.

As a matter of fact, whenever an agent stops following the rule set, due to spontaneous failure, tampering, or even to malicious behavior [5], agents' safety is at risk, and an automatic intrusion detection system (IDS) is needed to find agents which are not cooperating correctly. We are not interested in a centralized solution, because an omniscient monitor observing the whole system may be unrealizable for scalability and security reasons. We prefer instead a decentralized IDS which tries to identify intruders based only on locally available information and on the knowledge of the cooperation protocol.

A robot, an automated vehicle or a generic device can often be modeled by a hybrid–state system [6] because we generally need a continuous sub–state to represent physical variables like position or velocity, and a discrete one to represent maneuvers, logic variables, etc. Therefore, a set of rules contains, together with the description of a continuous control strategy, a representation of the agent's decision model, which usually can be modeled as a discrete event driven system (DES) [7]. In our example the agent's decision–making process is represented by an automaton, whose states represent agent's maneuvers, and events are expressed by mathematical conditions on logical variables and neighbors' configuration.

The literature on failure detection in DES is very rich [8, 9, 10], but while in a DES a failure is usually represented by the reaching of an undesired automaton state, in our setting failures correspond to agents arbitrarily misbehaving, therefore we can not use the same diagnosis methods used for DES. In our scenario the goal of an agent acting as a decentralized IDS is to distinguish a faulty or malicious agent in its neighborhood from a correctly cooperating one, whose action may be influenced by other agents out of the monitor visibility range. We will show that it is possible to decide if a maneuver performed by a target–agent is correct or not by estimating the events which could have stimulated it.

In many cases the observation of a single agent is not enough to obtain an unambiguous opinion on the behavior of a monitored agent because of the limits that agents may have to their sensing capability. Therefore different agents can cooperate in the detection of a single intruder because they may have a better "vision" of the target–agent's neighborhood and can make a better evaluation of the goodness of its behavior. In order to enrich our intrusion detection method we will show also a *distributed consensus* algorithm which largely improves a single monitor's detection capability.

This work is organized as follows:

In Chapter 2 we will describe the system class in which we are interested, and we will also introduce a running example that will accompany us during the whole exposition of our method.

In Chapter 3 we will expose a theoretical IDS description by which we will show how our decentralized IDS works.

In Chapter 4 we will describe more deeply an IDS implementation, with particular attention on formulas and algorithms.

In Chapter 5 we will show a different application example of our intrusion–detection method to prove its generality.

In Chapter 6 we will introduce a consensus algorithm in order to transform our decentralized method in a distributed one. We will also give a theoretical demonstration for the convergence of the algorithm.

In Appendix A we will give also some basic notes about the Multi–Agent System Simulator used to test our IDS.

Chapter 2

System Description

In this chapter we will introduce a running example to make the exposition more clear. We need a scenario where autonomous agents collaborate to achieve a common task. Each agent musts follow a decentralized policy, and its decisions must depend only on the relative positions of the neighbors that it is able to "see". The chosen example, shown in Fig. 2.1, is an automated highway where N vehicles travel together and cooperate to avoid collisions and to keep a smooth flowing traffic. Note that N is a time–variable quantity, because vehicles may enter or exit from the highway without any restriction.

Let us say that each vehicle can perform four basic maneuvers:

- FAST: the vehicle heads for the center of its current lane at its maximum speed.
- SLOW: the vehicle heads for the center of its current lane and gradually reduces its speed until it stops or it switches maneuver.
- LEFT: the vehicle performs a left turn until it reaches the bound of

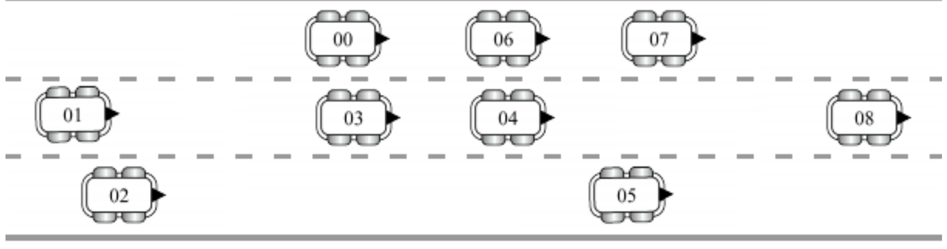


Figure 2.1: A simulated 3–lane highway with a fleet of automated vehicles.

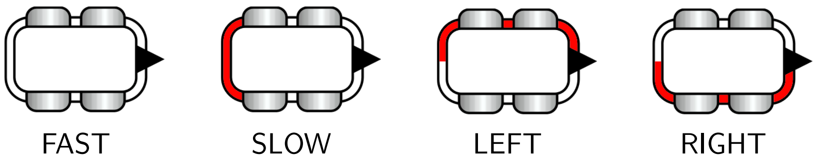


Figure 2.2: The representations of the four possible maneuvers.

the lane at its left.

- RIGHT: the vehicle performs a right turn until it reaches the bound of the lane at its right.

Moreover, let vehicles have different maximum speeds; we will indicate with V_i the maximum speed of the i-th vehicle, taking value in a defined range of allowed speeds.

A vehicle's behavior can be described by few simple rules:

- Vehicles must occupy the rightmost free lane.
- Vehicles must overtake preceding ones when the distance between the two vehicles is lower than a minimum defined value (D_f), and if the lane on their left is free.
- Vehicles must brake if their distance from a preceding vehicle is lower than D_f, and if the lane on their left is occupied.

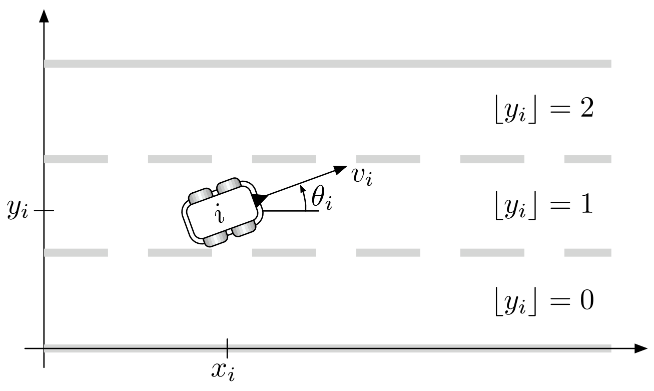
In the figures and the screenshots that will be shown in this work, we will use the images of Fig. 2.2 to represent vehicles performing a specific maneuver.

2.1 Continuous subsystem

Each vehicle can be modeled as a hybrid–state system $\mathcal{H}$, where the continuous sub–state

$$q \in \mathcal{Q} \tag{2.1}$$

represents its physical variables, and the discrete sub–state represents the currently performed maneuver. More precisely, in our example the i-th



$\lfloor y_i \rfloor = 2$
v_i
θ_i
y_i
i
$\lfloor y_i \rfloor = 1$
$\lfloor y_i \rfloor = 0$
x_i

Figure 2.3: A vehicle's continuous sub–state components.

agent's continuous sub–state is

$$q_i = \begin{bmatrix} x_i \\ y_i \\ \theta_i \\ v_i \end{bmatrix} , \tag{2.2}$$

and

$$\mathcal{Q} = \mathbb{R}^3 \times [0, 2\pi) \ . \tag{2.3}$$

Continuous sub–state components are shown in Fig. 2.3: lane 0 is represented by $y \in [0, 1)$, lane 1 is represented by $y \in [1, 2)$, etc.

The evolution of q_i is driven by the equation

$$\dot{q}_i = f(q_i, \sigma_i) \ , \tag{2.4}$$

where the discrete sub–state σ_i can assume a value representing one of the four maneuvers listed in the set

$$\Sigma = \{\mathsf{FAST}, \mathsf{SLOW}, \mathsf{LEFT}, \mathsf{RIGHT}\} \ . \tag{2.5}$$

The function f in Eq. (2.4) can be defined by describing separately the free and the forced evolution of q_i:

$$\dot{q}_i = g(q_i) + u(q_i, \sigma_i) \ , \tag{2.6}$$

where

$$g(q_i) = \begin{bmatrix} v_i \cos(\theta_i) \\ v_i \sin(\theta_i) \\ 0 \\ 0 \end{bmatrix} , \tag{2.7}$$

and

$$u(q_i,\sigma_i) = \begin{bmatrix} 0 \\ 0 \\ \omega(q_i,\sigma_i) \\ a(q_i,\sigma_i) \end{bmatrix} . \tag{2.8}$$

As shown in Eq. (2.8), control u is applied only on variables θ_i and v_i; functions ω and a depend on the value of σ_i as follows:

$$\begin{aligned}
\omega(q_i,\mathsf{FAST}) &= -\Big(y_i - \mathrm{ctr}(y_i)\Big)\,\frac{\sin(\theta_i)}{\theta_i}\,v_i - k\,v_i\,\theta_i & (2.9)\\
\omega(q_i,\mathsf{SLOW}) &= -\Big(y_i - \mathrm{ctr}(y_i)\Big)\,\frac{\sin(\theta_i)}{\theta_i}\,v_i - k\,v_i\,\theta_i & (2.10)\\
\omega(q_i,\mathsf{LEFT}) &= \begin{cases} C & \text{if } \theta_i < \theta_{max} \\ 0 & \text{otherwise} \end{cases} & (2.11)\\
\omega(q_i,\mathsf{RIGHT}) &= \begin{cases} -C & \text{if } \theta_i > -\theta_{max} \\ 0 & \text{otherwise} \end{cases} & (2.12)
\end{aligned}$$

$$\begin{aligned}
a(q_i,\mathsf{FAST}) &= \begin{cases} A & \text{if } v_i < V_i \\ 0 & \text{otherwise} \end{cases} & (2.13)\\
a(q_i,\mathsf{SLOW}) &= \begin{cases} -A & \text{if } v_i > 0 \\ 0 & \text{otherwise} \end{cases} & (2.14)\\
a(q_i,\mathsf{LEFT}) &= \begin{cases} A & \text{if } v_i < V_i \\ 0 & \text{otherwise} \end{cases} & (2.15)\\
a(q_i,\mathsf{RIGHT}) &= \begin{cases} A & \text{if } v_i < V_i \\ 0 & \text{otherwise} \end{cases} & (2.16)
\end{aligned}$$

The variables A, C, θ_{max}, and k are constant for the whole system, and the function $\mathrm{ctr}(y_i)$ of Eq. (2.9) and Eq. (2.10) returns the center of vehicle's current lane:

$$\mathrm{ctr}(y_i) \triangleq \lfloor y_i \rfloor + \frac{1}{2}L , \tag{2.17}$$

where L represents the lane width.

The control function ω of Eq. (2.9) and Eq. (2.10) is obtained from the study of the path–following problem for a kinematic model of the unicycle [11] when the desired trajectory is an horizontal line ($y = \overline{y}$). The solution, with constant v, is

$$\omega = (y - \overline{y})\,\frac{\sin(\theta)}{\theta}\,v - k\,\theta , \tag{2.18}$$

and the convergence velocity can be controlled by the constant k. In our case vehicles' speed may be non–constant, and in Eq. (2.18) we substituted the term $k\,\theta$ with $k\,v\,\theta$ in order to make the resultant trajectory almost constant in the range of the allowed velocities. An example of convergence of a vehicle's ordinate towards the center of the second lane is shown in Fig. 2.4 and Fig. 2.5.

2.2 Continuous sub–state observability

In our example we suppose that the state q of the continuous sub–system is fully observable and instantaneously measurable by other vehicles' sensors, i.e. the output of the physical layer coincides with its internal state. Of course these sensors may have a sensing range R, i.e. they cannot measure states of vehicles too far. Therefore we can define a measurable region, computed starting from the state q_i, as

$$\mathcal{Q}_i^m = \left\{ q \in \mathcal{Q} \mid d(q_i, q) < R \right\} , \tag{2.19}$$

where function d expresses the distance between tho vehicles.

Furthermore the measure of a state may be obstructed by the presence of vehicles in the observer's line of sight (LOS), therefore we have to reduce the set $\mathcal{Q}_i^m$ by considering the position of any neighboring vehicle which could limit the observer's "vision". Let us define the *observable region* as the set

$$\mathcal{Q}_i^o = \left\{ q \in \mathcal{Q}_i^m \mid M(\mathbf{Q}, q_i, q) = \mathsf{true} \right\} , \tag{2.20}$$

where

$$\mathbf{Q} = \left\{ q_1, \ldots, q_N \right\} \tag{2.21}$$

is the time–variable and length–variable set of all the continuous sub–states of the vehicles in the system. Function M returns true if there is not any state of $\mathbf{Q}$ in the LOS from q_i to q, so that agent i can measure the state of an agent lying in q, otherwise it returns false.

The *unobservable region*, i.e. the part of the state space $\mathcal{Q}$ that cannot be observed by the i-th agent, can consequently be defined as follows:

$$\mathcal{Q}_i^u = \mathcal{Q} \setminus \mathcal{Q}_i^o \; . \tag{2.22}$$

In Fig. 2.6 is shown the process by wich a simplified version of the unobservable region is computed in our Multi–Agent System Simulator (MASS).

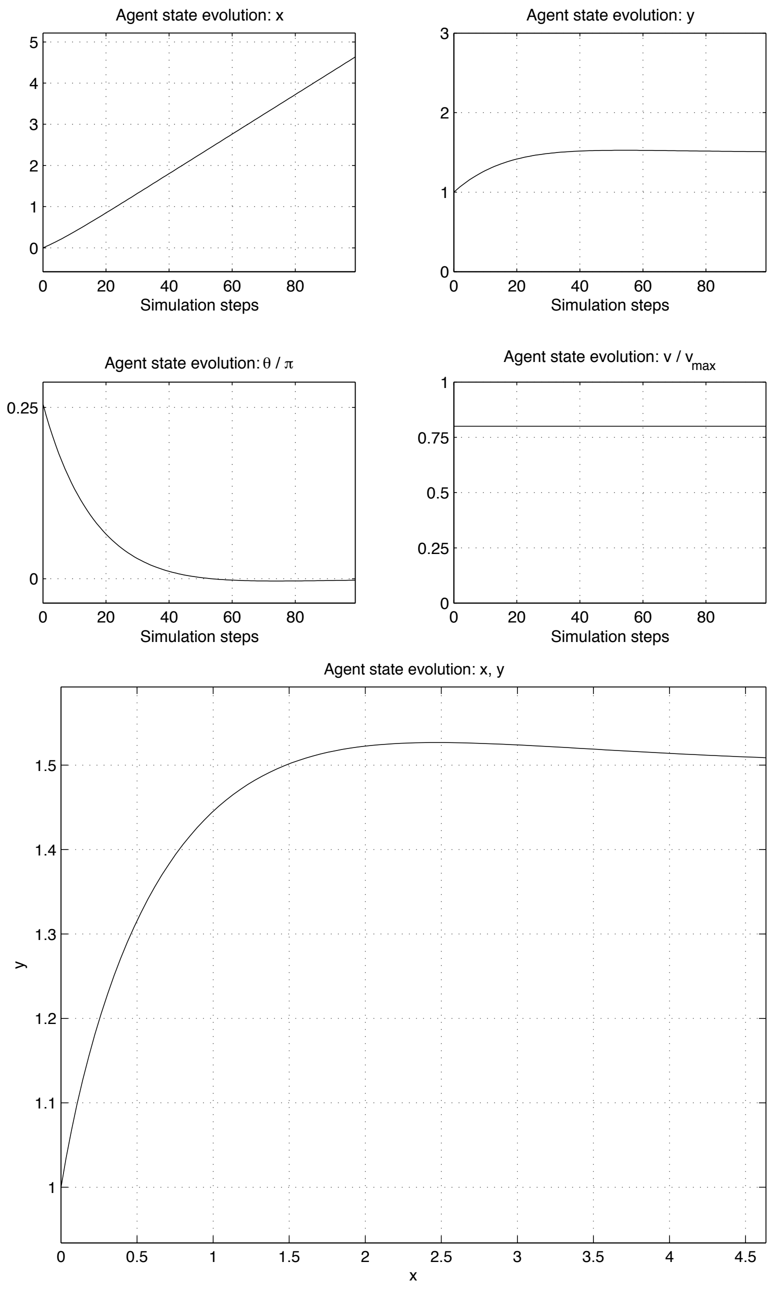


Figure 2.4: An example of a vehicle's continuous sub–state convergence to the center of the lane ($y = 1.5$) with constant speed.

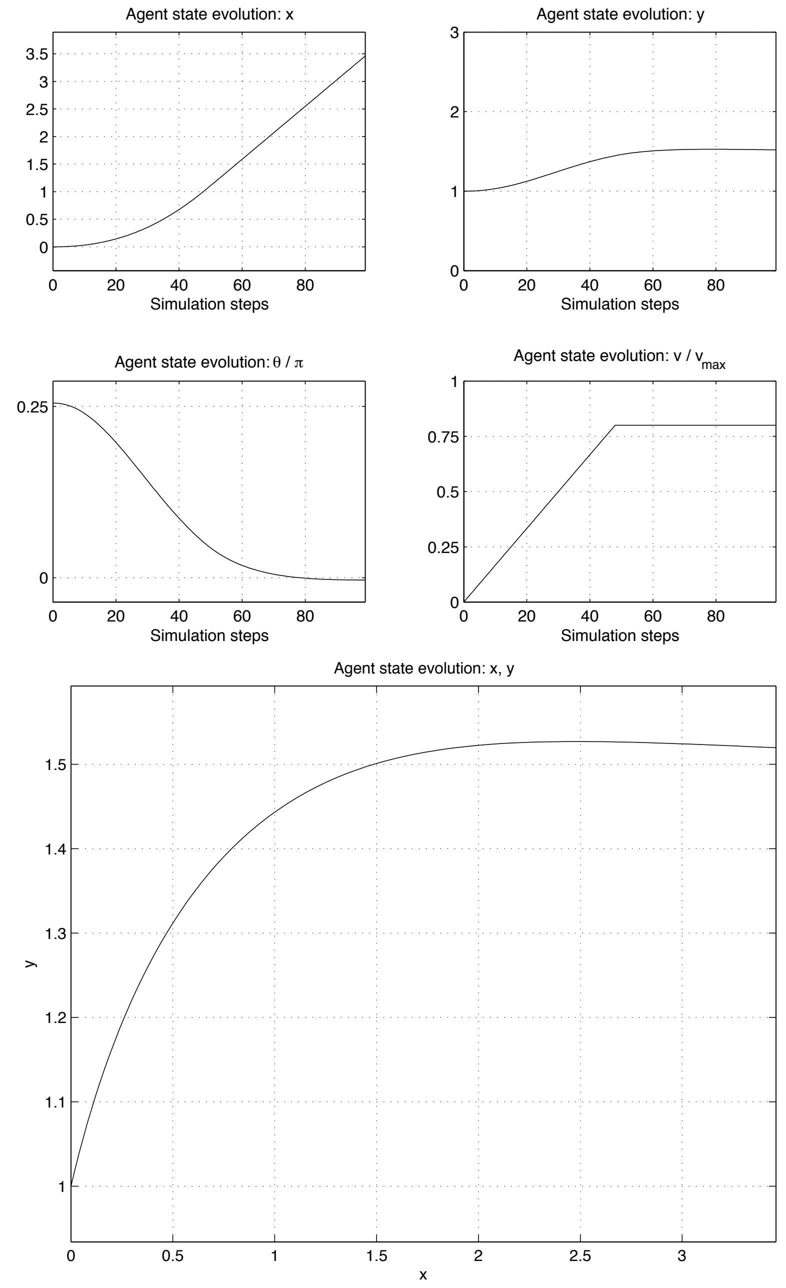


Figure 2.5: An example of a vehicle's continuous sub–state convergence to the center of the lane ($y = 1.5$) with increasing speed.

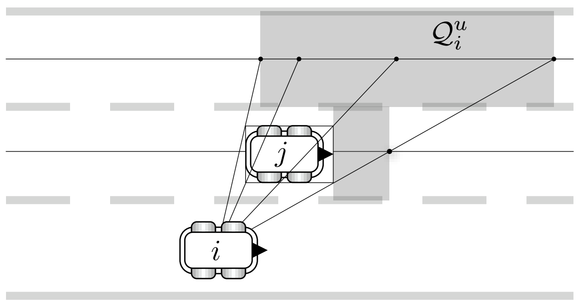


Figure 2.6: A simplified representation of an agent's unobservable region.

We consider a bounding rectangle around each vehicle in the observer's LOS and trace all the four lines joining the observer's center with the four vertices of each rectangle. The two extern points in which such lines intersect the center of each lane are assumed as the x–extrema of the hidden region.

In Fig. 2.7 we show the same simulation example of Fig. 2.1, but we evidence the subjective subdivision of the state space into the observable part $\mathcal{Q}_i^o$ and the non–observable part $\mathcal{Q}_i^u$, represented by the grey areas. The "points of view" are those of agent 01 (Fig. 2.7.a) and agent 04 (Fig. 2.7.b).

Note 2.1. *Note that in our example only the x–component and the y–component of the vector q contribute to determine if a state is observable by an agent or not, but in a more general condition this property might be not valid any more. So, the regions $\mathcal{Q}_i^o$ and $\mathcal{Q}_i^u$ have the same dimension of $\mathcal{Q}$, even if in our scenario we do not consider θ and v.*

2.3 Hybrid system inputs and outputs

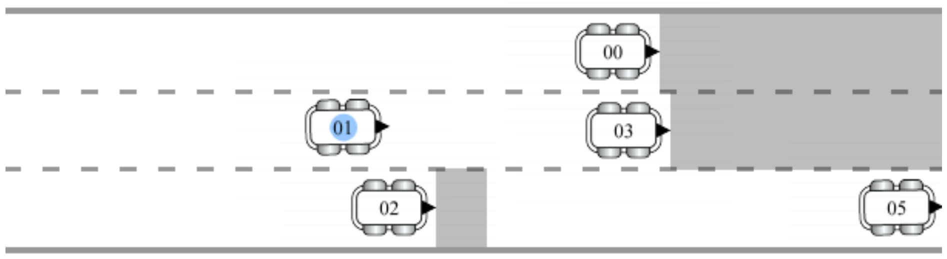
According to Eq. (2.19) and Eq. (2.20) we can define the set of observable states by the i-th agent

$$\mathcal{N}_i = \left\{ q_j \in \mathbf{Q} \mid q_j \in \mathcal{Q}_i^o \right\} , \tag{2.23}$$

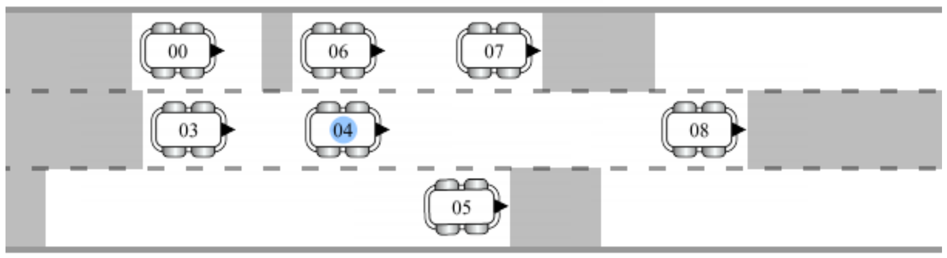
and the index set of the observable agents

$$N_i = \left\{ j \in N \mid q_j \in \mathcal{N}_i \right\} . \tag{2.24}$$

Moreover, as previously shown, the i-th vehicle can make its decisions based only on vehicles whose continuous sub–state is measurable, i.e. vehicles



(a)



(b)

Figure 2.7: Two examples of simulated unobservable regions.

whose state q is contained in $\mathcal{N}_i$. Therefore the hybrid system representing the i–th vehicle can be viewed as a single system whose continuous input and output are respectively $\mathcal{N}_i$ and q_i, as shown in Fig. 2.8.

2.4 Discrete sub–system

The vehicles' decision model is a system able to decide when a maneuver transition is needed. It musts evaluate all the rules contained in the shared collaborative policy before taking a decision and then communicate to the continuous sub–system that a change in the control function is needed. Such device can be easily modeled as a finite state machine (FSM), whose discrete states represent the maneuvers described at the begin of this chapter.

Automaton events consist in mathematical conditions on the continuous sub–state of agents' neighboring vehicles and can depend also on the automaton's internal logical variable[1]

$$\xi_i \in \Xi \; . \tag{2.25}$$

[1]More generally ξ_i could be also a vector of logical variables.

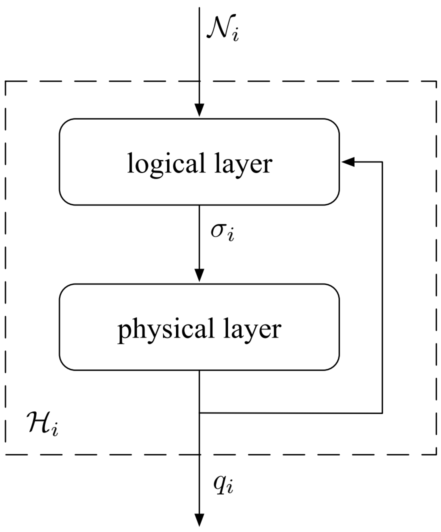


Figure 2.8: An agent's hybrid system structure with inputs and outputs.

This logical variable is used to prevent the number of discrete states from growing too much, or e.g. to represent timers as in [3]. In our example the logical variable ξ_i represents the number (0, 1, 2) of the lane where the i–th agent is intentioned to go during a LEFT or RIGHT maneuver.

A deterministic automaton, as described by Cassandras and Lafortune in [7], can be defined by the sixtuple

$$\mathcal{A} = \left\{\Sigma, \mathcal{E}, \delta, \Gamma, \sigma_0, \Sigma_m\right\} , \tag{2.26}$$

where

- Σ is the set of states.
- $\mathcal{E}$ is the set of events associated with the transitions in $\mathcal{A}$.
- $\delta : \Sigma \times \mathcal{E} \to \Sigma$ is the transition function. $\delta(\sigma_1, e) = \sigma_2$ means that, when the automaton is in the state σ_1 and the event e occurs, there will be a transition to the state σ_2.
- $\Gamma : \Sigma \to 2^{\mathcal{E}}$ is the feasible event function; $\Gamma(\sigma)$ is the set of all events e for which function $\delta(\sigma, e)$ is defined.
- σ_0 is the initial state.

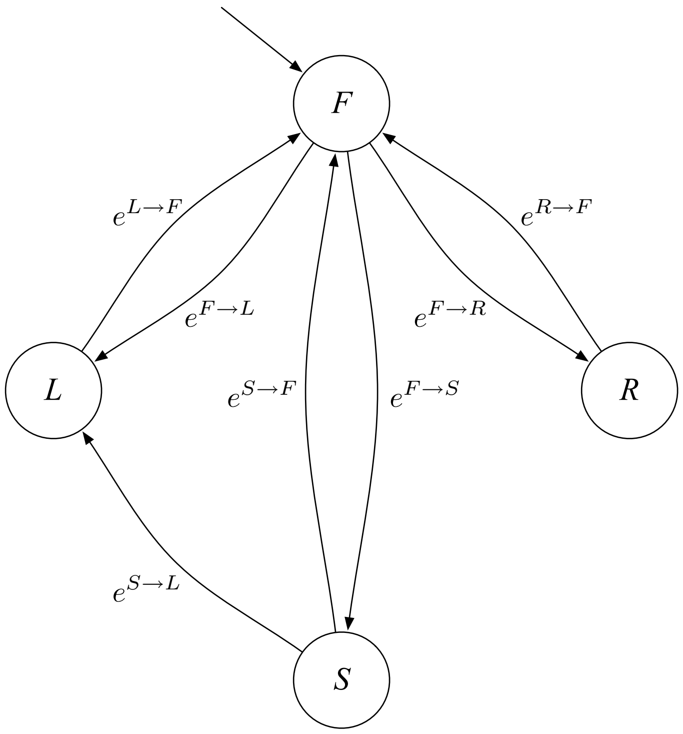


Figure 2.9: The automaton used to model the logical layer of the vehicles.

- Σ_m is a set of marked or desired states.

In our setting we do not need Σ_m because we do not have "final" or "desired" states to mark. We added instead the logical variable ξ, updated every time a transition occurs according to a reset law

$$\mathcal{R} : \mathcal{Q} \times \Sigma \to \Xi \; . \tag{2.27}$$

The automaton used to model a vehicle's logical layer is shown in Fig. 2.9, and is defined as follows:

$$\begin{aligned}
\Sigma &= \{\mathsf{FAST}, \mathsf{SLOW}, \mathsf{LEFT}, \mathsf{RIGHT}\} && (2.28)\\
\mathcal{E} &= \left\{ \begin{array}{l} e^{F\to L}, e^{F\to R}, e^{F\to S}, e^{S\to L}, \\ e^{S\to F}, e^{L\to F}, e^{R\to F} \end{array} \right\} && (2.29)\\
\sigma_0 &= \mathsf{FAST} && (2.30)
\end{aligned}$$

The functions Γ and δ can be easily deduced from Fig. 2.9, whereas the function $\mathcal{R}$ is defined as

$$\mathcal{R}(q,\sigma) = \begin{cases} \lfloor y \rfloor + 1 & \text{if } \sigma = \mathsf{LEFT} \\ \lfloor y \rfloor - 1 & \text{if } \sigma = \mathsf{RIGHT} \\ \lfloor y \rfloor & \text{otherwise} \end{cases} . \tag{2.31}$$

The event set $\mathcal{E}$ for the i-th agent[2] is defined as follows:

$$e_i^{F\to S} = \mathrm{Fd}_i \wedge \big(\mathrm{Lt}_i \vee \neg \mathrm{aln}(q_i) \vee (\lfloor y_i \rfloor = 2)\big) \tag{2.32}$$

$$e_i^{F\to L} = \mathrm{Fd}_i \wedge \neg \mathrm{Lt}_i \wedge \mathrm{aln}(q_i) \wedge (\lfloor y_i \rfloor \neq 2) \tag{2.33}$$

$$e_i^{F\to R} = \neg \mathrm{Rt}_i \wedge \mathrm{aln}(q_i) \wedge (\lfloor y_i \rfloor \neq 0) \tag{2.34}$$

$$e_i^{S\to F} = \neg \mathrm{Fd}_i \tag{2.35}$$

$$e_i^{S\to L} = \mathrm{Fd}_i \wedge \neg \mathrm{Lt}_i \wedge \mathrm{aln}(q_i) \wedge (\lfloor y_i \rfloor \neq 2) \tag{2.36}$$

$$e_i^{L\to F} = \mathrm{Lt}_i \vee (\lfloor y_i \rfloor = \xi_i) \tag{2.37}$$

$$e_i^{R\to F} = \mathrm{Rt}_i \vee \lfloor y_i \rfloor = \xi_i \tag{2.38}$$

The expressions Fd_i, Lt_i, and Rt_i, used in Eq. (2.32–2.38), are defined as follows:

$$\mathrm{Fd}_i = \big\{\exists\, q_j \in \mathcal{N}_i \mid f_F(q_i,q_j) = \mathsf{true}\big\} \tag{2.39}$$

$$\mathrm{Lt}_i = \big\{\exists\, q_j \in \mathcal{N}_i \mid f_L(q_i,q_j) \vee f_B(q_i,q_j) = \mathsf{true}\big\} \tag{2.40}$$

$$\mathrm{Rt}_i = \big\{\exists\, q_j \in \mathcal{N}_i \mid f_R(q_i,q_j) = \mathsf{true}\big\} \tag{2.41}$$

Finally we can define the functions f_F, f_L, f_B, and f_R, which represent the presence of an agent j in front, on the left, at the back, or on the right of agent i:

$$f_F(q_i,q_j) = (0 < x_j - x_i < D_f) \wedge (\lfloor y_j \rfloor = \lfloor x_i \rfloor) \tag{2.42}$$

$$f_L(q_i,q_j) = (x_i - D_b < x_j < x_i + D_f) \wedge (\lfloor y_j \rfloor = \lfloor x_i \rfloor + 1) \tag{2.43}$$

$$f_B(q_i,q_j) = (-D_b < x_j - x_i < 0) \wedge (\lfloor y_j \rfloor = \lfloor x_i \rfloor) \tag{2.44}$$

$$f_R(q_i,q_j) = (x_i - D_b < x_j < x_i + D_f) \wedge (\lfloor y_j \rfloor = \lfloor x_i \rfloor - 1) \tag{2.45}$$

The distance D_f is computed as the minimum space needed by a vehicle to get to a complete stop when running at the maximum allowed speed V_{max} in the system:

$$D_f = \frac{1}{2}\frac{V_{max}^2}{A} , \tag{2.46}$$

[2]We will use the notation e_i when the event e is expressed with respect to the i–th agent. We will use this notation also for the other functions used to define the events.

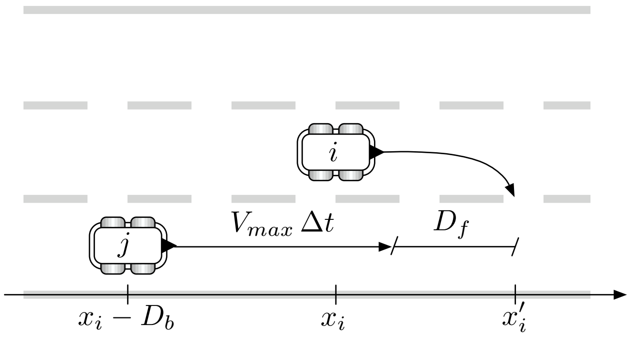


Figure 2.10: The computation of D_b.

where the constant A is the one used in Eq. (2.13–2.16). The other distance D_b is computed, as shown in Fig. 2.10, as the minimum distance of vehicle j from vehicle i such that, after vehicle i reaches lane 0, vehicle j has not to brake even if the two vehicles run at the maximum allowed speed.

The function aln of Eq. (2.32–2.38) checks if the i–th agent is aligned with the middle of the lane, and can be defined as

$$\operatorname{aln}(q_i) = \begin{cases} \mathsf{true} & \text{if } \big((y_i - \lfloor y_i \rfloor) = 0.5\big) \wedge (\theta_i = 0) \\ \mathsf{false} & \text{otherwise} \end{cases} \tag{2.47}$$

We added this check because we do not want that a vehicle crosses multiple lanes in succession. In Fig. 2.11 is shown a vehicle's trajectory while crossing two lanes, from the third one to the first one. When it reaches the second lane, its direction θ is taken back near 0 before that the vehicle steers again towards the first lane.

2.5 Continuous time and discrete time instants

As explained in [6] by J. Lygeros, dynamical systems can also be classified based on the set of times over which their state evolves:

Continuous time , when the set of times is a subset of real numbers, i.e. $t \in \mathbb{R}$.

Discrete time , when the set of times is a subset of integer numbers, i.e. $k \in \mathbb{Z}$.

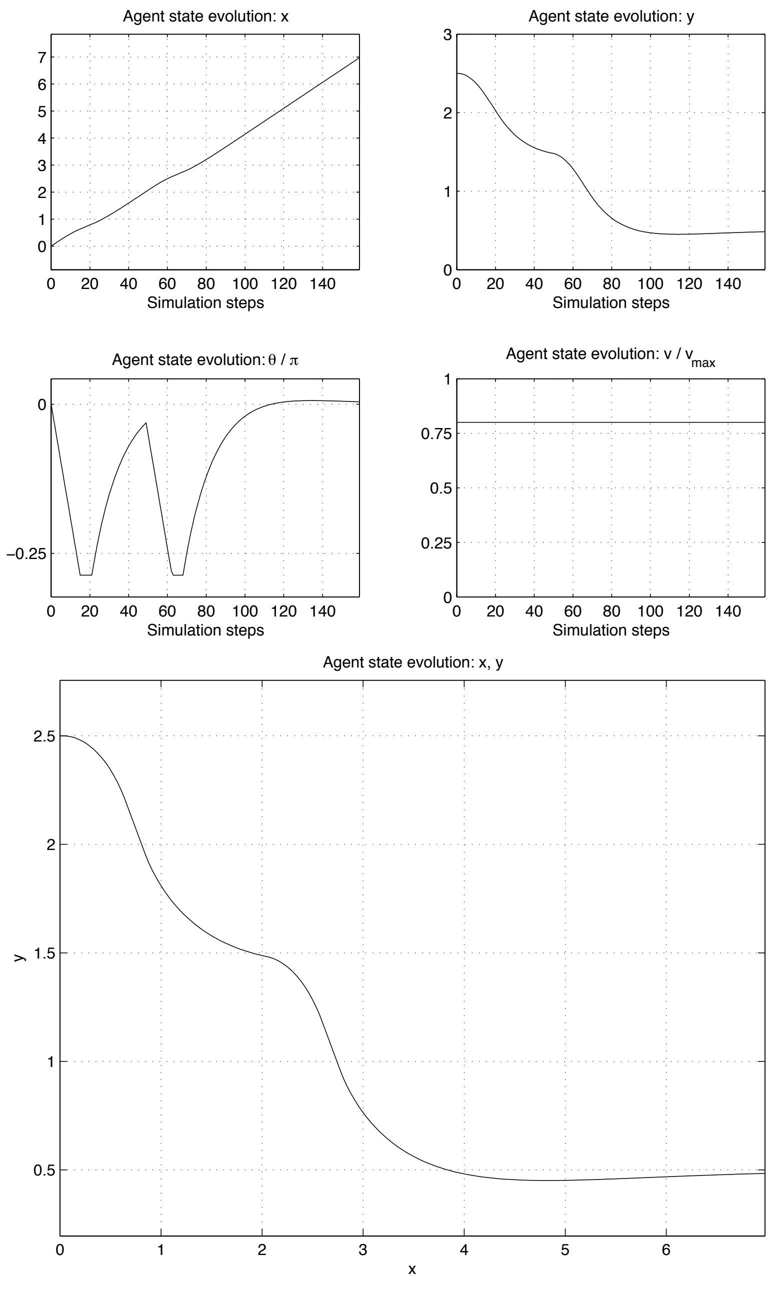


Figure 2.11: A vehicle's trajectory while crossing two lanes.

Hybrid time , when the set of times is a subset of real numbers, but there may be some particular discrete instants where ≪something of "special" happens≫.

In our system the set of times is continue but there are indeed some "special" instants where an event occurs and there may be a discontinuity in the control function.

When an event e occurs, we use the following notation to represent a discrete–state transition:

$$\sigma_i := \delta(\sigma_i, e) \ , \tag{2.48}$$

where the function δ is described at page 13.

2.6 Hybrid system dynamics

According to what we have said so far, we can now refine the representation of the hybrid system for a generic agent that we gave in Fig. 2.8, with the one shown in Fig. 2.12, where the logical layer and the physical layer are explicitly defined. The function

$$\mathcal{D} : \mathcal{Q} \times 2^{\mathcal{Q}} \times \Xi \ , \tag{2.49}$$

represents an event detector module which, based on the agent's state, its neighboring agents and automaton's internal variables, provides the occurred event to the automaton in order to make the vehicle to instantaneously change its current maneuver. We will say that $e = \mathsf{null}$ when no events are detected.

The complete dynamics of the hybrid system can finally be described as follows:

$$\dot{q} = g(q) + u(q, \sigma) \ , \tag{2.50}$$

$$e = \mathcal{D}(q, \mathcal{N}, \xi) \ , \tag{2.51}$$

and

$$\sigma := \delta(\sigma, e) \quad \text{whenever } e \in \Gamma(\sigma) \ . \tag{2.52}$$

In the last equation we assumed that if $e = \mathsf{null}$, then $e \notin \Gamma(\sigma)$, for any σ.

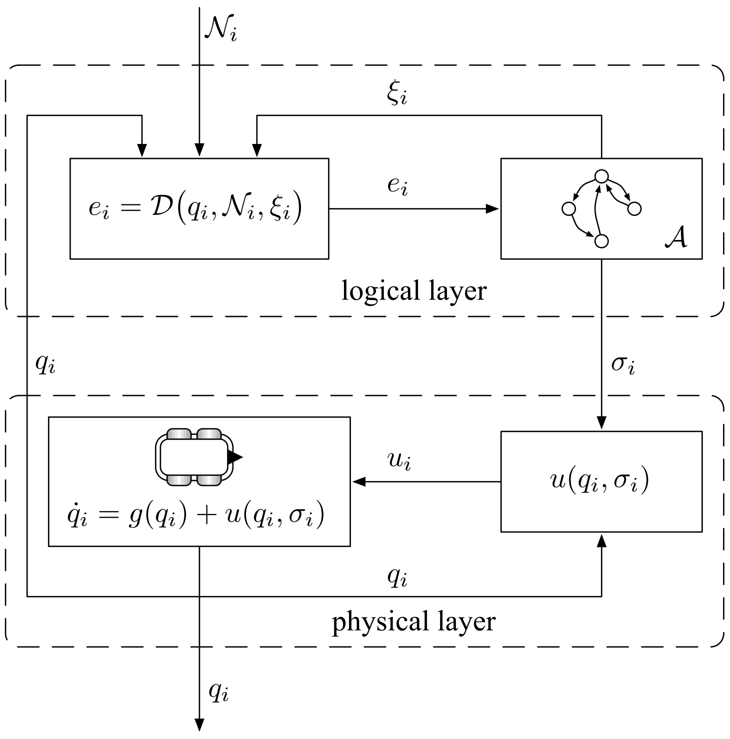


Figure 2.12: A more detailed structure of the agent hybrid system.

Chapter 3

Decentralized IDS: Theoretical Ideas

In DES world the problem of state observability is often to reconstruct the evolution of an automaton's discrete state, when the sequence of occurred events is completely or partially known [12, 13]. In these cases also the problems of control, stability, and diagnosability have been tackled in literature [14, 15, 16, 17]. Unfortunately, in our settings, the events of the automaton in the logical layer are almost never completely known, and what we have to do is instead to find a technique to estimate the automaton's internal state σ by measuring only externally available information, i.e. the vehicle's continuous sub–state q.

In the ambit of continuous systems, the state–observability problem is to reconstruct the internal state trajectory when the output of the system[1] is known. Once again, we need something different, because in our case the continuous sub–state q of the physical layer is supposed to be completely known and measurable by any observer, while we are interested in the discrete sub–state σ to understand the logical behavior of the monitored agent.

For these reasons the problem of monitoring, on which we are currently focused, is quite different from the ones found in the literature. Our goal is to develop a *synthesis technique* that allows to build a decentralized IDS for securing the considered class of multi–agent systems without the use of any form of centralization.

[1]Consider e.g. a linear system in the canonical form $\dot{x} = Ax + Bu$, where the output is $y = Cx + D$ and x is not directly measurable.

In order to show clearly the structure of the IDS, we have first to give a definition of what an intruder is:

Definition 3.1. *A* non–cooperative *agent, or* intruder*, is a faulty or malicious robot whose behavior arbitrarily deviates from the one imposed by the cooperation rules.*

From this definition it is easy to realize that a criterion to decide if a target–agent i is an intruder or not is to put ourselves in its place and to check if the measured trajectory of q_i is the same as the one that we would have chosen for ourselves.

3.1 Reputation

With the mentioned criterion every monitoring agent can assign all its neighbors with a *reputation*, i.e. a measure of their cooperativeness. The concept of reputation is normally employed in Peer–To–Peer (P2P) systems, and in Mobile Ad–hoc NETworks (MANET), where a form of cooperation is required, e.g. for establishing a message routing service that enables the communication among all agents. In these systems — see e.g. the works of Le Boudec [18, 19] —, each agent assigns its neighbors with a reputation rate that depends on whether they display a collaborative behavior or not, e.g. with respect to message forwarding.

In our scenario the reputation value with which each monitor assigns a target–agent depends on the number of feasible neighborhoods found explaining the measured target's behavior. The possible reputation levels are:

- correct (or cooperative): the monitor has a full vision of the i–th agent's neighborhood, and the measured trajectory of q_i is correct, according to that neighborhood.
- uncertain: the monitor has a partial vision of the i–th agent's neighborhood, but it exists at least a value of $\mathcal{N}_i$ explaining the measured evolution of q_i.
- faulty (or non–cooperative): the trajectory $q_i(t)$ is unexplainable according to the measured neighborhood or, if the monitor has a partial

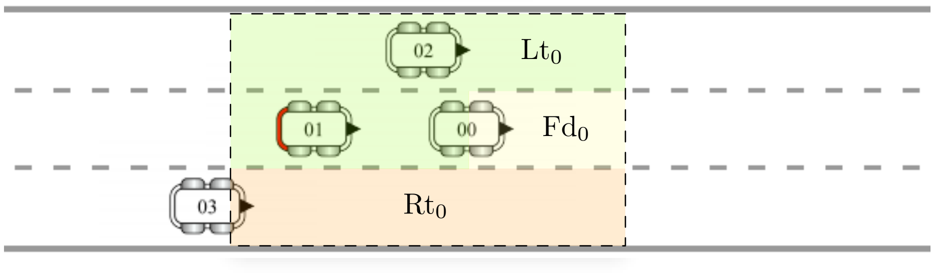


Figure 3.1: The *active region* of the agent 4, $\mathcal{Q}_4^a$.

knowledge, does not exists any possible neighborhood which could explain such trajectory.

3.2 Active region

In the previous chapter we supposed that an agent is able to measure neighboring agents' states if they lay in the measurable region. Now we need to define a subset of this region, that is the active region:

Definition 3.2. *We will refer to the i-th agent's* active region *as the set of possible states q which could affect agent i's behavior.*

In other words we say that $\mathcal{Q}_i^a$ is defined such that, if for agent j

$$q_j \notin \mathcal{Q}_i^a \ , \tag{3.1}$$

then the behavior of agent i cannot be affected by the position of agent j. Therefore, if we define

$$\mathcal{N}_i^a = \left\{ q_j \in \mathcal{N}_i \mid q_j \in \mathcal{Q}_i^a \right\} \ , \tag{3.2}$$

then the expression

$$\mathcal{D}(q_i, \mathcal{N}_i, \xi_i) = \mathcal{D}(q_i, \mathcal{N}_i^a, \xi_i) \tag{3.3}$$

(the function $\mathcal{D}$ is the one shown in Fig. 2.12) musts hold for any i and at any instant. In Fig. 3.1 an example of an agent's active region is shown.

Note 3.1. *Please note that the value of $\mathcal{Q}_i^a$ depends continuously on q_i, so we can also define a function $\mathcal{Q}^a : \mathcal{Q} \to 2^{\mathcal{Q}}$, such that*

$$\mathcal{Q}_i^a = \mathcal{Q}^a(q_i) \ . \tag{3.4}$$

3.3 Monitoring strategy

Assumed that an agent plans its trajectory based only on the elements of $\mathcal{N}^a$, whenever the i-th agent's neighborhood $\mathcal{N}_i^a$ is *completely known*, and also the state q_i can be measured, supposing that each monitor knows the inner structure of $\mathcal{H}$, it can compute the future trajectory of q_i running a simulation of a virtual vehicle with the same inputs as the real hybrid system. Then, if the real trajectory of q_i is the same as the simulated one, the target–agent is said to be *certainly cooperative.* Otherwise, if the two trajectories differs, it means that the monitored agent has a different behavior from the one imposed by the rules, and it is said to be *faulty* or *non–cooperative.*

The situation is more tricky when a monitoring robot has a *partial knowledge* of the neighborhood $\mathcal{N}_i^a$ of agent i. In facts, as described in section 2.2, supposing that monitor h wants to evaluate the behavior of agent i, often it can get only a subjective estimation of $\mathcal{N}_i^a$. We can define the observable part of $\mathcal{N}_i^a$, measured by agent h, as follows:

$$\hat{\mathcal{N}}_i^a = \left\{ q_j \in \mathcal{N}_h \mid q_j \in \mathcal{Q}^a(q_i) \right\} . \tag{3.5}$$

Since necessarily

$$\hat{\mathcal{N}}_i^a \subseteq \mathcal{N}_i^a \ , \tag{3.6}$$

monitor h has to verify if exists at least a possible value of $\mathcal{N}_i^a$ such that

1. Eq. (3.6) holds,
2. the estimated value of $\mathcal{N}_i^a$ is compliant[2] with the vision of vehicle h,
3. vehicle i's behavior is "explained".

Whenever an acceptable value of $\mathcal{N}_i^a$ exists, agent h musts say that it is uncertain about the i–th agent cooperativeness, because, since the monitor has not a complete vision of vehicle i's neighborhood, its behavior could still be correct. When no possible values of $\mathcal{N}_i^a$ are found it means that, even if the monitor has not a complete vision of the i–th agent's neighborhood, its behavior cannot be explained in any way and therefore is considered faulty.

[2]E.g. each state q such that $q \in \mathcal{N}_i^a$ and $q \notin \hat{\mathcal{N}}_i^a$ musts belong to $\mathcal{Q}_h^u$, otherwise it should have been directly measured by monitor h.

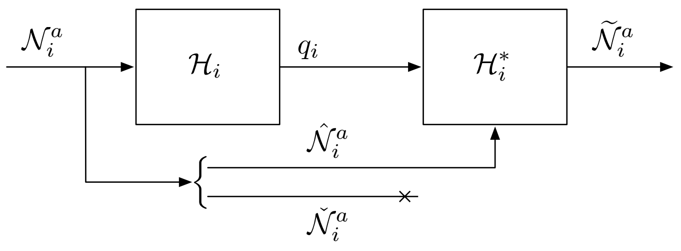


Figure 3.2: An example of an unknown–input observer module.

3.4 Unknown input observability

A possible solution to the monitoring problem could be a Unknown–Input Observer (UIO) for the vehicles' hybrid system. Let us denote with $\check{\mathcal{N}}_i^a$ the unobserved part of the set $\mathcal{N}_i^a$ defined in Eq. (3.6):

$$\check{\mathcal{N}}_i^a = \mathcal{N}_i^a \setminus \hat{\mathcal{N}}_i^a \ . \tag{3.7}$$

A UIO $\mathcal{H}_i^*$ would try to reconstruct the whole set of the values of $\mathcal{N}_i^a$ which could have led to the measured trajectory, based only on $\hat{\mathcal{N}}_i^a$. An example of a possible structure is shown in Fig. 3.2, where the set $\widetilde{\mathcal{N}}_i^a$ represents all the possible values of $\mathcal{N}_i^a$ computed by the UIO.

Unfortunately this kind of solution is not easily approachable, because of the complexity, the non–linearities, and the differential dynamics of the hybrid system $\mathcal{H}$. Furthermore, a direct approach for the computation of such a UIO leads to find ad–hoc solutions for very specific cases, and lacks of generality. Our interest is instead in a solution providing an automatic procedure to generate all the modules of the IDS, even re–using some parts of the vehicle's logical layer.

3.5 Discretization of the system

First of all we must say that a discretization of the system is needed. Suppose that a monitoring agent h is trying to evaluate the reputation of a target–agent i at $t = t_0$. If agent i detects an events at $t = t_0 + \mathrm{d}t$, it will instantaneously changes its trajectory, and if agent h does not detect the same event at the same instant, its evaluation of the target's reputation will not be correct. We can be certain that such mistakes are avoided if we suppose that all the agents take their own decisions simultaneously at determined time instants, so that each monitor is able to synchronize its event evaluation with the one of its targets.

According to this hypothesis, we can suppose that the vehicles' continuous dynamics can be discretized, and each event evaluation is made every Δt seconds. The relations of Eq. (2.50–2.52), can be redefined as follows:

$$\begin{aligned}
t_k &= t_0 + k\,\Delta t && (3.8)\\
e(t_k) &= \mathcal{D}\big(q(t_k), \mathcal{N}(t_k), \xi(t_{k-1})\big) && (3.9)\\
\sigma(t_k) &= \begin{cases} \delta\big(\sigma(t_{k-1}), e(t_k)\big) & \text{if } e(t_k) \in \Gamma\big(\sigma(t_{k-1})\big) \\ \sigma(t_k) & \text{otherwise} \end{cases} && (3.10)\\
\xi_i(t_k) &= \mathcal{R}\big(\sigma(t_k), q(t_k)\big) && (3.11)\\
q(t_{k+1}) &= q(t_k) + g\big(q(t_k)\big)\Delta t + u\big(q(t_k), \sigma(t_k)\big)\Delta t && (3.12)
\end{aligned}$$

This working hypothesis can be overcomed e.g. by supposing that, when an event occurs, it remains detectable for at least a minimum amount of time, and by introducing opportune tolerances on the evaluation of a target–agent's trajectory. But the treatment of such a subject is beyond the goals of this thesis and we hope to deal with it on future works.

3.6 Observable and unobservable events

In order to find a way to represent monitors' uncertainty about a target–agent's reputation, we have to split all the events of the automaton described in Section 2.4 into an observable and an unobservable part, based on the instantaneous configuration of the agents around the target.

Let $\mathcal{E}$ be the set of all feasible events. We can define the two sets $\mathcal{E}^o$ and $\mathcal{E}^u$ such that

$$\mathcal{E} \subseteq \mathcal{E}^o \times \mathcal{E}^u \; , \tag{3.13}$$

and a generic event $e \in \mathcal{E}$ can be represented as

$$e = e^o \wedge e^u \; . \tag{3.14}$$

The event $e^o \in \mathcal{E}^o$ represents the observable part of e, while $e^u \in \mathcal{E}^u$ represents the unobservable one. A completely observable event can be represented by

$$e = e^o \wedge \mathsf{null} \; , \tag{3.15}$$

while a completely unobservable one can be represented by

$$e = \epsilon \wedge e^u \ . \tag{3.16}$$

Therefore the two events ϵ, and null must necessarily belong respectively to the set $\mathcal{E}^o$ and to $\mathcal{E}^u$ in order to enable us to represent completely observable and unobservable events[3].

Note 3.2. *Please note that the event subdivision shown in Eq.* (3.14) *is not possible for any set of $\mathcal{E}$, $\mathcal{E}^o$, and $\mathcal{E}^u$, because in some cases the observable part of an event e is necessarily tied up by a logical* or *with the unobservable one. In these cases we can either redefine the event set $\mathcal{E}$ splitting such event into two new events, or we can decide to represent the whole event e as an unobservable one.*

3.7 Visibility projections

We can now define a suitable observation map, i.e. an operation of direct projection from $\mathcal{E}$ to $\mathcal{E}^o$, which extracts the observable part from each event. We will denote with

$$\Lambda_e : \mathcal{E} \to \mathcal{E}^o \ , \tag{3.17}$$

the function defined as

$$\Lambda_e(e^o \wedge e^u) = e^o \ . \tag{3.18}$$

Moreover, it is also possible to define an inverse observation map

$$\Lambda_e^{-1} : \mathcal{E}^o \to \mathcal{E} \ , \tag{3.19}$$

i.e. the inverse projection from $\mathcal{E}^o$ to $\mathcal{E}$ which provides, starting from an observable part $e^o \in \mathcal{E}^o$, all the events in $\mathcal{E}$ which, projected with Λ_e, could have such observable part:

$$\Lambda_e^{-1}(e^o) = \left\{ e \in \mathcal{E} \mid \Lambda_e(e) = e^o \right\} \ . \tag{3.20}$$

[3]The event ϵ is often used in the literature [7] to represent the occurrence of an unobservable event.

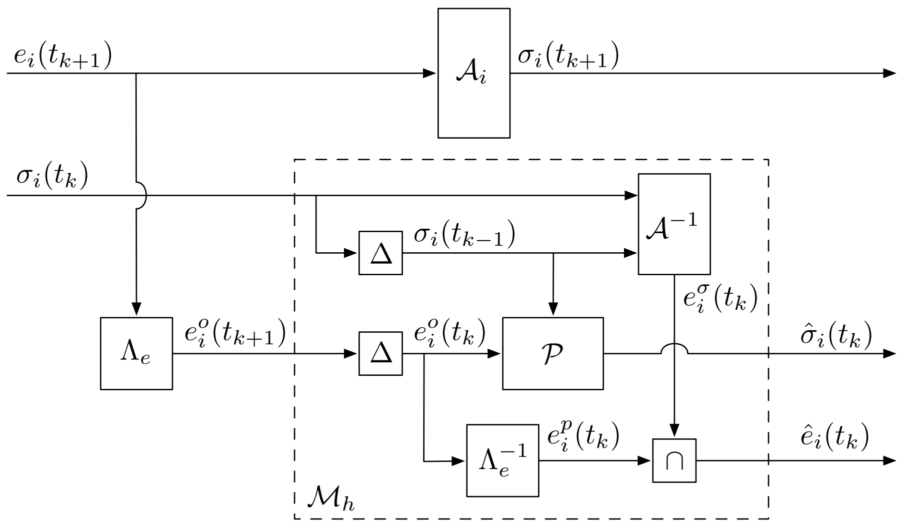


Figure 3.3: The structure of a decentralized monitor module.

3.8 Monitoring system

Let us show now the theoretical structure of the proposed monitoring system, using all the functions defined so far. Our solution is composed of three "ingredients":

- a local monitor,
- a classifier,
- a logic block.

In Fig. 3.3 is shown the block scheme of the local monitor. The automaton $\mathcal{A}_i$ on the top of the figure represents the decision–making process of the monitored agent i, which chooses its next maneuver σ_i based on the occurred event e_i. Simultaneously the monitor module, surrounded by the box in dashed line, receives the target–agent's current maneuver together with the event observable part e_i^o and provides

a. the set of events $\hat{e}_i$ compliant with such maneuver,

b. the set of maneuvers $\hat{\sigma}_i$ compliant with the event observation.

With these estimations, a target–agent's reputation will be not faulty if

$$\sigma_i(t_{k+1}) \in \hat{\sigma}_i(t_{k+1}) \; . \tag{3.21}$$

The event set $\hat{e}_i$ is important because it represents an estimation of the occurred events: since different monitors could communicate with each other, it is useful to have such estimation at monitor's disposal so that it can depute some other agents to verify its hypothesis about agent i's neighborhood.

Let us give a closer look to the monitor's structure. The occurred event e_i is an input for the system and is filtered by the projection Λ_e defined in Eq. (3.18). The so computed observable part of the event e_i reaches the second projection block Λ_e^{-1}, defined in Eq. (3.20), which provides the set e_i^p of all the feasible events according to the observed part of e_i. The other input of the local monitor is the maneuver σ_i currently performed by the agent i. This signal is an input for the "inverted" automaton $\mathcal{A}^{-1}$

$$\mathcal{A}^{-1} : \mathcal{Q} \times \mathcal{Q} \to 2^{\mathcal{E}} \ , \tag{3.22}$$

defined as follows:

$$\mathcal{A}^{-1}(\sigma_1, \sigma_2) = \big\{ e \in \mathcal{E} \mid \delta(\sigma_1, e) = \sigma_2 \big\} \ . \tag{3.23}$$

The signal e_i^σ is the set of the feasible events, according to the observed state transition $\sigma(t_k) \to \sigma(t_{k+1})$. This set, intersected with e_i^p, will provide the event set $\hat{e}_i$ which represents all the feasible events according to the observed event e_i^o and to the observed discrete–state transition.

3.9 The non–deterministic predictor

The module $\mathcal{P}$ of Fig 3.3, which is charged with forecasting the next feasible target–agent's maneuver, consists of a non–deterministic automaton that can be automatically derived from the original one depicted in Fig. 2.9. The idea is that, if monitor h cannot evaluate the whole event e, which could provoke the transition $\sigma_j \to \sigma_k$, and it observes the occurrence of e^o, then there are two possibilities:

a. the unobservable part e^u has not occurred and there will not be a discrete state transition,

b. the unobservable part e^u has occurred and the target–agent's automaton $\mathcal{A}_i$ will change its internal state.

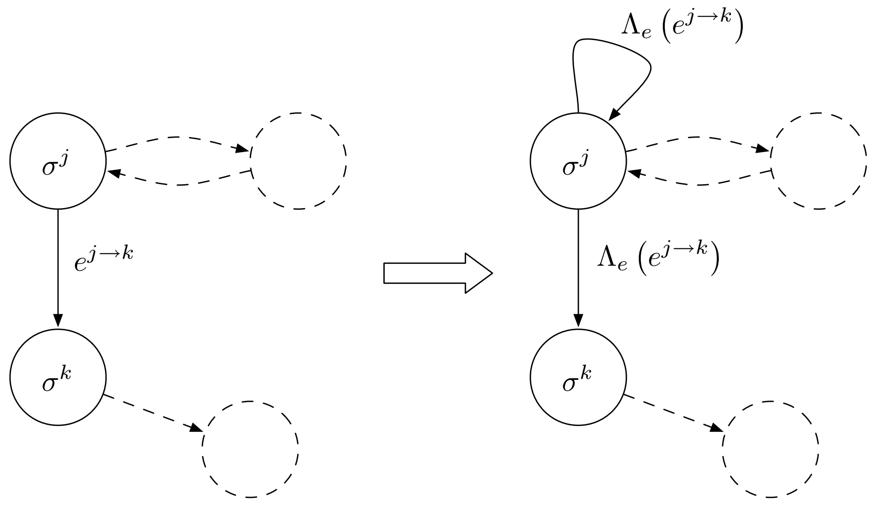

σ^j
$e^{j\to k}$
σ^k
$\Lambda_e\left(e^{j\to k}\right)$
σ^j
$\Lambda_e\left(e^{j\to k}\right)$
σ^k

Figure 3.4: A generic example of the procedure for building the non–deterministic automaton $\mathcal{P}$.

This uncertainty on the decision of the automaton $\mathcal{A}_i$, once a partially observable event has ben detected, is represented with a non–deterministic automaton which can be automatically built, starting from automaton $\mathcal{A}$, by following a simple procedure. For each state σ, consider the node in the automaton's graph representing that state: for each edge starting from there, substitute each event with its projection through the map Λ_e and add also on the node σ a self–loop assigned with the same label. A generic example of the procedure is shown in Fig. 3.4.

3.10 Estimating the number of neighbors

At this point of our exposition is useful to note that the definition of the automaton shown in the previous chapter can be re–formulated in order to make evident how events depend on the number of neighbors. For example, if we suppose that a target–agent i has not any neighboring vehicles, the automaton becomes the one of Fig. 3.5.a, where ϵ is an event whose value is always `true`, and where the value of the other events have to be consequently redefined[4] based on the fact that there is not any other vehicle than i — some events have also been disabled. Moreover, if we increase the number $\hat{n}$

[4]The new event set definition can be easily derived from Eq. (2.32–2.38) at page 15.

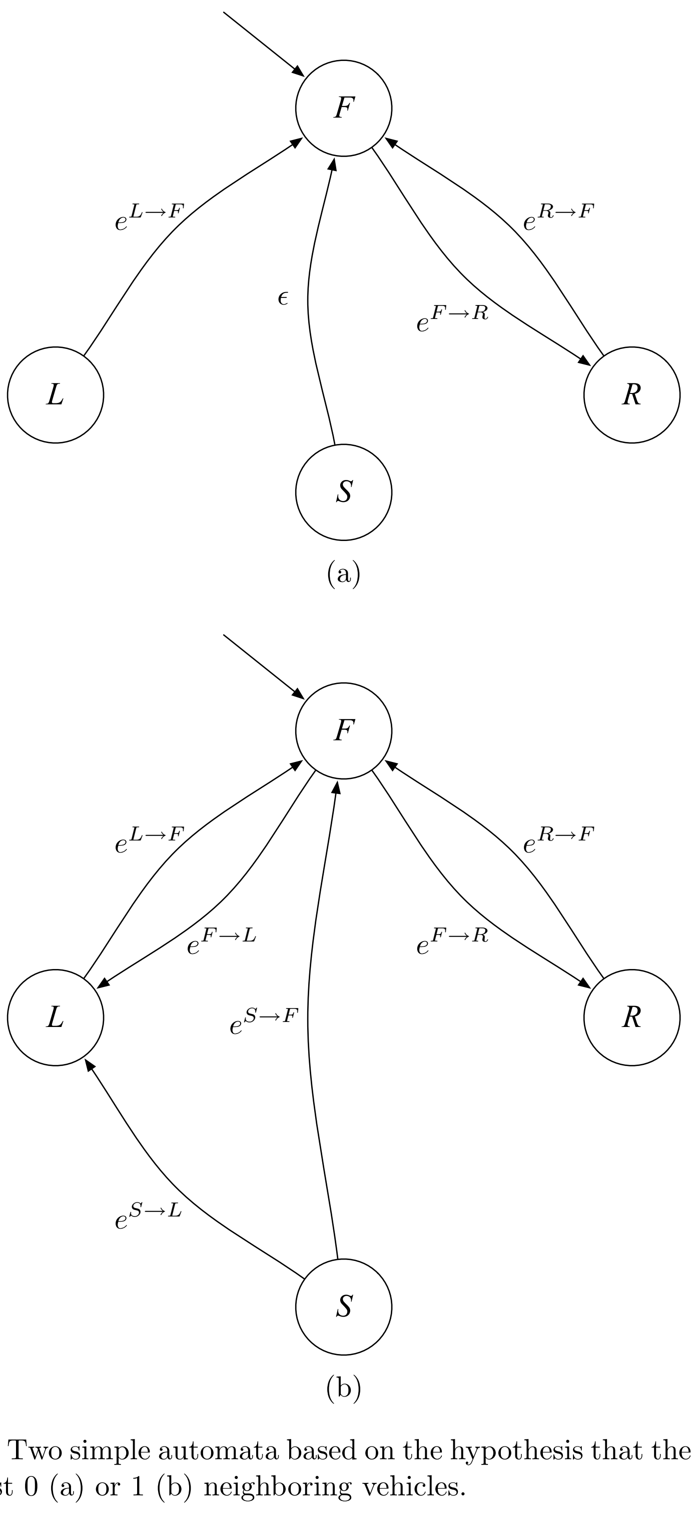


Figure 3.5: Two simple automata based on the hypothesis that the i-th agent has at most 0 (a) or 1 (b) neighboring vehicles.

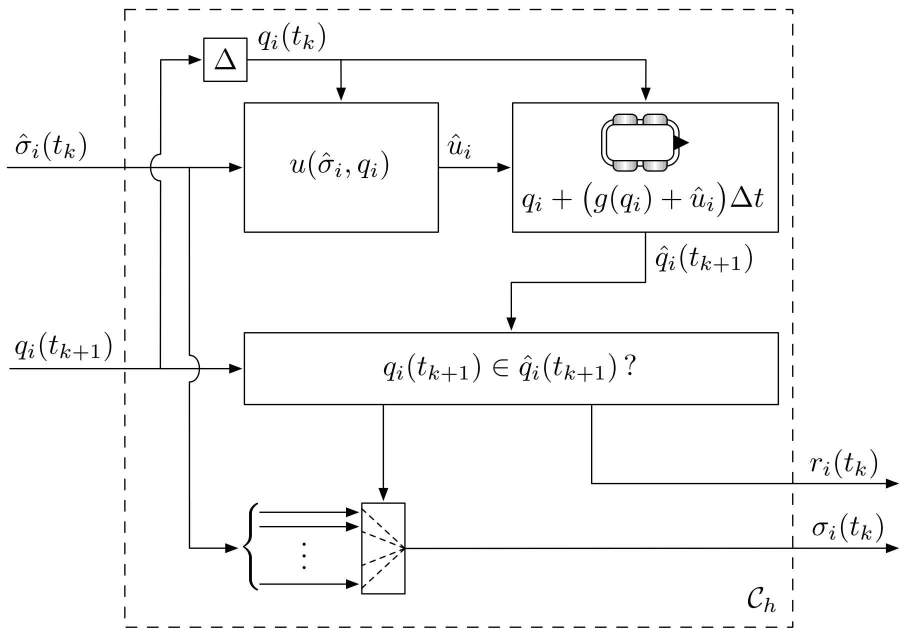


Figure 3.6: The box diagram of a classifier module.

of estimated neighbors, the automaton becomes the one of Fig. 3.5.b, and so on, until we reach the maximum allowed number of neighbors.

Because of this dependency we can imagine that the monitor module shown in Fig. 3.3 can accept another numeric input representing the number of estimated neighbors around the target–agent. Based on the value of this input, the results of the monitoring process, i.e. $\hat{\sigma}_i$ and $\hat{e}_i$, can change and it is reasonable to ask the monitor to explain target–agent's behavior starting with $\hat{n}$ equals to the number of visible neighbors, and successively increase $\hat{n}$ until an unobservable explanation is found or the maximum number of neighbors physically acceptable is reached.

3.11 The classifier module

According to what we said in Section 3.3, we need to introduce a classifier module $\mathcal{C}$ for computing all the feasible trajectories of the target–agent's continuous sub–state and deciding if the measured one is acceptable, given the observed events. Such a module can be represented by the box diagram of Fig. 3.6.

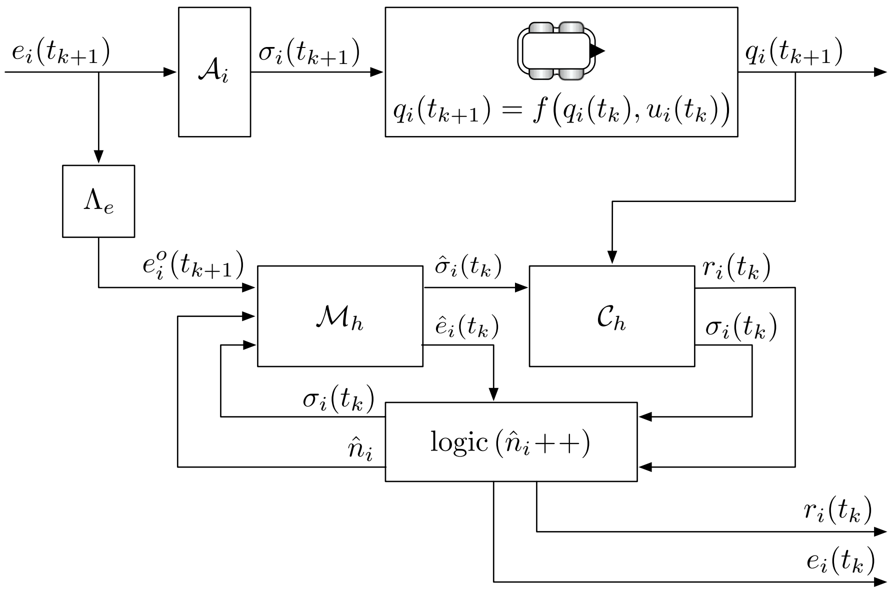


Figure 3.7: A complete scheme of the supervisor.

The continuous sub–state $q_i(t_k)$ previously measured and the set of admissible maneuvers $\hat{\sigma}_i(t_k)$ are used to compute the set $\hat{u}_i$ of all the possible continuous controls, and then the set of all the possible trajectories $\hat{q}_i(t_{k+1})$. Of course there is a bijective relation between the maneuvers in $\hat{\sigma}_i$ and the simulated evolutions in $\hat{q}_i$, therefore the classifier module can check if the current measured sub–state $q_i(t_k)$ is in the set of the simulated ones, and identify the index of the maneuver performed by the target–agent at the instant t_k.

The reputation with which the monitored agent is assigned is given by the value of the variable r_i. Unfortunately whenever the monitoring agent has not a complete vision of the target–agent's neighborhood, the output of the classifier is often uncertain.

3.12 Putting all together

It is now time to put together all the modules that we have described so far. The complete scheme of our monitoring strategy is shown in Fig. 3.7. On the top of the figure there is a part of the observed vehicle's logical and physical layer. It is the discrete "version" of the physical layer described in

Section 3.5, because we supposed the synchronization between monitor and target vehicle. On the bottom of the figure are shown the three modules of the monitor connected with each other:

The local monitor $\mathcal{M}_h$ receives as inputs the observable part of the occurred event, the last measured maneuver of the target vehicle, and the estimated number of neighbors provided by the logic block.

It provides a set of admissible maneuvers, according to the event received, and a set of possibly occurred complete events.

The classifier $\mathcal{C}_h$ receives as inputs the set of possible maneuvers and the last measured continuous sub–state $q_i(t_{k+1})$.

It provides the maneuver performed by the monitored vehicle and the value of its computed reputation, given the current inputs.

The logic module connects together and coordinates the two previous modules. It receives the performed maneuver and the "transitional" value of the monitored agent's reputation r_i. While this value is faulty, the logic module tries to raise the number $\hat{n}_i$ of estimated neighbors, until it gets a possible explanation or it reach the maximum number of neighbors physically allowed by the system.

Finally let us describe how data flow through the modules of the monitoring sytem in order to make our ideas a bit more clear about all the information paths. At the instant $k+1$ a new event occurs and its observable part reaches the local monitor module $\mathcal{M}_h$. At this point the input $\sigma_i(t_k)$ is still unknown (see Fig. 3.3) and the set $\hat{e}_i(t_k)$ is unusable. The set $\hat{\sigma}_i(t_k)$, containing all the possible maneuvers that the target–agent could have performed, reaches the classifier module $\mathcal{C}_h$. This module checks if the detected maneuver is contained in the set of the expected ones and provides a value of the target–agent's reputation. The value of r_i is passed to the logic block which, starting with $\hat{n}_i$ equal to the number of visible neighbors, while $r_i = \mathsf{faulty}$, raises the value of $\hat{n}_i$ in order to find an explanation for the target–agent's behavior. When — and if — a not faulty reputation is reached, the detected maneuver $\sigma_i(t_k)$ reaches the input of the local monitor $\mathcal{M}_i$, and allows the update of the set $e_i^{\sigma}(t_k)$ of the possibly occurred events. This set, intersected with $e_i^p(t_k)$, represents the set of all the events which may have been occurred and is forwarded to the outputs, together with the reputation value, by the logical block.

Chapter 4

Decentralized IDS: Implementation

The construction of the monitoring system presented in the previous chapter can be quite cumbersome to build, since generating both a different automaton and an event subdivision for each configuration of vehicles would require a great deal of computing effort. In this chapter we will present a more efficient solution which can be considered as an implementation of the theory that we have expounded so far.

The main goal in the implementation that we will explain is to find a way to automatically represent the monitoring agents' uncertainty and to describe a general algorithm by which it is possible to decide about a target–agent's reputation. This goal is achieved by first doing a basic evaluation of all the target–agent's events based on their definition and then by refining the evaluation based on the information about monitoring agent's observable region.

4.1 Event representation

In order to be able to compute the event subdivision into the observable and unobservable part, at any time and with any vehicle configuration, we need a particular tree representation in which all events can be defined as the logical and of particular sub–events. Not all multi–agent systems can be modeled with such an event representation, but we can be sure that if this decomposition is possible, then our general monitoring strategy is applicable.

A sub–event is a function as the ones in Eq. (2.39–2.41) at page 15. Each sub–event musts be expressed in one of the following evaluation modes:

a. ‘OR’ mode:

$$\exists\ q_j \in \mathcal{N}_i \mid b(q_i, \xi_i, q_j) = \mathsf{true} \tag{4.1}$$

b. ‘NOR’ mode:

$$\nexists\ q_j \in \mathcal{N}_i \mid b(q_i, \xi_i, q_j) = \mathsf{true} \tag{4.2}$$

c. ‘SINGLE’ mode:

$$b(q_i, \xi_i) = \mathsf{true} \tag{4.3}$$

Moreover, if we give a common domain and codomain for the “binary” functions b appearing in Eq. (4.1–4.3)

$$b : \mathcal{Q} \times \mathcal{Q} \times \Xi \to \{\mathsf{true}, \mathsf{false}\}\ , \tag{4.4}$$

and since Eq. (4.1) and Eq. (4.2) can be rewritten as

$$\left\{ \bigvee_{q \in \mathcal{N}_i} b(q_i, \xi_i, q) \right\} = \mathsf{true} \tag{4.5}$$

$$\left\{ \neg \bigvee_{q \in \mathcal{N}_i} b(q_i, \xi_i, q) \right\} = \mathsf{true}\ , \tag{4.6}$$

then any sub–event can be unequivocally characterized by the couple

$$\big\{ b, \mathrm{mode} \big\}\ , \tag{4.7}$$

where

$$\mathrm{mode} \in \big\{ \mathrm{OR}, \mathrm{NOR}, \mathrm{SINGLE} \big\}\ . \tag{4.8}$$

Note 4.1. *Please note that both functions like* ($\lfloor y_i \rfloor = \xi_i$) *and functions like the one in Eq. (2.42–2.45) at page 15 can be represented in the more general form of Eq.* (4.4)*.*

Let us define a set $\mathcal{S}$ of all the sub–events that are necessary to model the system, and let us denote with $\mathcal{S}(e)$ all the sub–events needed to represent the event e. Such an event can be evaluated as

$$e = \bigwedge_{s \in \mathcal{S}(e)} s\ . \tag{4.9}$$

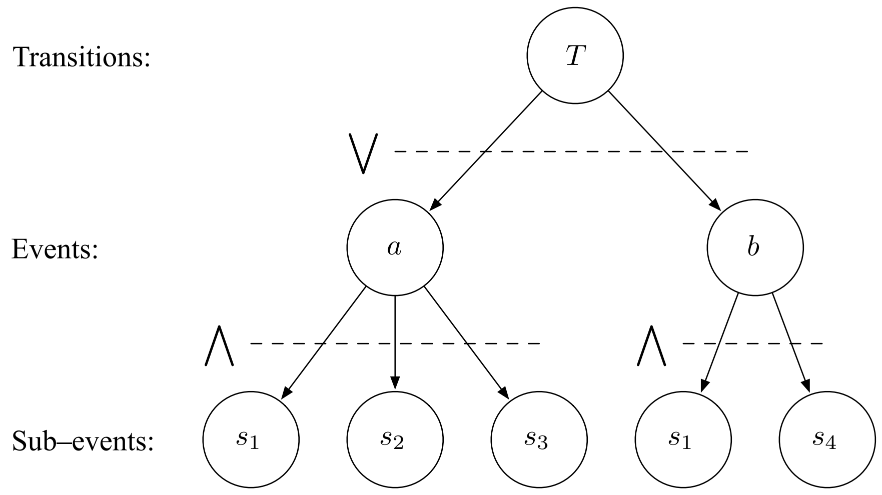


Figure 4.1: An example of the event tree representation.

Furthermore let us also denote with $\mathcal{T}$ the set of all the possible transitions in automaton $\mathcal{A}$. Each transition will be triggered by one or more events, and if we define the set $\mathcal{E}(T)$ of all the events inducing the same transition $T \in \mathcal{T}$, then

$$T = \bigvee_{e \in \mathcal{E}(T)} e \ , \tag{4.10}$$

and therefore

$$T = \bigvee_{e \in \mathcal{E}(T)} \left\{ \bigwedge_{s \in \mathcal{S}(e)} s \right\} \ . \tag{4.11}$$

Since not all events can be expressed in the form of Eq. (4.9), in some cases we may need to split or join them, but since Eq. (4.11) is the Disjunctive Normal Form (DNF) for the expression of T, given a general logical combination of sub–events, such a representation is always possible. An example of the event tree organization is shown in Fig. 4.1.

In order to avoid ambiguity in the automaton definition, it is useful to compute the *invariant event* for all its states.

Definition 4.1. *Let us define the invariant event for the state σ as follows:*

$$\mathrm{Inv}(\sigma) = e \iff \delta(\sigma, e) = \sigma \ . \tag{4.12}$$

Such an event, if it has not been explicitly described, can be computed

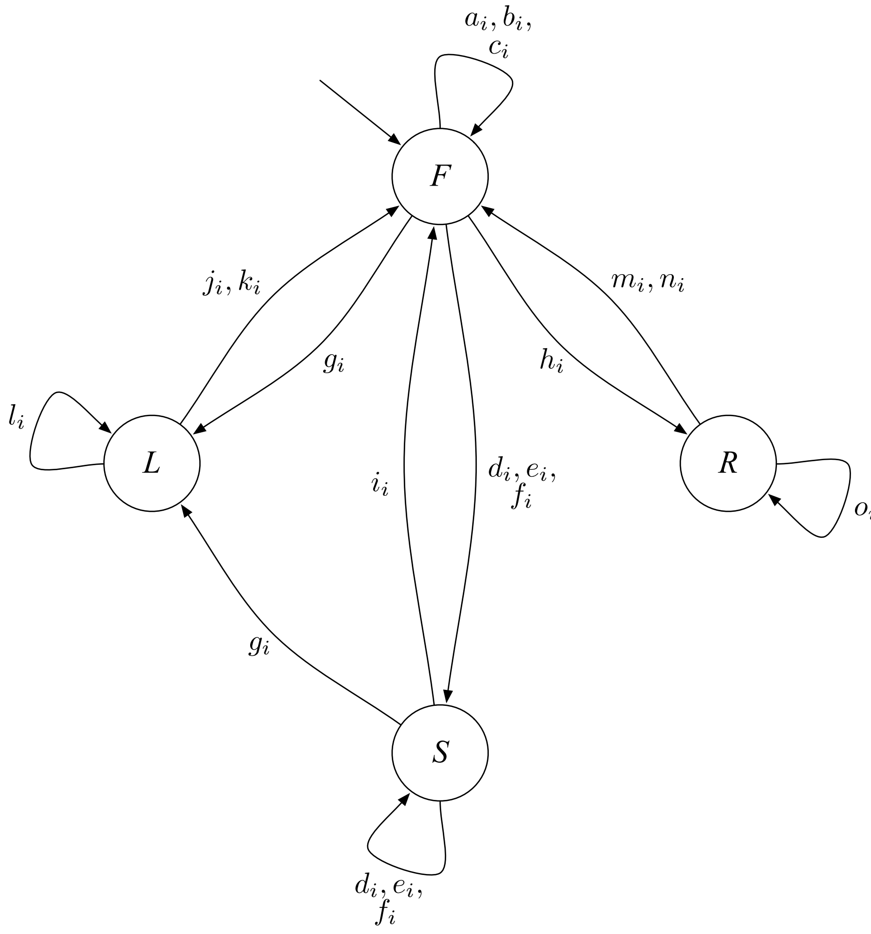


Figure 4.2: The resulted automaton for agent i after that the events were decomposed as requested.

starting from the event in $\Gamma(\sigma)$ as

$$\mathrm{Inv}(\sigma) = \neg \left\{ \bigvee_{e \in \Gamma(\sigma)} e \right\} . \tag{4.13}$$

To continue our running example, we can now transform the automaton shown in Fig. 2.9 at page 14 as in the form that we have just described: the result for the i–th agent is shown in Fig. 4.2, while in Table 4.1–4.3 all the events and sub–events are defined[1]. Please refer to Eq. (2.42–2.45) at page 15 and Eq. (2.47) at page 16 for the definitions of the functions used in

[1]Please remember that we use the notation a_i for any event or sub–event a when it is expressed with respect to the i–th agent.

Transition	Event	Sub–events
$F \to F$	a_i	$s_{i,1} \wedge s_{i,2}$
	b_i	$s_{i,1} \wedge s_{i,8}$
	c_i	$s_{i,1} \wedge s_{i,13}$
$F \to S$	d_i	$s_{i,0} \wedge s_{i,2} \wedge s_{i,12}$
	e_i	$s_{i,0} \wedge s_{i,6} \wedge s_{i,12}$
	f_i	$s_{i,0} \wedge s_{i,13}$
$F \to L$	g_i	$s_{i,0} \wedge s_{i,3} \wedge s_{i,7} \wedge s_{i,12}$
$F \to R$	h_i	$s_{i,1} \wedge s_{i,5} \wedge s_{i,9} \wedge s_{i,12}$
$S \to F$	i_i	$s_{i,1}$
$S \to S$	d_i	$s_{i,0} \wedge s_{i,2} \wedge s_{i,12}$
	e_i	$s_{i,0} \wedge s_{i,6} \wedge s_{i,12}$
	f_i	$s_{i,0} \wedge s_{i,13}$
$S \to L$	g_i	$s_{i,0} \wedge s_{i,3} \wedge s_{i,7} \wedge s_{i,12}$
$L \to F$	j_i	$s_{i,2}$
	k_i	$s_{i,10}$
$L \to L$	l_i	$s_{i,3} \wedge s_{i,11}$
$R \to F$	m_i	$s_{i,4}$
	n_i	$s_{i,10}$
$R \to R$	o_i	$s_{i,5} \wedge s_{i,11}$

Table 4.1: The redefinition of the automaton events for the i–th vehicle.

Table 4.2. The reset rule $\mathcal{R}$, by which the variable ξ_i is updated, is defined in Eq. (2.31) at page 15.

4.2 Omniscient event evaluation

Given the event definition of the previous section, let us show how an agent's *event detector* should take its decisions. First of all it has to evaluate all the sub–events of Table 4.2 and then use their value to evaluate all the feasible events. When an event is `true`, if it is not the invariant one, the event detector musts communicate to the vehicle's automaton that a maneuver switch is needed. The procedure by which the i–th agent evaluates a sub–event s is described in Algorithm 4.1.

Sub–event	Function b	mode
$s_{i,0}$	f_F	OR
$s_{i,1}$	f_F	NOR
$s_{i,2}$	$f_L \vee f_B$	OR
$s_{i,3}$	$f_L \vee f_B$	NOR
$s_{i,4}$	f_R	OR
$s_{i,5}$	f_R	NOR
$s_{i,6}$	$\lfloor y_i \rfloor = 2$	SINGLE
$s_{i,7}$	$\lfloor y_i \rfloor < 2$	SINGLE
$s_{i,8}$	$\lfloor y_i \rfloor = 0$	SINGLE
$s_{i,9}$	$\lfloor y_i \rfloor > 0$	SINGLE
$s_{i,10}$	$\lfloor y_i \rfloor = \xi_i$	SINGLE
$s_{i,11}$	$\lfloor y_i \rfloor \neq \xi_i$	SINGLE
$s_{i,12}$	aln	SINGLE
$s_{i,13}$	¬aln	SINGLE

Table 4.2: All the sub–events for the i–th vehicle as they are defined in the MASS simulator.

Sub–event	Meaning
$s_{i,0}$	Someone is in front of agent i
$s_{i,1}$	Nobody is in front of agent i
$s_{i,2}$	Someone is on the left of agent i
$s_{i,3}$	Nobody is on the left of agent i
$s_{i,4}$	Someone is on the right of agent i
$s_{i,5}$	Nobody is on the right of agent i
$s_{i,6}$	Agent i is on the maximum lane
$s_{i,7}$	Agent i is not on the maximum lane
$s_{i,8}$	Agent i is on the minimum lane
$s_{i,9}$	Agent i is not on the minimum lane
$s_{i,10}$	Agent i is on its target lane
$s_{i,11}$	Agent i is not on its target lane
$s_{i,12}$	Agent i is aligned with the center of the lane
$s_{i,13}$	Agent i is not aligned with the center of the lane

Table 4.3: The “meaning” of all the sub–events defined in Table 4.2.

Algorithm 4.1 Omniscient evaluation of the sub–event s_i.

Inputs: $s = \{b, \text{mode}\}$, q_i, ξ_i, $\mathcal{N}_i$
Outputs: the value of s

```
1: if mode = SINGLE then
2:     return $b(q_i, \xi_i)$                          ◁ this is the simplest case
3: end if
4: value ← false                                       ◁ we start with false
5: for all $q \in \mathcal{N}_i^a$ do
6:     if $b(q_i, \xi_i, q)$ = true then
7:         value ← true                                ◁ at least one state $q$ verifies the condition
8:         break the loop                              ◁ therefore we can exit from the loop
9:     end if
10: end for
11: if mode = NOR then
12:     value ← ¬(value)                               ◁ NOR means not OR
13: end if
14: return value
```

4.3 Estimation of ξ

In section 2.2, we supposed that the continuous sub–state of a target vehicle i is completely measurable by each monitoring agent which is able to "see" i. We did not make any hypothesis on ξ_i, and that is because this variable is stored inside the target–agent's automaton and cannot be measured in any way. However, it can be estimated exploiting the fact that the reset law $\mathcal{R}$ is known to all the monitors and the state q_i is measurable. Therefore, whenever monitor h detects a maneuver transition of vehicle i, it can compute $\mathcal{R}(q_i, \sigma_i)$ and update its estimation of the target–agent's variable ξ at the same instant in which the target–agent updates its own value. We will say that a monitoring agent h has "locked" a target–agent i when there has been a maneuver transition for agent i, and its variable ξ_i has been estimated.

4.4 3–level logic

In order to represent the uncertainty of a vehicle monitoring another one, we use a 3–level logic in which event and sub–event values can be

- true,
- uncertain,
- false.

We can also redefine the logical operators $\wedge$, $\vee$, and $\neg$ in order to make them to work also on this value set. The truth tables of these operators are shown in Tables 4.4–4.6.

4.5 Non–omniscient event evaluation

With these new value set and the redefined operators we can now model the decision–making process of a monitor h which has to evaluate an event for a target vehicle i, whose neighborhood is not completely known. This evaluation is based only on the definition of the events and will be subsequently refined using the information about monitoring agent's visibility.

Since the evaluation mode of a sub–event s can be OR, NOR, and SINGLE, the uncertainty of the monitor depends also on this value:

- if the evaluation mode of s is SINGLE, then, since both the target–agent's state q_i and its logic variable ξ_i are known[2], then the sub–event is completely computable by the monitor.
- If the evaluation mode of s is OR, it means that monitor h has to compute the binary function b associated with s for all the visible agents' state $q_j \in \mathcal{N}_i$. But monitor h cannot know precisely the value of agent i's neighborhood $\mathcal{N}_i$, it can only get an estimated value $\hat{\mathcal{N}}_{h,i}$, as shown in Eq. (3.5) at page 23. Therefore, even after computing the value of b for each $q_j \in \hat{\mathcal{N}}_{h,i}$, in some cases it cannot be certain of the value of the sub–event s_i. Such estimated value will be

  a. true if it exists at least an agent's state q for which the function b is true,

  b. uncertain otherwise, because the monitor cannot exclude that a hidden agent's state q, such that $q \in \mathcal{N}_i$ but $q \notin \mathcal{N}_h$, may verify the condition $b =$ true.

[2]Let us suppose that monitor h has locked its target.

a	b	$a \wedge b$
true	true	true
true	false	false
true	uncertain	uncertain
false	false	false
false	uncertain	false
uncertain	uncertain	uncertain

Table 4.4: The truth table of the logical operator $\wedge$ over the new domain.

a	b	$a \vee b$
true	true	true
true	false	true
true	uncertain	true
false	false	false
false	uncertain	uncertain
uncertain	uncertain	uncertain

Table 4.5: The truth table of the logical operator $\vee$ over the new domain.

a	$\neg a$
true	false
false	true
uncertain	uncertain

Table 4.6: The truth table of the logical operator $\neg$ over the new domain.

- If the evaluation mode is NOR, the monitor's behavior is the same as in the previous case, but the final values are different. The estimated value of s will be

    a. false if it exists at least an agent's state q for which the function b is true,

    b. uncertain otherwise.

The non–omniscient evaluation of a sub–event s_i by monitor h is shown in Algorithm 4.2.

Algorithm 4.2 Non–omniscient evaluation of the sub–event s_i.
Inputs: $s = \{b, \text{mode}\}$, q_i, ξ_i, $\mathcal{N}_h$
Outputs: the estimated value of s_i

```
1: if $\text{mode} = \text{SINGLE}$ then
2:     return $b(q_i, \xi_i)$                          ◁ this is the simplest case
3: end if
4: $\text{active} \leftarrow \mathcal{Q}^a(q_i)$        ◁ compute target–agent's active region
5: $\text{value} \leftarrow$ uncertain                  ◁ we start with an uncertain value
6: for all $q \in \mathcal{N}_h$ do
7:     if $q_i \notin \text{active}$ then
8:         continue with next $q$               ◁ consider only states inside the active region
9:     end if
10:    if $b(q_i, \xi_i, q) =$ true then
11:        $\text{value} \leftarrow$ true                ◁ at least one state $q$ verifies the condition
12:        break the loop                       ◁ therefore we can exit from the loop
13:    end if
14: end for
15: if $\text{mode} = \text{NOR}$ then
16:    $\text{value} \leftarrow \neg(\text{value})$       ◁ NOR means not OR
17: end if
18: return value
```

The evaluating strategy we have expounded so far is based only on the 'mode' attribute of the sub–events and does not take into consideration the monitor's observable region the mutual configuration of target and monitor. In the next sections, we will expose a method for refining these non–omniscient monitor's predictions using the information about *observability*.

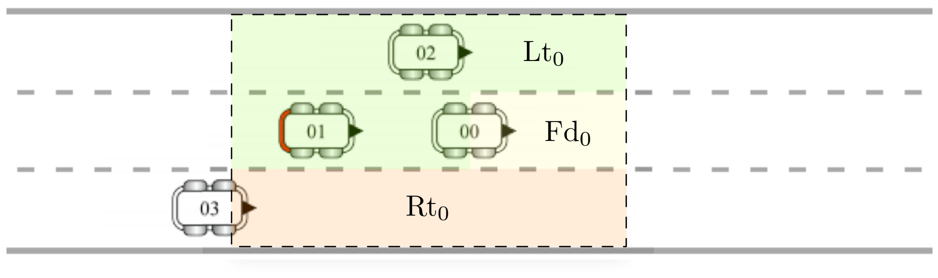


Figure 4.3: An application example of the indicator function Υ.

4.6 Indicator function

Let us define a function Υ which applies to all the binary functions as the ones of Eq. (4.4). The set $\mathcal{B}$ of all this kind of functions can be denoted as

$$\mathcal{B} = (\mathcal{Q} \times \mathcal{Q} \times \Xi)^{\{\mathsf{true},\mathsf{false}\}} , \tag{4.14}$$

then the function

$$\Upsilon : \mathcal{Q} \times \mathcal{B} \to 2^{\mathcal{Q}} \tag{4.15}$$

can be defined as follows:

$$\Upsilon(q_i, \xi_i, b) = \left\{ q \in \mathcal{Q} \mid b(q_i, \xi_i, q) = \mathsf{true} \right\} , \tag{4.16}$$

where $b \in \mathcal{B}$. An application example of the indicator function Υ is shown in Fig. 4.3, from which it is also easy to understand how the active region $\mathcal{Q}^a$ can be computed as the union of all the values of $\Upsilon(q, \xi, b)$ for any sub–event and for any value of ξ.

Note 4.2. *Please note that in general the region $\Upsilon(q_i, b)$ has the same dimensions as the state space $\mathcal{Q}$, even if on the shown figures is represented as a 2–dimensional region.*

4.7 Prediction refinement

Let us suppose that monitor h computed the set $\hat{\sigma}_i$ of the possible maneuvers that the target–agent i can perform according to the estimated events. It computed the set $\hat{e}_i$ of all possible events which could trigger that maneuver transitions. This set can be further refined by making hypothesis about the target–agent's neighborhood and then doing a sort of compatibility verification with the monitor's current vision. In particular, since the function Υ has been defined, the monitor can use it to make some hypothesis about the presence or the absence — or in general about the value of q_j — of other hidden vehicles around the target–agent.

Whenever an uncertain transition is detected, the monitor can visit the event–representation tree and control which events of $\mathcal{E}(T)$ have an uncertain value and which ones are false[3]. For each uncertain event e, there will be a set of true sub–events in $\mathcal{S}(e)$ and a set of uncertain ones[4]. Now, for each uncertain sub–event s, the monitor can pose the question: "what should be the value of a hidden vehicle's state q_j in order to make the sub–event s true?". The *indicator function* can provide an answer: given a sub–event $s = \{q, \text{mode}\}$, where mode is e.g. OR, and given a target–agent i, the values q which make the sub–event true are the ones in the set[5]

$$\hat{q}_j = \Upsilon(q_i, \xi_i, b) \ . \tag{4.17}$$

Now let us consider the observable and unobservable regions $\mathcal{Q}_h^o$ and $\mathcal{Q}_h^u$, defined in Eq. (2.20–2.22) at page 8: given the sub–event s and these regions, the values of q_j inside the set

$$\hat{q}_j^o = \hat{q}_j \cap \mathcal{Q}_h^o \tag{4.18}$$

are not possible, because the monitoring agent h is able to see all the vehicles whose state is in $\mathcal{Q}_h^o$. Therefore if it had existed a vehicle j such that $q_j \in \hat{q}_j^o$, then the sub–event s should have been evaluated as true by monitor h instead of uncertain. Then we can conclude that the only possible values for a hypothetical state q_j are the ones in the set

$$\hat{q}_j^u = \hat{q}_j \setminus \mathcal{Q}_h^o \ . \tag{4.19}$$

Whenever the evaluation mode is NOR, the line of reasoning is quite similar: according with what we previously said, $\hat{q}_j$ now represents the region where a hidden vehicle *should not be* so that the sub–event s can be true. But the

[3]If one of them was true, then the transition T should be true.
[4]If a sub–event was false, then the whole event should be false.
[5]Please note that the set $\hat{q}_j$ is necessary a subset of $\mathcal{Q}_i^a$.

monitoring agent h is able to measure all the vehicles inside $\mathcal{Q}_h^o$, and if it had existed a vehicle j such that $q_j \in \hat{q}_j^o$, then the sub–event s should have been false instead of uncertain. Therefore the interested region is $\hat{q}_j^u$, i.e. the monitoring agent makes hypothesis only about its unobservable region.

Now let us consider the case in which, after computing $\hat{q}_j^u$, this quantity is an empty set. If $\hat{q}_j^u = \emptyset$ it means that there is not any hidden position in which a vehicle's state q_j could lay in order to make the condition b to be true. Therefore if the evaluation mode of s is OR, then the real value of the sub–event is false, whereas if the evaluation mode of s is NOR, then the real value of the sub–event is true. The whole procedure provides a refined value of all uncertain events before the detection of the next target–agent's maneuver and can be summed up in Algorithm. 4.3.

Algorithm 4.3 Prediction refinement for the uncertain events e_i.

Inputs: e, q_i, ξ_i, $\mathcal{Q}_h^o$
Outputs: the refined estimated value of e_i

```
1:  for all $s \in \mathcal{S}(e)$ do
2:      if valueof($s_i$) = true then
3:          continue with next $s$                  ◁ refinement is only for uncertain values
4:      end if
5:      $\hat{q}[s] \leftarrow \Upsilon(q_i, \xi_i, b)$        ◁ compute all the possible values of hidden states
6:      $\hat{q}[s] \leftarrow \hat{q}[s] \setminus \mathcal{Q}_h^o$          ◁ refine the possible values based on $\mathcal{Q}_h^o$
7:      if $\hat{q}[s] = \emptyset$ then
8:          if mode = OR then
9:              valueof($s_i$) ← false                ◁ the value of $s_i$ is false
10:             return false                         ◁ the value of $e_i$ becomes immediatly false
11:         else
12:             valueof($s_i$) ← true                 ◁ the value of $s_i$ is true, then also
13:         end if                                     the value of $e_i$ might change
14:     end if
15: end for
16: return valueof($e_i$)                            ◁ the updated value of $e_i$
```

4.8 Hypothesis formulation

In the previous sections we said that whenever a monitor remains uncertain about a target–agent's reputation, it can formulate some hypothesis on the eventual presence or absence of other vehicles around target, which the monitor is not able to see. These hypothesis consist in an set of values that a vehicle's state q_j could assume in order to verify the conditions defined in the set of possibly occurred events and sub–events. These values are computed in Algorithm 4.3, where we denoted with $\hat{q}[\,s\,]$ the hypothesis about sub–event s. Of course such hypothesis types depend also on the 'mode' attribute of s:

- If mode = OR, the hypothesis could be enunciated as "I suppose that there is a vehicle j such that $q_j \in \hat{q}$".
- If mode = NOR, the hypothesis could be enunciated as "I suppose that there is not a vehicle j such that $q_j \in \hat{q}$".

Hypothesis formulation represents the farthest goal that a single monitoring agent is able to accomplish on its own. In the next chapters, after giving some simulation results for the monitoring strategy, we will illustrate a *consensus algorithm*, by which many monitors can communicate with each other and reach a shared common decision about a target–agent's reputation.

4.9 Reputation assignment

Once that all the sub–events in $\mathcal{S}$ have been estimated for the target–agent i, the monitor can evaluate also all the events, based on their definition of Table 4.1 and the operators of Tables 4.4–4.6: the value of an event will be the logical and of all its sub–events, whereas the value of a transition will be the logical or of all its events. At the end of this procedure, monitor h has a complete estimation of the values of the target–agent's events. If at least one of the transitions has a true value, the monitor has a precise idea of which should be the target–agent's future behavior, otherwise if one or more transitions have an uncertain value, then monitor h can expect that agent

i will perform a limited set of maneuvers, i.e. the maneuvers corresponding to the feasible transitions (see Algorithm 4.4).

Algorithm 4.4 Non–omniscient evaluation of the i–th agent's possible future maneuvers.

Inputs: σ_i, q_i, ξ_i, $\mathcal{N}_h$
Outputs: all the possible next maneuvers

```
1: maneuvers ← $\emptyset$                          ◁ an initial empty set
2: for all $T \in \mathcal{T}$ do
3:    $\sigma^+ \leftarrow T(\sigma_i)$                 ◁ get the destination state of $T$
4:    if valueof($T_i$) = true then
5:       return {$\sigma^+$}                ◁ $T$ is the only certain transition
6:    else if valueof($T_i$) = uncertain then
7:       maneuvers ← maneuvers ∪ {$\sigma^+$}      ◁ $T$ is a possible transition
8:    end if
9: end for
10: return maneuvers
```

After that monitor h computed the set $\hat{\sigma}_i$ of all possible maneuvers that agent i can perform according to the rule set, it has to check if the next detected maneuver is inside $\hat{\sigma}$ and decide for agent i's reputation. Two cases can present themselves:

1. The set of admissible maneuvers is composed of only a value σ^*, so if the detected maneuver is equal to σ^*, then the target's reputation is correct, otherwise is faulty.

2. The set of admissible maneuvers is made of more than one value, so if the detected maneuver is not among them, then the target's reputation is faulty, otherwise it is uncertain, because even if the monitor has estimated a set of "good" maneuvers, it cannot know with certainty which is the correct one.

4.10 Simulation results

In this section we will show some simulation results in order to prove the effectiveness of our method. In the following figures each vehicle has its identification number printed on its top, and the monitoring vehicle is marked by a light blue circle. The target–agent is marked

a. with a *green circle* when the monitor is *certain* that the target–agent's behavior is *cooperative*,

b. with a *yellow circle* when the monitor is *uncertain* about the target–agent's behavior,

c. with a *red circle* when the monitor is *certain* that the target–agent's behavior is *uncooperative.*

We represented also with red and green rectangles the regions in which a monitoring agent estimates respectively the presence or the absence of another vehicle.

Note 4.3. *Please note that, although we can only show 2–dimensional regions with our graphical representation, all the monitoring vehicle's hypothesis are generally about the whole 4–dimensional state. The fact that in our example only x–component and y–component contribute in the computation of the events is only for chance.*

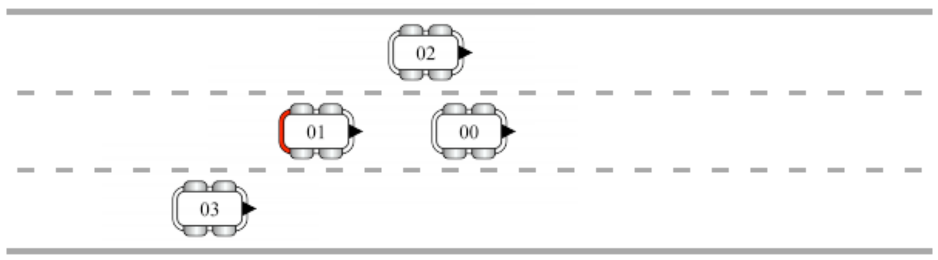
In Fig. 4.4.a is shown a frame of a simulated video with 3 monitors and a target–agent, while in Fig. 4.5 are shown the same frames of 4.4 in which has been highlighted the target–agent's active region.

Agent 0's behavior is faulty because it should return on the first lane instead of running on the second one, since there is not any vehicle on its right. Agents 1–3 are monitoring agent 0 without communicating with each other. The simulation results are shown in Fig. 4.4.b, 4.4.c, and 4.4.d, representing respectively the "point of view" of agent 1, 2, and 3. It is easy to note that only agent 3 is able to detect the agent 0's faulty behavior, while agents 1 and 2 remain uncertain about the agent 0's reputation[6].

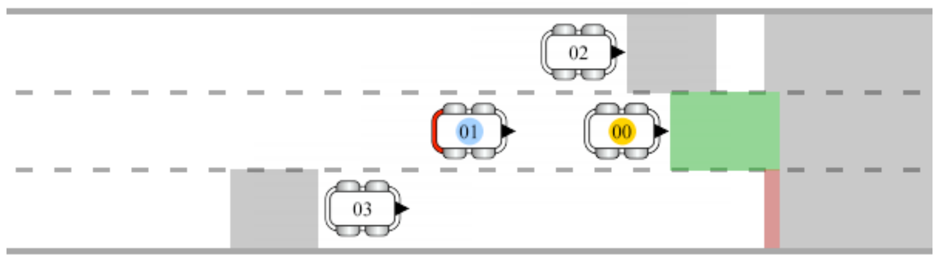
This is the limit of the *decentralized method*: only if a monitor has a complete view of the target–agent's neighborhood it can be sure about its reputation. Mathematically the sufficient condition for monitor h is

$$\mathcal{Q}_i^a \subset \mathcal{Q}_h^o \ . \tag{4.20}$$

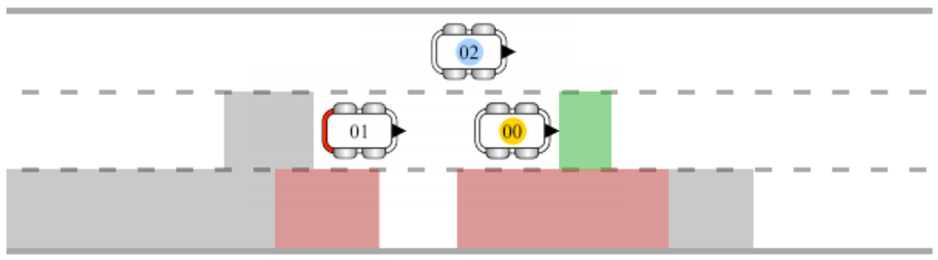
[6]While observing these simulation screenshots please bear in mind that all vehicles are considered as points, so a vehicle is inside a region only if its center do. For example vehicle 3 is not considered inside the red region of Fig. 4.4.c.



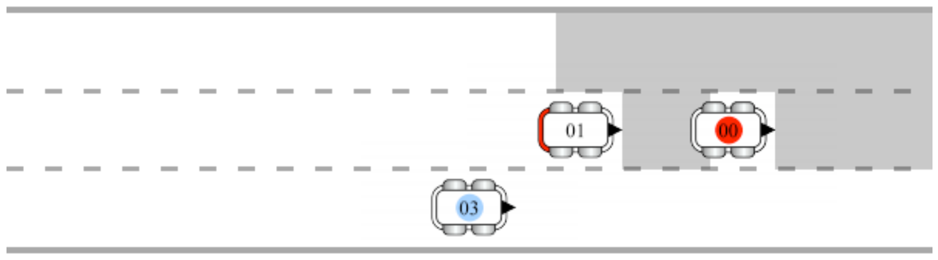
(a)



(b)

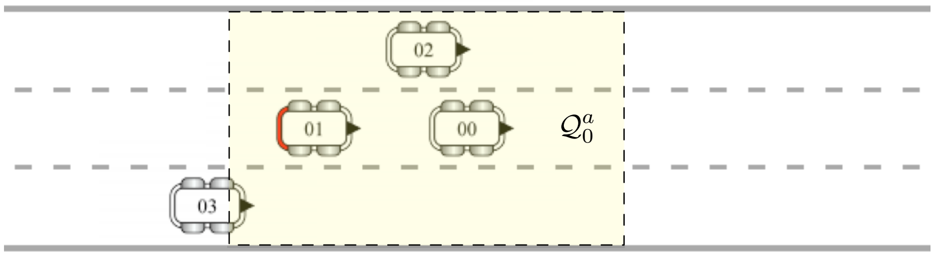


(c)

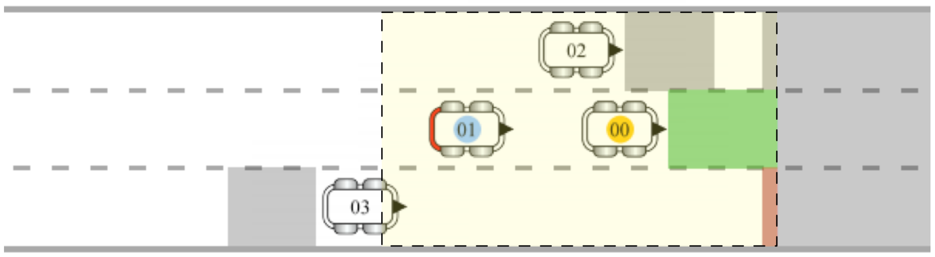


(d)

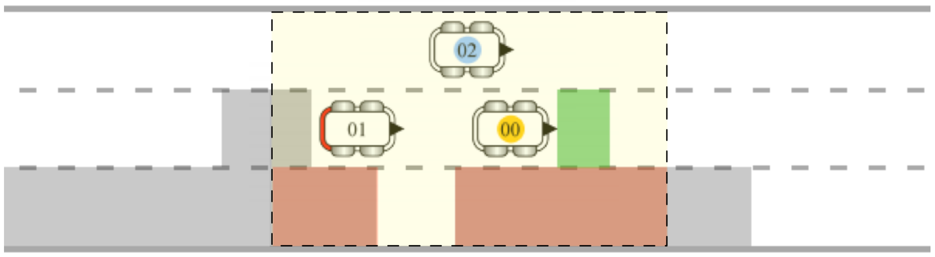
Figure 4.4: A 3–lane highway simulation where 3 agents monitor a faulty one.



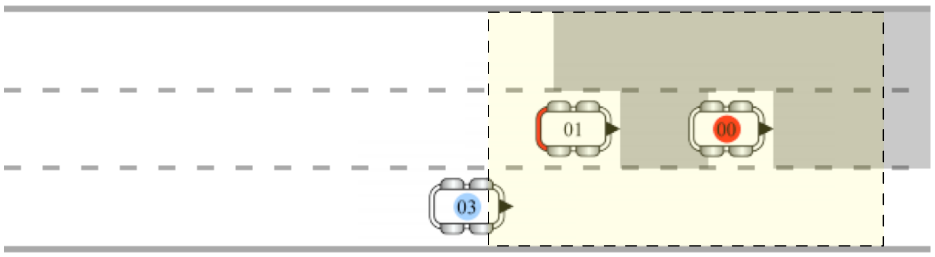
(a)



(b)

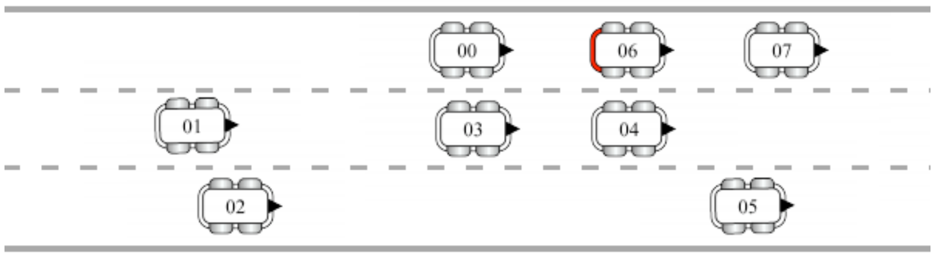


(c)

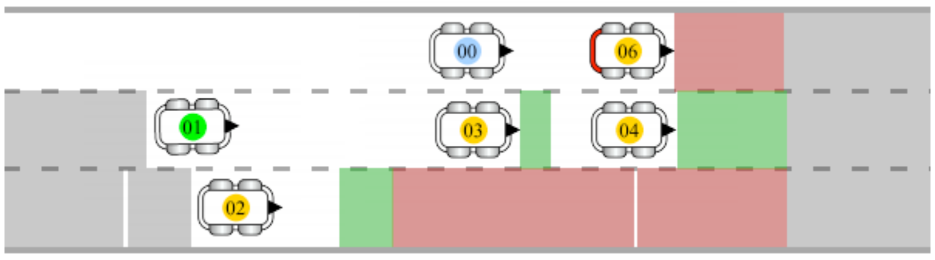


(d)

Figure 4.5: A 3–lane highway simulation where 3 agents monitor a faulty one.



(a)



(b)

Figure 4.6: A 3–lane highway simulation where agent 0 monitors all its visible neighbors.

Of course this condition is only sufficient, because in cases as the one of Fig. 4.4.d the monitor can detect a faulty behavior even without a complete knowledge of target–agent's neighborhood. As we said before, this limit can be overtaken only with a *distributed* monitoring algorithm in which several monitoring agents collaborate to obtain a common shared opinion about a target–agent's reputation.

In Fig. 4.6 is shown another example of the monitoring system applied to a 3–lane automated highway in which vehicle 0 monitors all its visible neighbors and make hypothesis about their neighborhoods[7].

[7]Please refer to Fig. 2.6 at page 11 to remind how the observable region is computed in this simulation.

Chapter 5

Crossing–Street Example

In this chapter we will show another application example of the decentralized monitoring strategy. This example regards a system in which several robots run on a street never changing their direction. Each robot can only accelerate and brake and the streets on which robots are running can intersect with each other. The task for which all the robots collaborate is to avoid collisions. An example of robot configuration is shown in Fig. 5.1.

5.1 System description

The system dynamics are almost the same as in the highway example previously described, except for the robot direction θ_i which remains constant for the whole evolution. The rules that the robots must follow are very simple: a robot musts brake if its trajectory intersect the one of another robot which has precedence over the first. The precedence rule is computed in the "european way", i.e. a robot b, coming from the right side with respect to a robot a's point of view, has precedence over a.

The maneuvers that each robot can perform are the following:

- FAST: the robot increases its speed until it reaches the maximum allowed speed.
- SLOW: the robot decreases its speed until it stops.

The 'slow' maneuver is represented in the following figures with a red triangle in front of the robots.

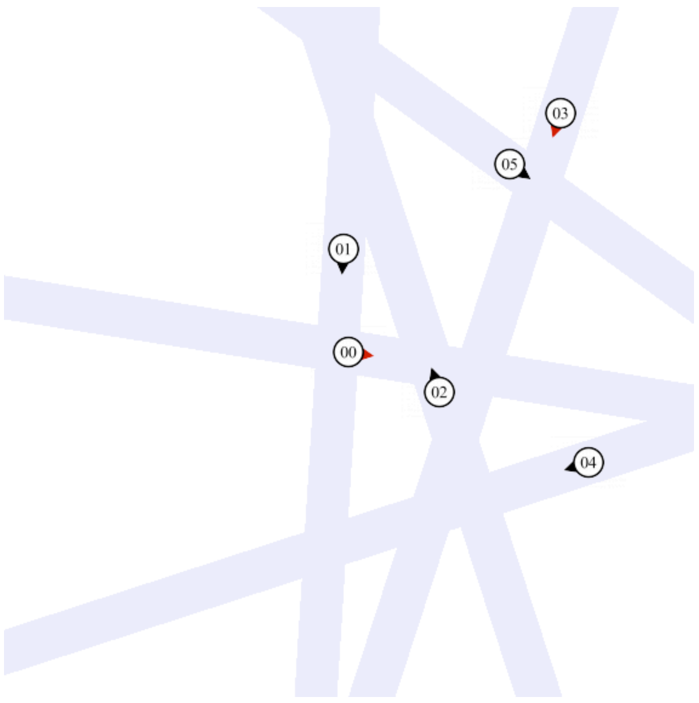


Figure 5.1: A screenshot of a simulated system with robots running on crossing streets.

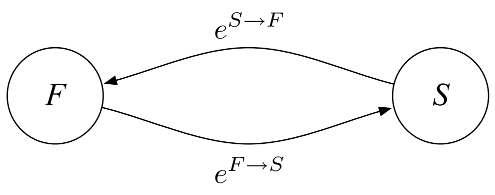


Figure 5.2: The automaton representation used for this second example.

5.2 Automaton definition

The event set $\mathcal{E}$ for the i-th agent is defined as follows:

$$e_i^{F\to S} = \mathrm{Fd}_i \tag{5.1}$$

$$e_i^{S\to F} = \neg\mathrm{Fd}_i \tag{5.2}$$

The expression Fd_i is defined as

$$\mathrm{Fd}_i = \left\{\exists\ q_j \in \mathcal{N}_i \mid f_P(q_i, q_j) = \mathsf{true}\right\}\ , \tag{5.3}$$

where

$$\begin{aligned} f_P(q_i, q_j) &= \left\{d(q_i, q_j) < D\right\} \wedge \left\{p(q_i, q_j) \times v_i \cdot \hat{z} > 0\right\} \\ &\wedge \left\{(\hat{\theta}_i \times \hat{\theta}_j) \cdot \hat{z} > 0\right\} \wedge \left\{(\hat{\theta}_i \cdot \hat{\theta}_j) > 0\right\}\ . \end{aligned} \tag{5.4}$$

The distance d is the euclidean distance and is computed as

$$d(q_i, q_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}\ , \tag{5.5}$$

while the vector p is the "position vector" which expresses the relative position of robot j with respect to the position of robot i:

$$p(q_i, q_j) = \begin{bmatrix} x_j - x_i \\ y_j - y_i \\ 0 \end{bmatrix}\ . \tag{5.6}$$

Moreover, D is a system constant, and with $\hat{z}$ and $\hat{\theta}_i$ we mean the versors representing the z axis direction, and the i–th agent's direction.

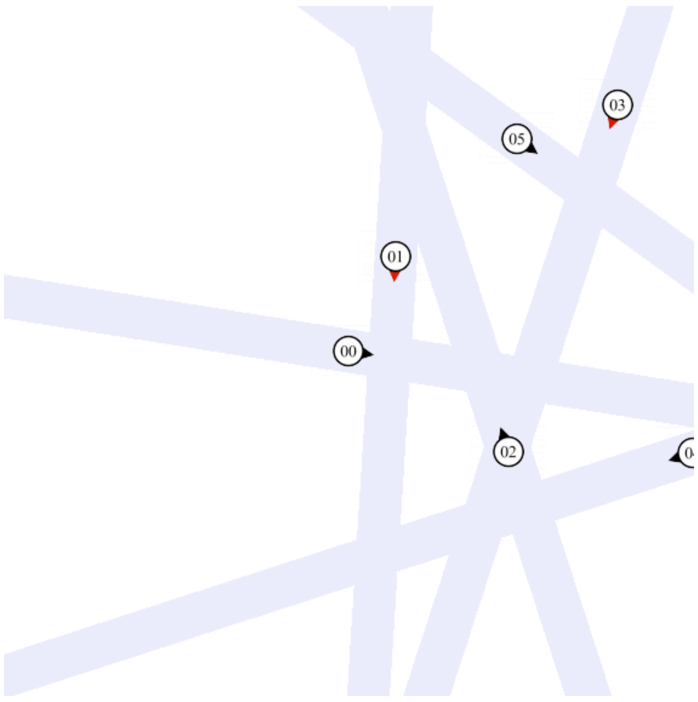
Note 5.1. *In this system the logical variable ξ is not used.*

5.3 Simulation results

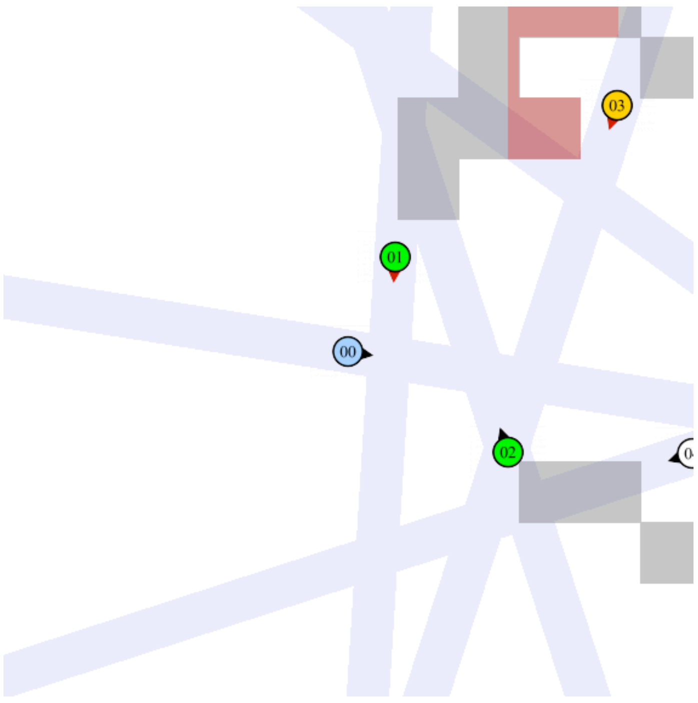
In our simulation, the regions expressed by $\Upsilon(q_i, f_P)$ and the unobservable regions, due to the complexity of calculus, have been hardly approximated with an union of 4–D cuboids, and in the next screenshots these region have been projected onto the x–y plane for representability reasons.

In Fig. 5.3 and Fig. 5.4 are shown the screenshots of a simulation in which robot 0 monitors all its visible neighbors. In the sub–figures (a) is represented the real evolution of the system, while in the sub–figure (b) is represented the point of view of the monitoring robot. Once again it is easy to realize that a monitor, if it works alone, can detect very few faults and in many cases it remains uncertain about target–agents' reputation[1].

[1]To represent the reputation with which each target agent is assigned we use the same conventions (green, yellow, and red circles) as in the previous chapter.

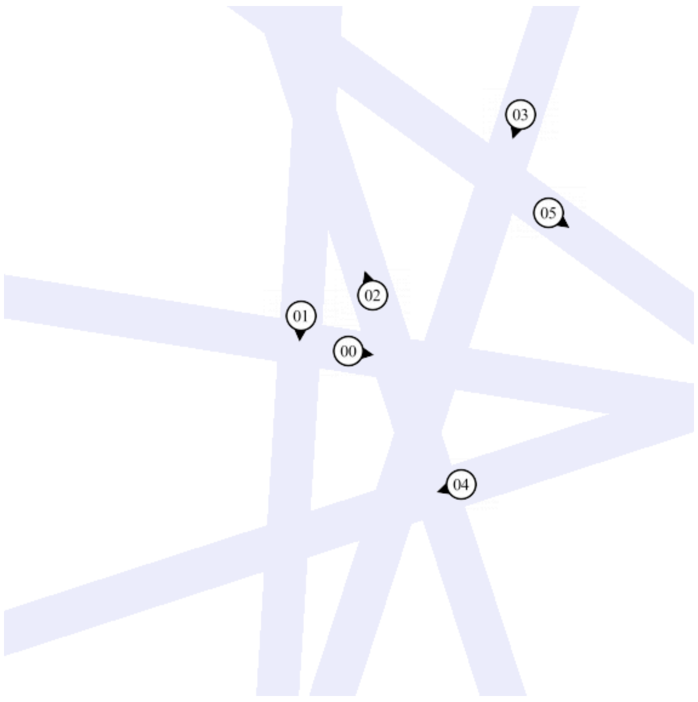


(a)

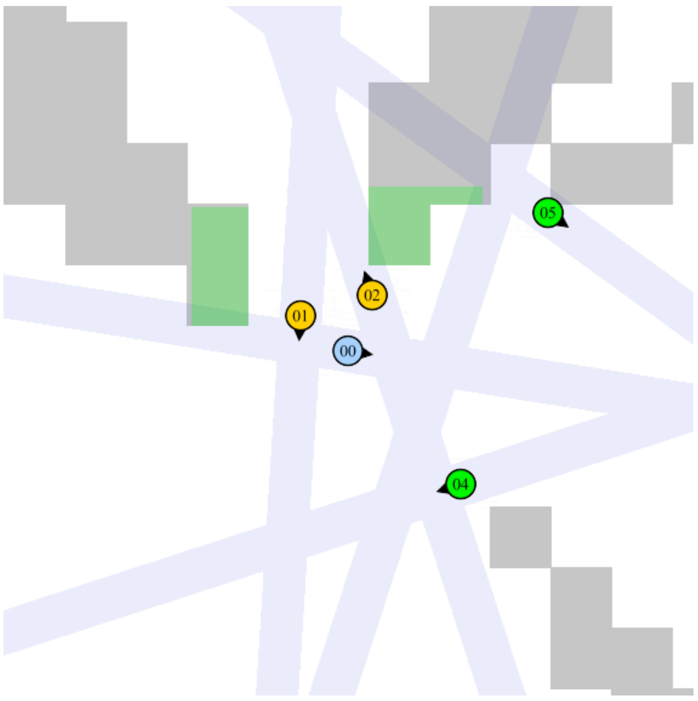


(b)

Figure 5.3: A monitoring simulation of the crossing–street system.



(a)



(b)

Figure 5.4: A monitoring simulation of the crossing–street system.

Chapter 6

From Decentralized to Distributed IDS

In this chapter we will show how the decentralized IDS can be enriched by introducing communication. We will make monitors to communicate with each other implementing a *consensus algorithm.* A consensus algorithm [20], [21] is a strategy by which several networked agents reach a common shared opinion about a certain value, starting from many individual initial estimations.

6.1 Shared information

In our scenario the shared information is an object that we called *reconstructed neighborhood.* The estimated i–th agent's neighborhood will be denoted by Ω_i.

Note 6.1. *In Eq.* (2.23) *at page 11 we already defined the set* $\mathcal{N}_i$ *which contains all the neighboring–agents' sub–state* q*. In this chapter we will give to the* neighborhood *a new wider meaning.*

Let us define more clearly what a reconstructed neighborhood Ω_i is. Each monitor, if it has not estimated a certain transition, puts together all its uncertain events and for each of them it builds a hypothesis k associated with each event. Each event e is made of a list of sub–events $\mathcal{S}(e)$: some of them will be true, while the others will be uncertain[1]. Each uncertain sub–event s corresponds to a supposition (see Section 4.8) about the existence

[1]No sub–events could be false because otherwise the whole event would be false.

(mode = OR) or the non–existence (mode = NOR) of a robot j such that $q_j \in \hat{q}[\,s\,]$. The reconstructed neighborhood Ω_i is a collection of all these pieces of information about the target–agent's behavior. Ω_i is an object containing two sets:

1. The set $\hat{\mathcal{N}_i^a}$ of all the visible target–agent's neighbors whose state q lies inside $\mathcal{Q}_i^a$, defined in Eq. (3.5) at page 23.
2. The set $\mathcal{K}_i$ of all the hypothesis that monitor h makes about the events occurred to agent i.

A hypothesis $k \in \mathcal{K}_i$ is a set composed of three objects:

1. A numerical id associated to the probably occurred event e_i.
2. A set of sub–hypothesis $\mathcal{L}_k$ corresponding to all the sub–events in $\mathcal{S}(e)$ that have been evaluated as uncertain and have an 'OR' evaluation mode.
3. An estimated "negative" region $\mathcal{Q}_k^-$, where monitor h supposes that there are not robot.

A sub–hypothesis $l \in \mathcal{L}_k$ is in its turn a set composed of two objects:

1. A numerical id associated to the probably occurred sub–event s_i.
2. An estimated "positive" region $\mathcal{Q}_l^+$, computed as $\Upsilon(q_i, \xi_i, b)$, where monitor h supposes that there should be a robot.

When a certain event is detected, the set Ω_i is composed of an empty hypothesis object which contains only the id of the detected event.

The method for building the reconstructed neighborhood Ω_i is shown in Algorithm 6.1, where the quantity $\hat{q}[\,\cdot\,]$ is the array shown in Algorithm 4.3 and the quantity $\hat{\mathcal{N}_i^a}$ is defined in Eq. (3.5) at page 23.

6.2 Communication model

Let us represent the communication capabilities of the monitors in our system with a non–oriented graph $G_c(V, E)$ [22, 23], where V is the set of all the nodes (or vertices) and E is the set of all the edges connecting two different nodes. An example of a non–oriented graph is shown in Fig. 6.1.

Algorithm 6.1 The building strategy of Ω_i.

Inputs: $\mathcal{E}$, $\hat{q}[\,\cdot\,]$, $\hat{\mathcal{N}}_i^a$
Outputs: Ω_i

```
1: $\mathcal{K}_i \leftarrow \emptyset$                                    ◁ an initial empty set of hypothesis
2: for all $e \in \mathcal{E}$ do
3:     if valueof($e_i$) = uncertain then
4:         $\mathcal{L}_k \leftarrow \emptyset$                            ◁ an initial empty set of sub–hypothesis
5:         $\mathcal{Q}_k^- \leftarrow \emptyset$                          ◁ an initial empty "negative" region
6:         for all $s \in \mathcal{S}(e)$ do
7:             if valueof($s_i$) = uncertain then
8:                 if modeof($s$) = NOR then
9:                     $\mathcal{Q}_k^- \leftarrow \mathcal{Q}_k^- \cup \hat{q}[\,s\,]$    ◁ there should not be any robot here
10:                else if modeof($s$) = OR then
11:                    $l \leftarrow \{\text{id}(s), \hat{q}[\,s\,]\}$          ◁ build the sub–hypothesis
12:                    $\mathcal{L}_k \leftarrow \mathcal{L}_k \cup l$          ◁ update the sub–hypothesis list
13:                end if
14:            end if
15:        end for
16:        $k \leftarrow \{\text{id}(e), \mathcal{L}_k, \mathcal{Q}_k^-\}$       ◁ build the hypothesis
17:        $\mathcal{K}_i \leftarrow \mathcal{K}_i \cup k$                      ◁ update the hypothesis list
18:    else if valueof($e_i$) = true then
19:        $k \leftarrow \{\text{id}(e), \emptyset, \emptyset\}$    ◁ we do not need nor sub–hypothesis, nor $\mathcal{Q}_k^-$
20:        $\mathcal{K}_i \leftarrow \{k\}$                                    ◁ $k$ is the only hypothesis
21:        $\Omega_i \leftarrow \{\hat{\mathcal{N}}_i^a, \mathcal{K}_i\}$
22:        return $\Omega_i$                                              ◁ we can exit from the algorithm
23:    end if
24: end for
25: $\Omega_i \leftarrow \{\hat{\mathcal{N}}_i^a, \mathcal{K}_i\}$
26: return $\Omega_i$
```

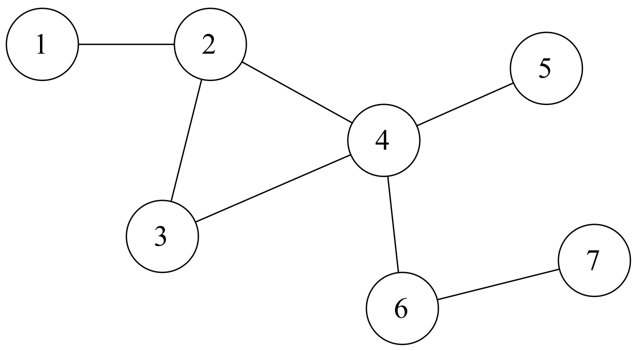


Figure 6.1: A connected communication graph.

Definition 6.1. *The distance $d(i,j)$ between node i and node j is defined as the shortest path length*[2] *in graph G_c between the two nodes [24]. If a path connecting the two nodes does not exists, then $d(i,i) = \infty$. It also holds $d(i,i) = 0$, $\forall i \in V$.*

In our system agent i, represented by the graph node i, is able to communicate with all the agents represented by the nodes of its *communication neighborhood* $V_i(1)$, where

$$V_i(p) = \left\{ j \in V \mid d(i,j) \leq p \right\} . \tag{6.1}$$

Supposing that the i–th vehicle is able to communicate whit all the surrounding vehicle whose *euclidean* distance from i is less than a determined maximum value D_c, then agent i's communication neighborhood can be expressed by

$$\mathcal{N}_i^c(1) = \left\{ q_j \mid d(q_i, q_j) \leq D_c \right\} , \tag{6.2}$$

and

$$\mathcal{N}_i^c(p) = \left\{ q_j \mid d(q_i, q_j) \leq p\, D_c \right\} . \tag{6.3}$$

6.3 Consensus algorithm

Assuming that a *merging operation* between different reconstructed neighborhoods is defined, the consensus algorithm used in our IDS is quite simple:

[2]This is also known as the geodesic distance.

at each consensus step n, each monitor broadcasts its reconstructed neighborhood $\Omega_i(n)$ to all the monitors with which it is able to communicate. At the same time it receives their reconstructed neighborhoods, and updates its estimation merging it with the received ones. This procedure is repeated for a finite number of steps, i.e. it ends at a certain step $n = \overline{n}$.

The number of consensus steps $\overline{n}$ can be chosen considering that after p steps the information broadcasted by agent i reaches all the nodes in $V_i(p)$ and therefore, all the agents whose state q is inside $\mathcal{N}_i^c(p)$ i.e. they are distant less than $p\, D_c$ from agent i. In Section 6.6 we will prove that the minimum number of consensus steps needed for the shared estimation to converge depends on the communication graph diameter[3].

6.4 Merging operation

We will describe how the merging between two reconstructed neighborhoods of agent i is made by monitoring agents. Let the two neighborhoods be

$$\Omega_{i,1} = \left\{\hat{\mathcal{N}}_{i,1}^a, \mathcal{K}_{i,1}\right\} , \tag{6.4}$$

and

$$\Omega_{i,2} = \left\{\hat{\mathcal{N}}_{i,2}^a, \mathcal{K}_{i,2}\right\} . \tag{6.5}$$

The lists $\hat{\mathcal{N}}_{i,1}^a$ and $\hat{\mathcal{N}}_{i,2}^a$ of the measured states can be simply joined together[4], while for the hypothesis set we need a procedure a bit less simple:

a. We start from one of the two hypothesis set, e.g. $\mathcal{K}_{i,1}$.

b. For each hypothesis of $\mathcal{K}_{i,2}$ we have to find the corresponding hypothesis in $\mathcal{K}_{i,1}$ and intersect the two “negative” regions. We have to do the same thing for each couple of sub–hypothesis and their “positive” regions.

c. For each state $q \in \hat{\mathcal{N}}_{i,2}^a$ we have to check if it refutes one of the hypothesis or if it verifies one of the sub–hypothesis of $\mathcal{K}_{i,1}$.

[3]From [25]: ≪A graph's diameter is the largest number of vertices which must be traversed in order to travel from one vertex to another when paths which backtrack, detour, or loop are excluded from consideration≫. The graph of Fig. 6.1 e.g. has diameter 4.

[4]We suppose that monitors does not lie.

d. Each refuted hypothesis and each verified sub–hypothesis have to be removed from their own list.

e. Whenever we obtain a complete verification of a hypothesis, we must remove all the other ones from the list.

In Algorithm 6.2 we describe more precisely the procedure. We denoted by '$\odot$' the merging operation.

6.5 Reputation

Once the reconstructed neighborhood Ω_i has been generated, the reputation value with which the target agent i is assigned can be obtained as follows:

- $r_i = \mathsf{faulty}$ if the hypothesis set $\mathcal{K}_i$ is empty.
- $r_i = \mathsf{correct}$ if
    a. the hypothesis set $\mathcal{K}_i$ is composed of only one element k^*,
    b. the "negative" region $\mathcal{Q}^-_{k^*}$ is empty,
    c. and the sub–hypothesis set $\mathcal{L}_{k^*}$ is empty.
- $r_i = \mathsf{uncertain}$ otherwise.

6.6 Consensus convergence theorem

Now we will prove that a more general version of the consensus algorithm described in the previous sections can converge to a shared common value, which consists in the simultaneous merge of all the agents' initial reconstructed neighborhoods.

Suppose that a time–invariant and non–oriented graph G_c represents how the N agents of a system can communicate with each other. Suppose also that each agent can compute its initial estimation $\xi_i(0)$ of a generic value $\tilde{\xi} \in \Xi$, and that a generic commutative and associative operator

$$\odot : \Xi \times \Xi \to \Xi \tag{6.6}$$

is defined.

Algorithm 6.2 Merging operation between two reconstructed neighborhoods.

Inputs: $\Omega_{i,1}$, $\Omega_{i,2}$
Outputs: $\Omega_i = \Omega_{i,1} \odot \Omega_{i,2}$

```
 1: $\mathcal{K}_i \leftarrow \mathcal{K}_{i,1}$                          ◁ start with a hypothesis set
 2: for all $k' \in \mathcal{K}_{i,2}$ do
 3:   $k \leftarrow$ lookfor($k'$, $\mathcal{K}_i$)                       ◁ find the corresponding hypothesis
 4:   if $\exists\ k$ then
 5:     $\mathcal{Q}_k^- \leftarrow \mathcal{Q}_k^- \odot \mathcal{Q}_{k'}^-$             ◁ refine the "negative" region $\mathcal{Q}_k^-$
 6:     for all $l' \in \mathcal{L}_{k'}$ do
 7:       $l \leftarrow$ lookfor($l'$, $\mathcal{L}_k$)                 ◁ find the corresponding sub–hypothesis
 8:       if $\exists\ l$ then
 9:         $\mathcal{Q}_l^+ \leftarrow \mathcal{Q}_l^+ \odot \mathcal{Q}_{l'}^+$         ◁ refine the "positive" region $\mathcal{Q}_l^+$
10:       end if
11:     end for
12:   end if
13: end for
14: for all $q \in \hat{\mathcal{N}}_{i,2}^a$ do
15:   for all $k \in \mathcal{K}_i$ do
16:     if $q \in \mathcal{Q}_k^-$ then
17:       $\mathcal{K}_i \leftarrow \mathcal{K}_i \setminus k$                      ◁ the hypothesis $k$ was false
18:       continue whith next $k$
19:     end if
20:     for all $l \in \mathcal{L}_k$ do
21:       if $q \in \mathcal{Q}_l^+$ then
22:         $\mathcal{L}_k \leftarrow \mathcal{L}_k \setminus l$                  ◁ the sub–hypothesis $l$ was true
23:       end if
24:     end for
25:     if $\mathcal{L}_k = \emptyset$ and $\mathcal{Q}_k^- = \emptyset$ then
26:       $\mathcal{K}_i \leftarrow \{k\}$                         ◁ the hypothesis $k$ has been verified:
27:       break the 2 nested loops                      we found a certain event
28:     end if
29:   end for
30: end for
31: $\hat{\mathcal{N}}_i^a = \hat{\mathcal{N}}_{i,1}^a \cup \hat{\mathcal{N}}_{i,2}^a$                 ◁ merge the list of measured states
32: $\Omega_i = \{\hat{\mathcal{N}}_i^a, \mathcal{K}_i\}$               ◁ build the new reconstructed neighborhood
33: return $\Omega_i$
```

Definition 6.2. *Let us define a distributed consensus algorithm by which each agent can merge its estimation with the one of its neighbors at each consensus step n:*

$$\xi_i(n+1) = \bigodot_{j \in V_i(1)} \xi_j(n) \ , \tag{6.7}$$

where the set $V_i(\cdot)$ is the one defined in Eq. (6.1)*.*

Definition 6.3. *Let us also say that if the equation*

$$\xi \odot \xi = \xi \tag{6.8}$$

holds $\forall \xi \in \Xi$, then the operator $\odot$ is said to be self–invariant*.*

Lemma 6.1. *If $\odot$ is associative, commutative, and self–invariant, the expression*

$$\xi_i(n) = \bigodot_{j \in V_i(n)} \xi_j(0) \tag{6.9}$$

holds for any i and n.

Proof. Lemma 6.1 can be proved by logical induction. Consider the evolution of the i-th agent estimation starting from its initial estimation $\xi_i(0)$: from Eq. (6.7) we get

$$\xi_i(1) = \bigodot_{j \in V_i(1)} \xi_j(0) \quad \forall i \in N \ . \tag{6.10}$$

Now assume that Eq. (6.9) holds for a certain value of n. Then from Eq. (6.7) and Eq. (6.8) we obtain

$$\xi_i(n+1) = \bigodot_{j \in V_i(1)} \left\{ \bigodot_{m \in V_j(n)} \xi_m(0) \right\} = \bigodot_{m \in V_i(n+1)} \xi_m(0) \ , \tag{6.11}$$

where the last equivalence holds if $\odot$ is commutative, associative, and self–invariant.

Since Eq. (6.9) holds for $n = 0$, as shown in Eq. (6.10), the general expression for $\xi_i(n)$ in Eq. (6.9) can be obtained by induction. ■

Now we are ready to give the main result in Theorem 6.1.

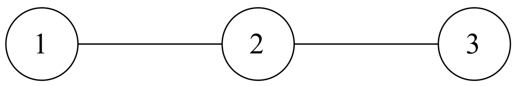


Figure 6.2: A simple 3–node communication graph.

Theorem 6.1. *If $\odot$ is commutative, associative, and self–invariant, then, for any set of initial conditions and for any connected graph, the distributed consensus algorithm of Eq. 6.7 converges to a unique network decision on the estimation of $\tilde{\xi}$ in a finite number $\overline{n}$ of consensus steps. Furthermore the shared network decision is equal to the centralized computation*

$$\xi^* = \bigodot_{i \in N} \xi_i(0) \ , \tag{6.12}$$

and the minimum number of consensus steps needed is

$$\overline{n} \leq \max_{i,j \in N} d(i,j) \ , \tag{6.13}$$

i.e. $\overline{n}$ is not greater than the diameter of the communication graph G_c.

Proof. Theorem 6.1 can be proved by observing that, if

$$\overline{n} = \max_{i,j \in N} d(i,j) \ , \tag{6.14}$$

then, since graph G_c is connected,

$$V_i(n) = V \quad \forall n \geq \overline{n} \ , \tag{6.15}$$

and for Lemma 6.1 and Eq. (6.12) we obtain

$$\xi_i(n) = \bigodot_{j \in V_i(n)} \xi_j(0) = \bigodot_{j \in N} \xi_j(0) = \xi^* \ , \tag{6.16}$$

for any $i \in V$, and for any $n \geq \overline{n}$, as requested. ■

Note 6.2. *If $\odot$ is commutative and associative, but not self–invariant, then it may exist a set of initial condition, and a connected graph for which convergence is not reached in n steps. Let us consider the example of Fig. 6.2 in which $\Xi \subseteq \mathbb{R}$, and $\odot \equiv +$, and let the initial conditions be the following ones:*

$$\xi_1(0) = 1 \ , \quad \xi_2(0) = 2 \ , \quad \xi_3(0) = 3 \ . \tag{6.17}$$

From Eq. (6.13) *we obtain $n \leq 2$; furthermore, from the definition of the consensus algorithm in Eq.* (6.7)*,*

$$\xi_1(2) = \xi_1(1) + \xi_2(1) = \big(\xi_1(0) + \xi_2(0)\big) + \big(\xi_1(0) + \xi_2(0) + \xi_3(0)\big) \tag{6.18}$$

and

$$\xi_3(2) = \xi_3(1) + \xi_2(1) = \big(\xi_3(0) + \xi_2(0)\big) + \big(\xi_1(0) + \xi_2(0) + \xi_3(0)\big)\ . \tag{6.19}$$

Bearing in mind Eq. (6.17)*, it's easy to find out that $\xi_1(2) \neq \xi_3(2)$, and convergence is not reached in $\overline{n} \leq 2$ steps.*

In our system, the algorithm by which the reconstructed neighborhoods are updated is made of logical *unions* and *intersection*. Therefore it is straightforward to realize that the operation described in Algorithm 6.2 is commutative, associative, and self–invariant. Thus Theorem 6.1 can be used also in our scenario, where ξ is the reconstructed neighborhood and $\odot$ is the operation described in Algorithm 6.2.

6.7 Simulation results

We will show now some evolution example of the consensus algorithm with different vehicle configurations. We defined a function μ which expresses the "amount" of uncertainty in a given reconstructed neighborhood Ω_i. This function is useful to show how monitors' uncertainty decreases at each consensus step, while they merge their knowledge together.

Note 6.3. *The reconstructed neighborhood object is composed of a set of measured states and a set of hypothesis. In all the following examples the neighborhoods are composed of only one hypothesis, but this is only for chance.*

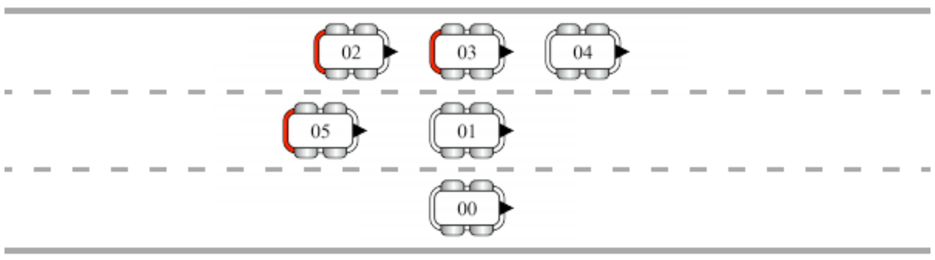
Example 6.1. Six vehicles are configured as shown in Fig. 6.3.a, and the communication distance D_c has been chosen such that vehicles can communicate as shown in Fig. 6.3.b. Vehicles 2, 3, 4, and 5 are monitoring vehicle 1. Vehicle 0 prevents vehicle 1 from turning right on the first lane, so vehicle 1's behavior is correct. Furthermore since the 4 monitoring vehicles, if communicating together, have a complete vision of vehicle 1's active region

$\mathcal{Q}_i^a$, they are able to obtain *a certain opinion* on its reputation level. Each vehicle has its initial subjective vision, shown in Fig. 6.4. The communication graph's diameter is 3, so we expect that the consensus algorithm will converge in at most 3 steps, and that all vehicles get the same estimation

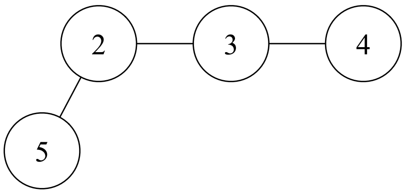
$$\Omega_1^* = \Omega_{1,2}(0) \odot \Omega_{1,3}(0) \odot \Omega_{1,4}(0) \odot \Omega_{1,5}(0) \ , \tag{6.20}$$

shown in Fig. 6.3.c.

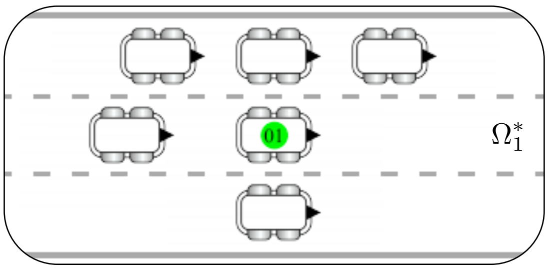
The consensus algorithm steps are shown in Fig. 6.5–6.8. In Fig. 6.9–6.11 are shown respectively the reputation value r_1 with which vehicle 1 is assigned, the uncertainty measure μ of each monitor's reconstructed neighborhood, and the communication overhead, i.e. the size in bytes of the message sent by each monitor. It is interesting to see how all these values converge to the same result in at most 3 consensus steps.



(a)

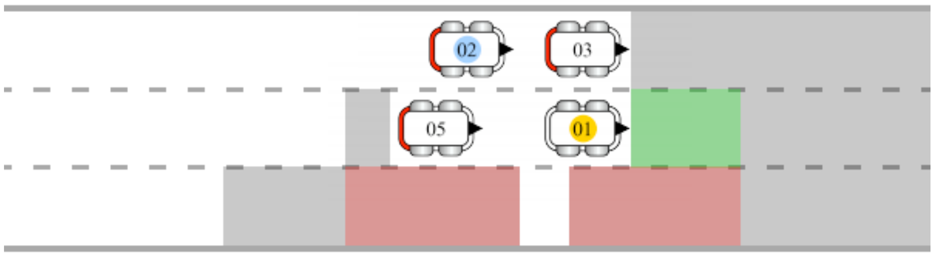


(b)

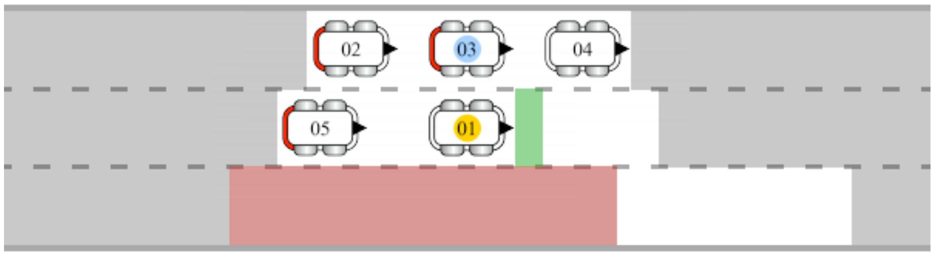


(c)

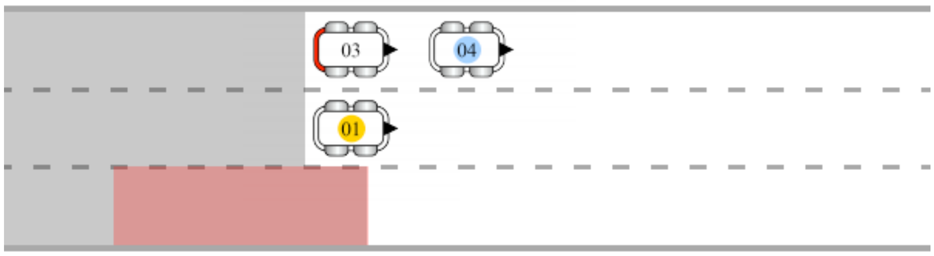
Figure 6.3: Vehicle configuration (a), communication graph (b), and centralized monitoring result Ω_1^* (c) for Example 6.1.



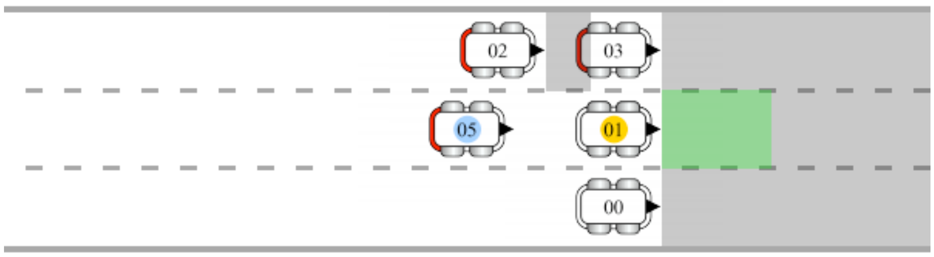
(a)



(b)



(c)



(d)

Figure 6.4: Vehicles' initial estimations — Example 6.1.

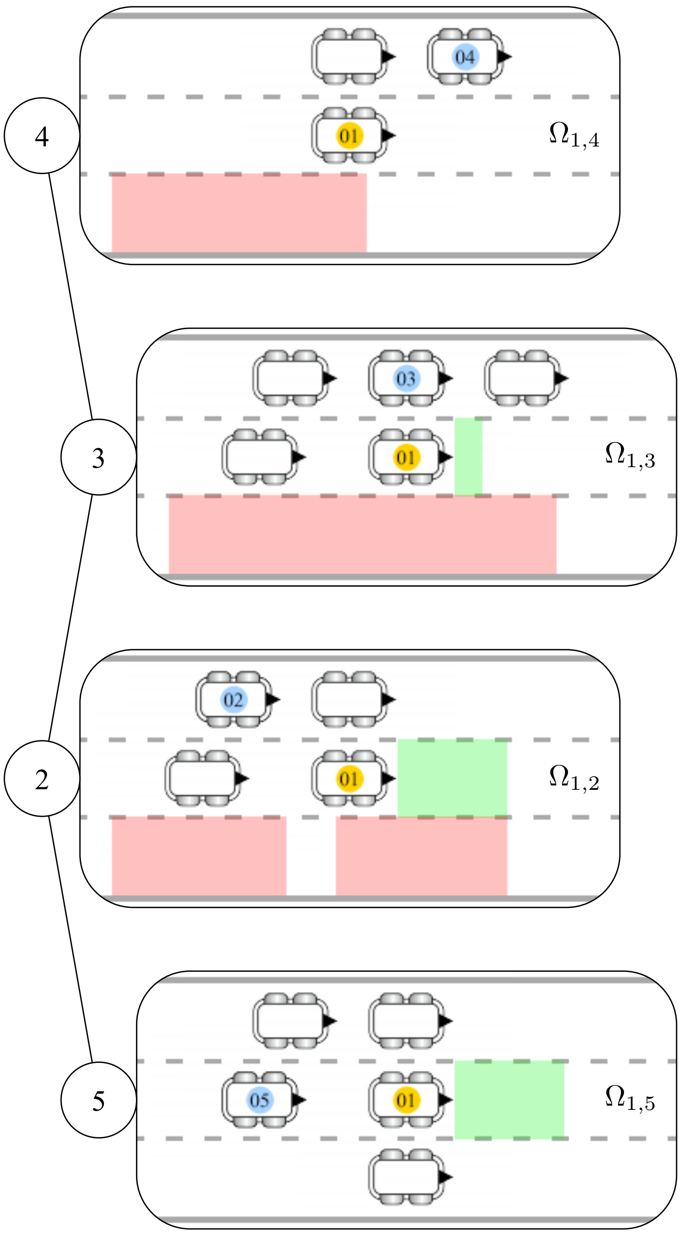


Figure 6.5: Consensus step 0 (initial conditions) — Example 6.1.

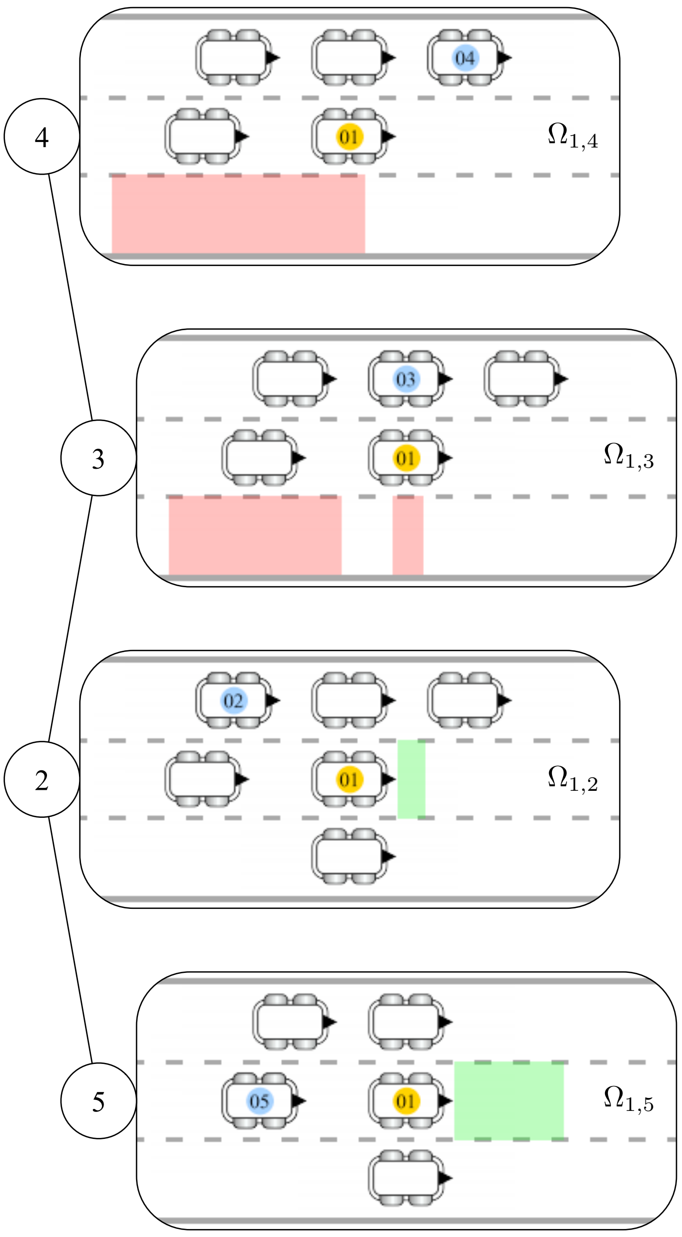


Figure 6.6: Consensus step 1 — Example 6.1.

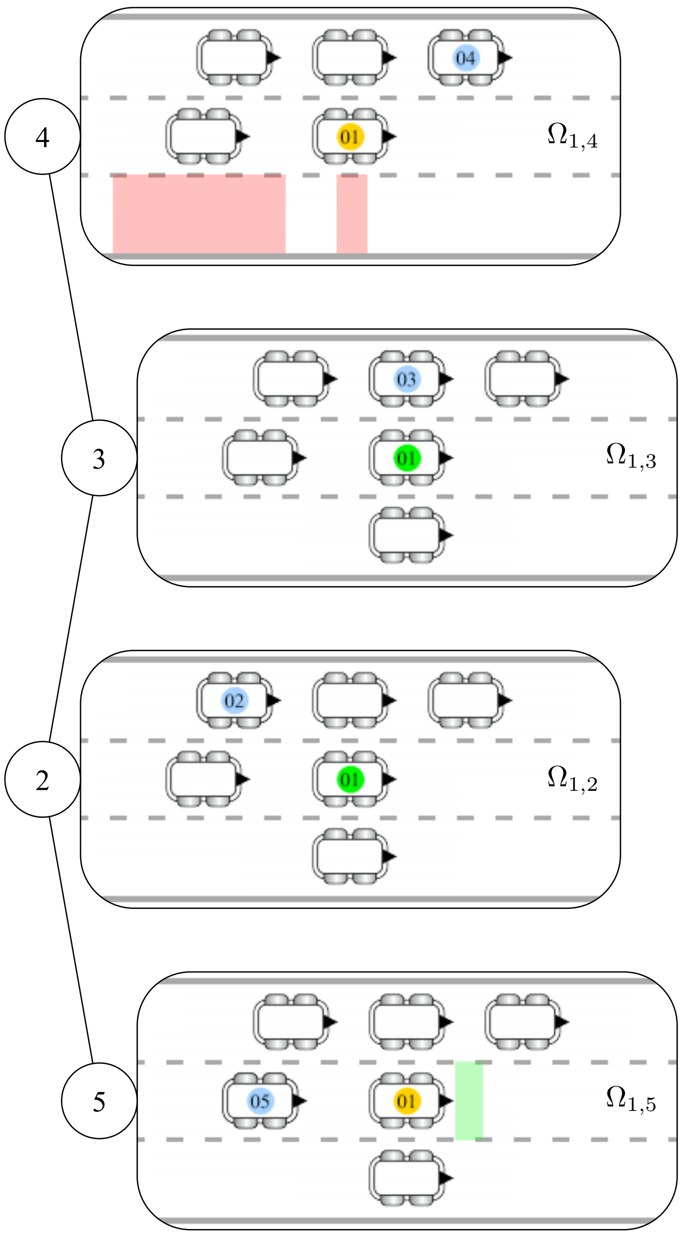


Figure 6.7: Consensus step 2 — Example 6.1.

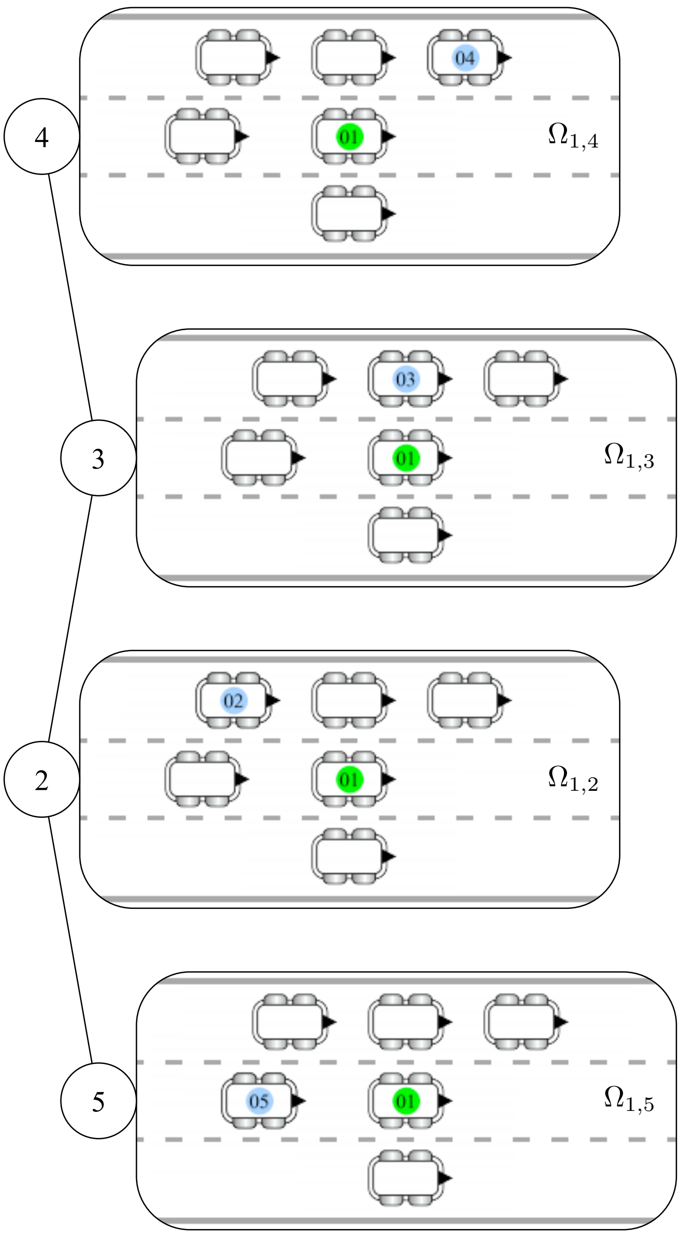


Figure 6.8: Consensus step 3 — Example 6.1.

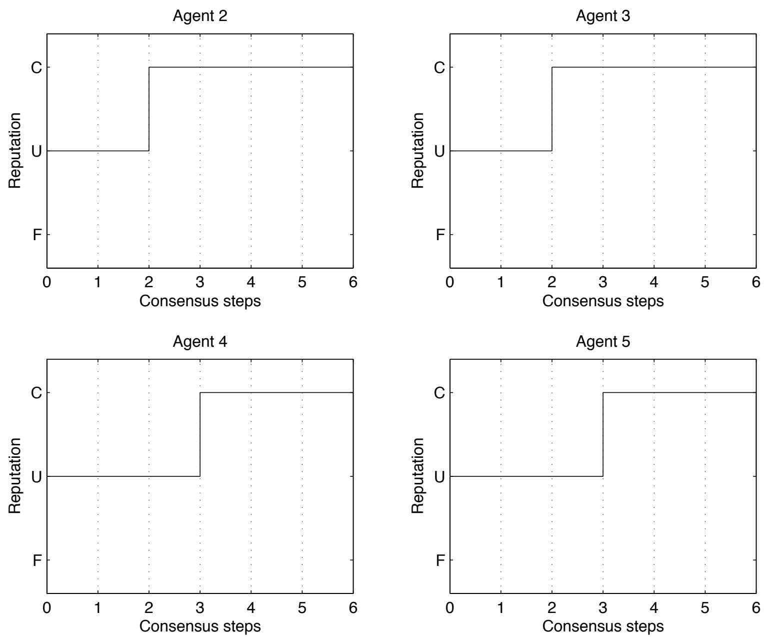


Figure 6.9: Reputation values — Example 6.1.

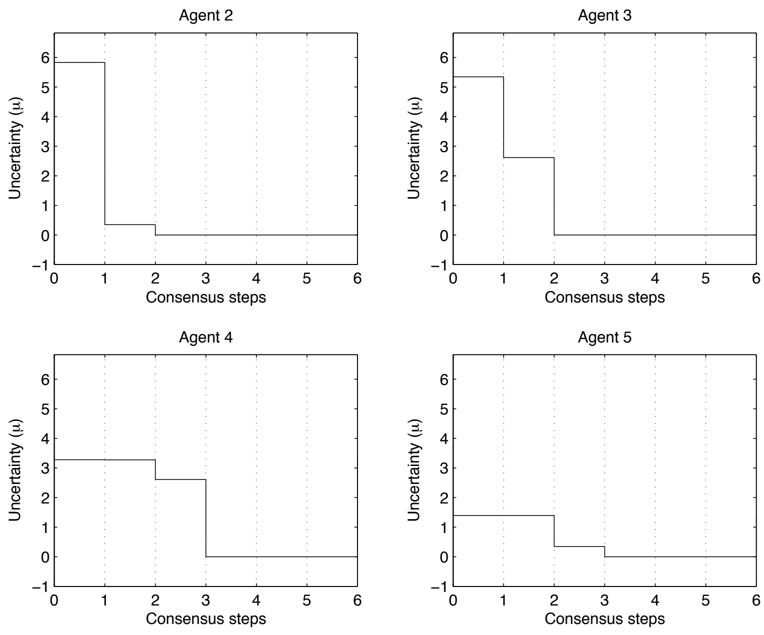


Figure 6.10: Uncertainty measure μ — Example 6.1.

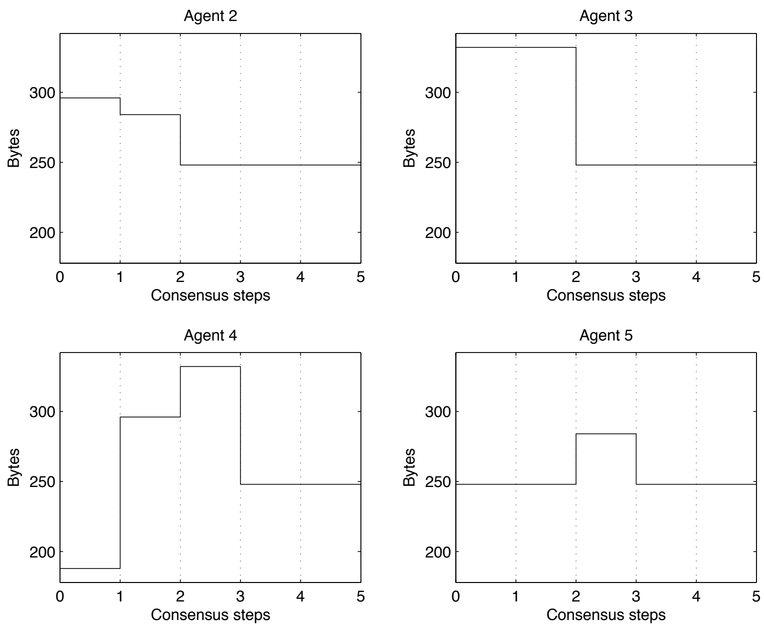
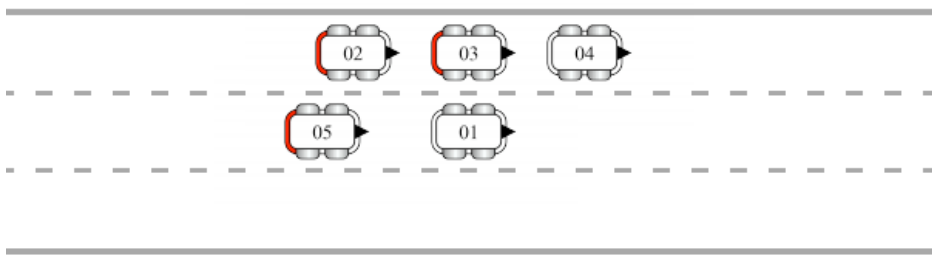
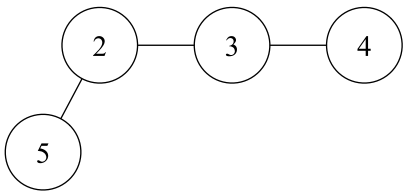


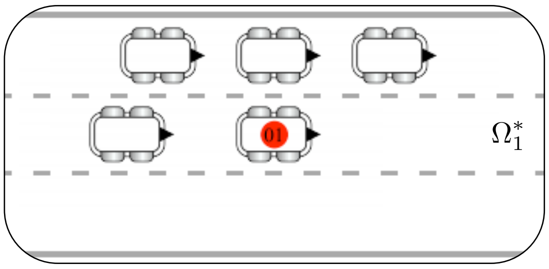
Figure 6.11: Communication overhead — Example 6.1.



(a)



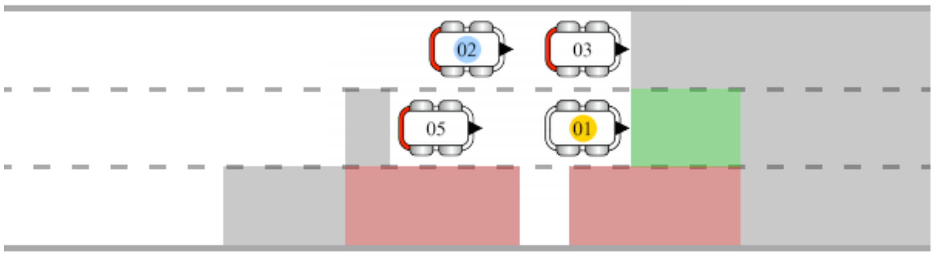
(b)



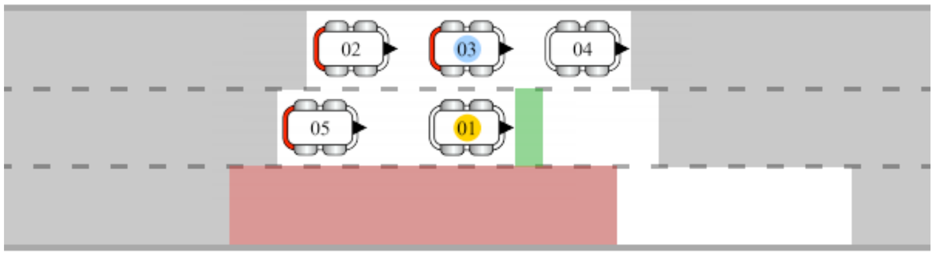
(c)

Figure 6.12: Vehicle configuration (a), communication graph (b), and centralized monitoring result Ω_1^* (c) for Example 6.2.

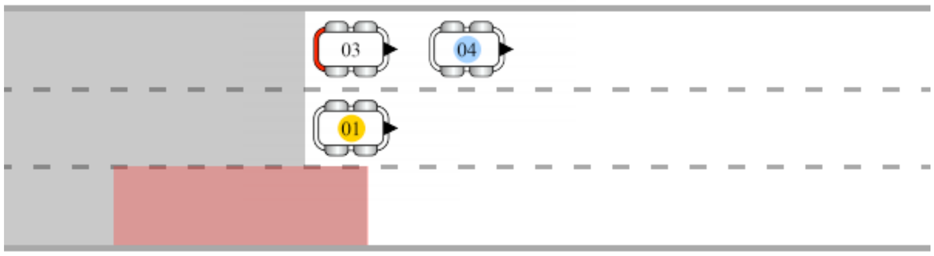
Example 6.2. Five vehicles are configured as shown in Fig. 6.12.a, and the communication graph is the same as in Example 6.1. Vehicles 2, 3, 4, and 5 are monitoring vehicle 1, but this time it does not exists a Vehicle 0 preventing vehicle 1 from turning right on the first lane. Therefore vehicle 1's behavior is faulty. Once again the consensus is reached in 3 steps. The initial estimations, the consensus evolution and all the results are shown in Fig. 6.13–6.20.



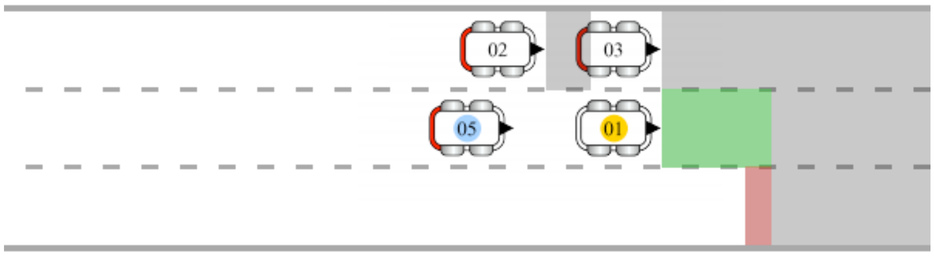
(a)



(b)



(c)



(d)

Figure 6.13: Vehicles' initial estimations — Example 6.2.

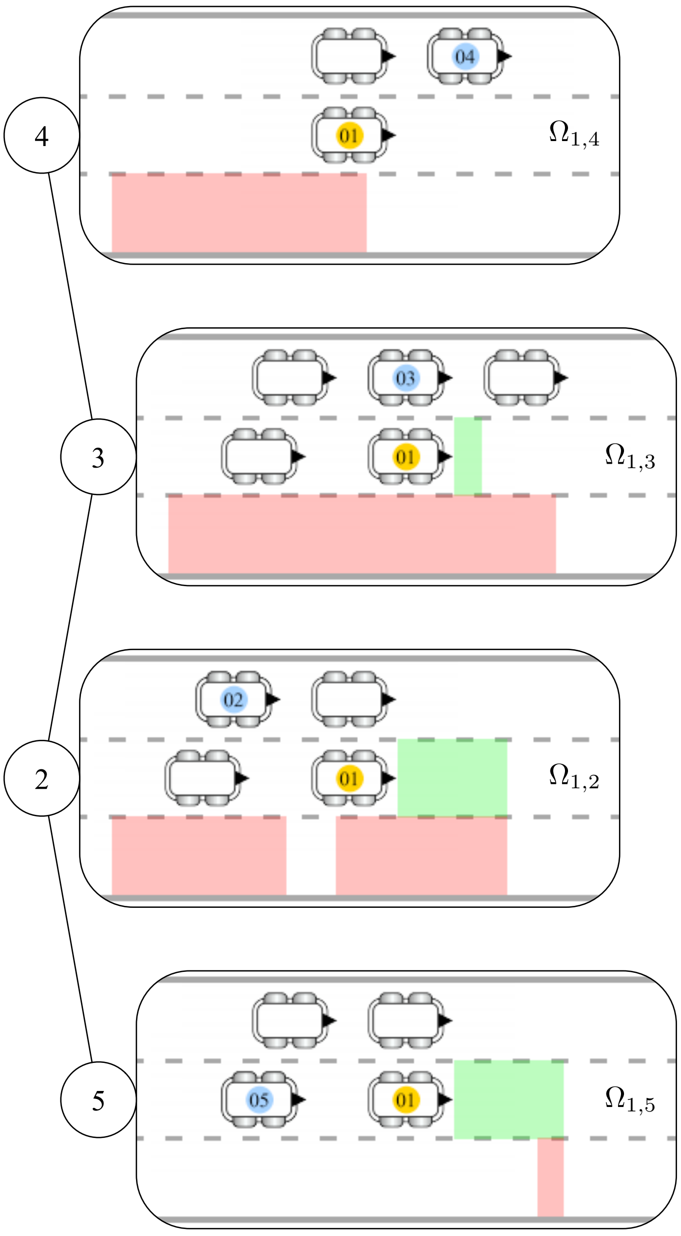


Figure 6.14: Consensus step 0 (initial conditions) — Example 6.2.

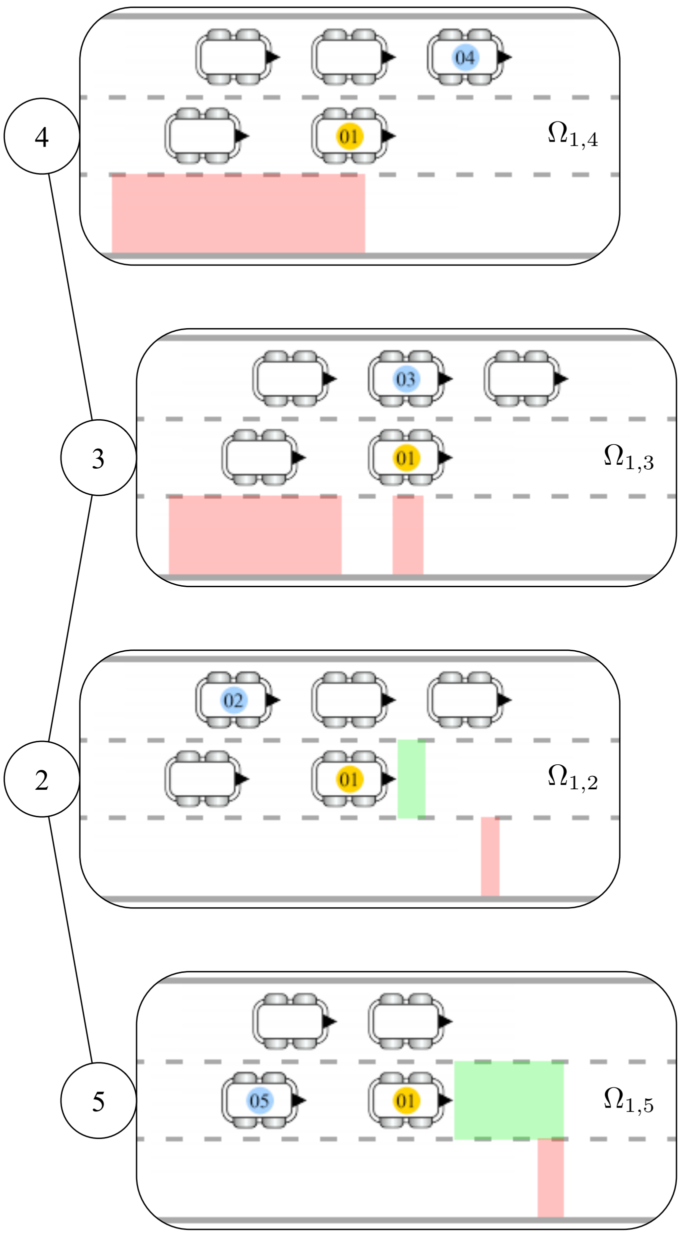


Figure 6.15: Consensus step 1 — Example 6.2.

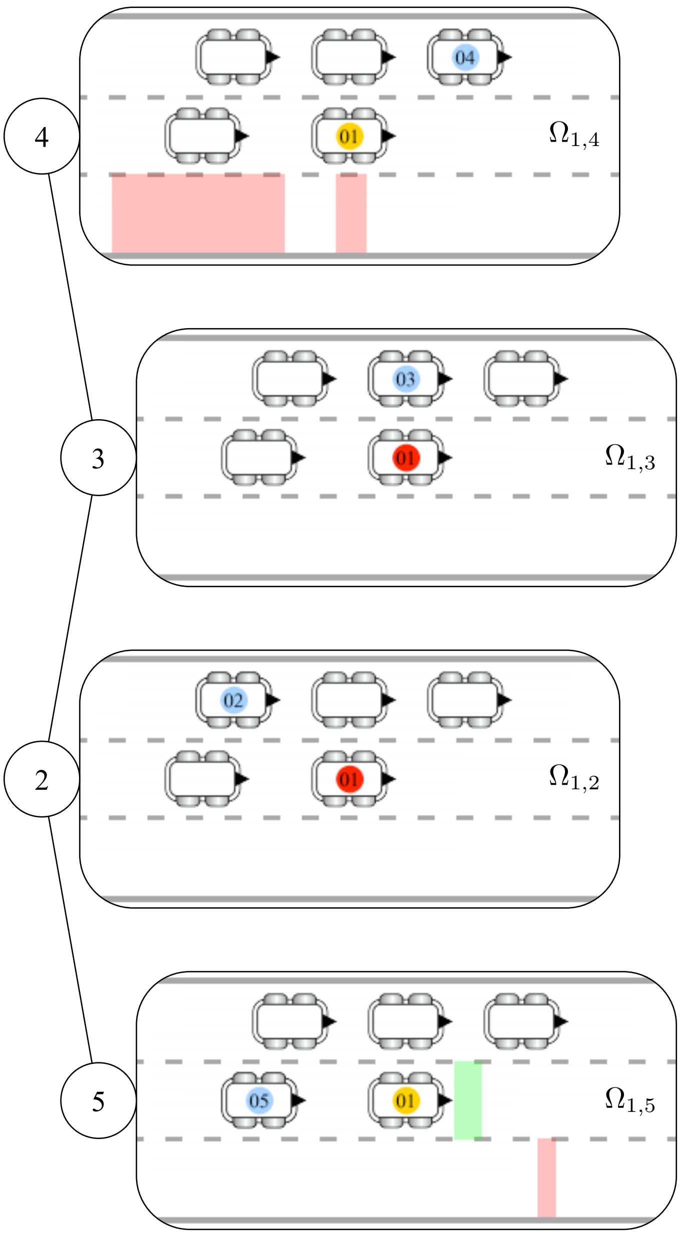


Figure 6.16: Consensus step 2 — Example 6.2.

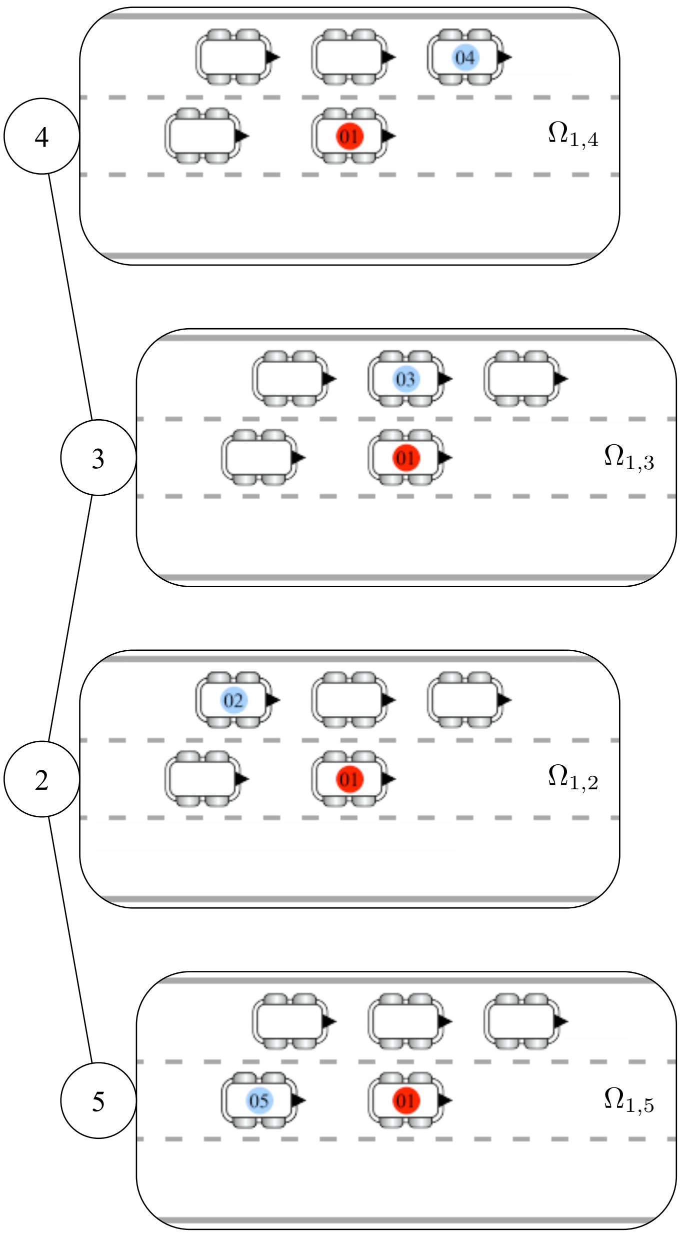


Figure 6.17: Consensus step 3 — Example 6.2.

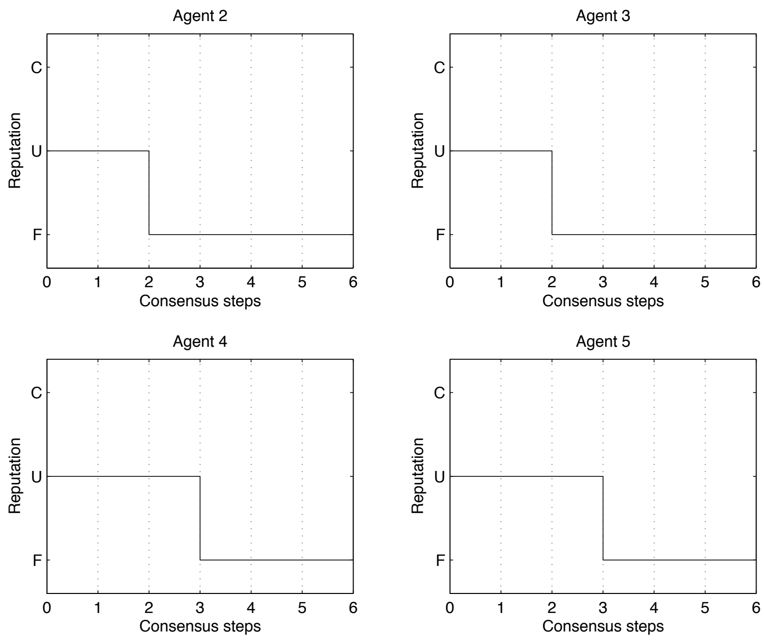


Figure 6.18: Reputation values — Example 6.2.

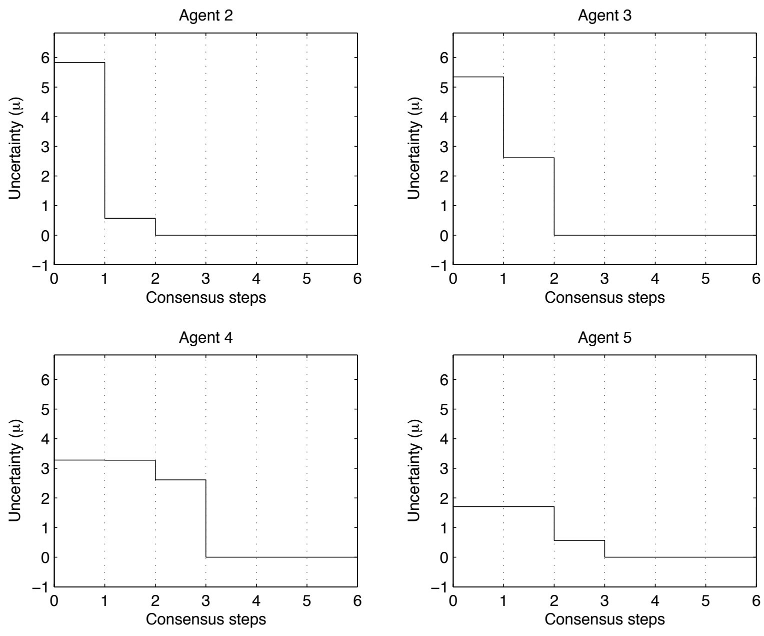


Figure 6.19: Uncertainty measure μ — Example 6.2.

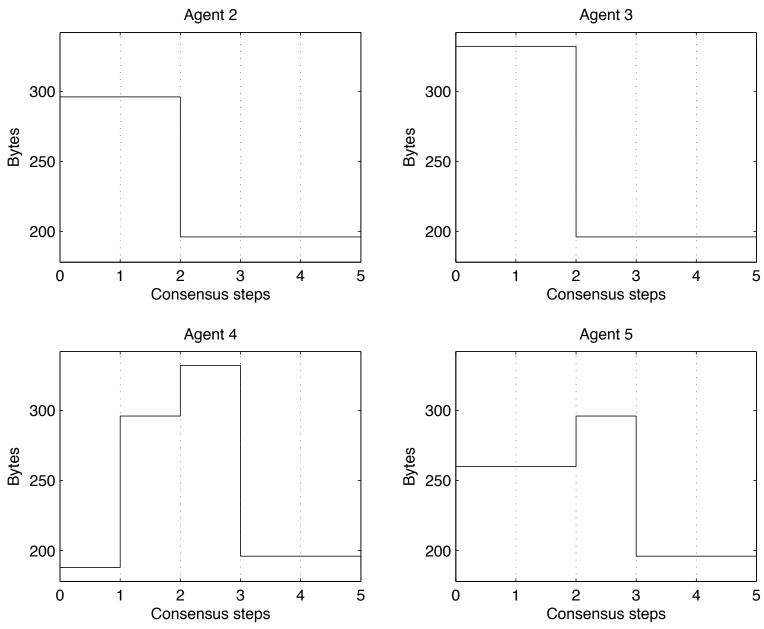
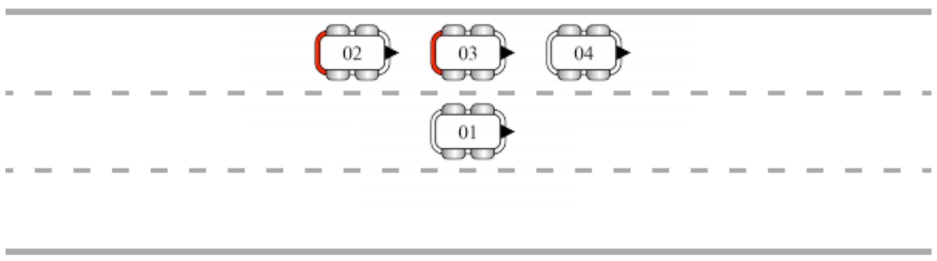
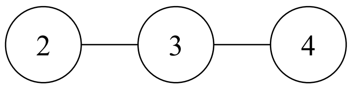


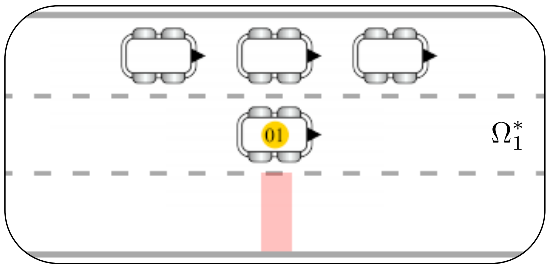
Figure 6.20: Communication overhead — Example 6.2.



(a)



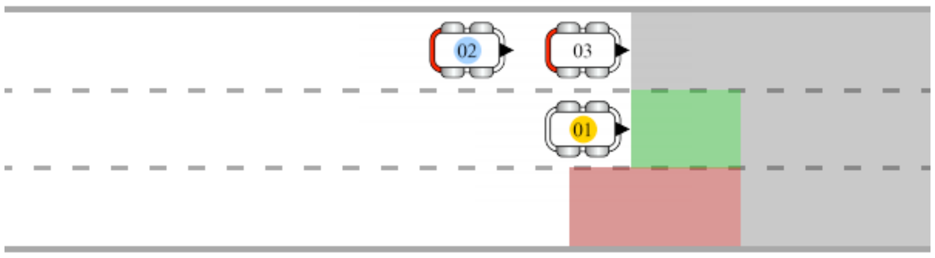
(b)



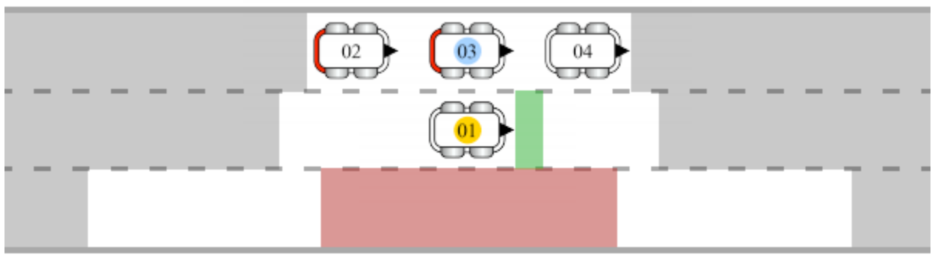
(c)

Figure 6.21: Vehicle configuration (a), communication graph (b), and centralized monitoring result Ω_1^* (c) for Example 6.3.

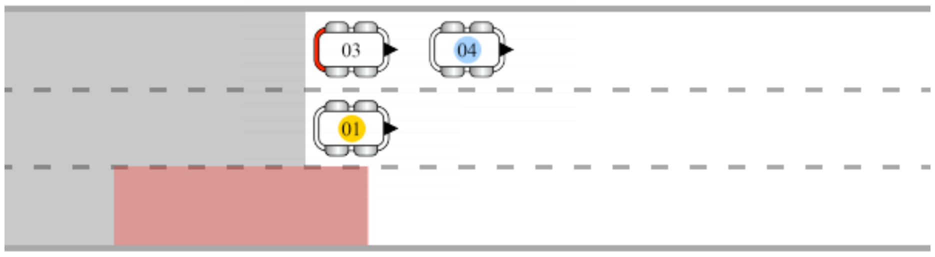
Example 6.3. Four vehicles are configured as shown in Fig. 6.21.a, and the communication graph is the same as in Example 6.1, except for vehicle 5 that in this Example does not exists. Vehicles 2, 3, and 4 are monitoring vehicle 1, and since it does not exists a Vehicle 0 preventing vehicle 1 from turning right on the first lane, vehicle 1's behavior is faulty again. Since the diameter of the graph of Fig. 6.21.b is now 2, we expect that all the values converge in at most two consensus steps. Let us note that this time the region observable by all the monitors together does not include all the target–agent's active region $\mathcal{Q}_1^a$, therefore we expect that all the monitors get to an uncertain evaluation of target–agent's reputation r_1. The initial estimations, the consensus evolution and all the results are shown in Fig. 6.22–6.28.



(a)



(b)



(c)

Figure 6.22: Vehicles' initial estimations — Example 6.3.

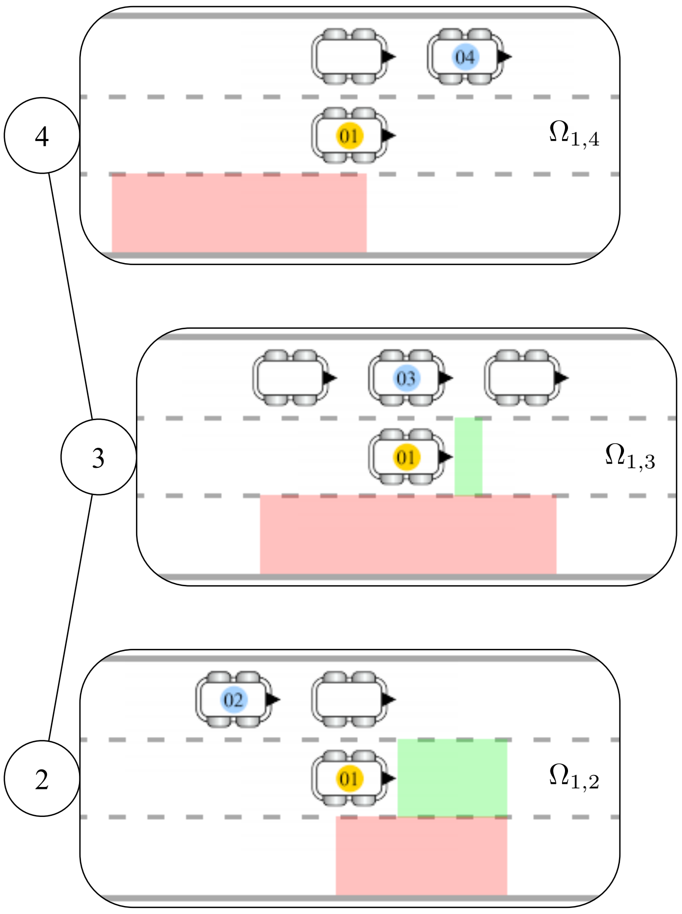


Figure 6.23: Consensus step 0 (initial conditions) — Example 6.3.

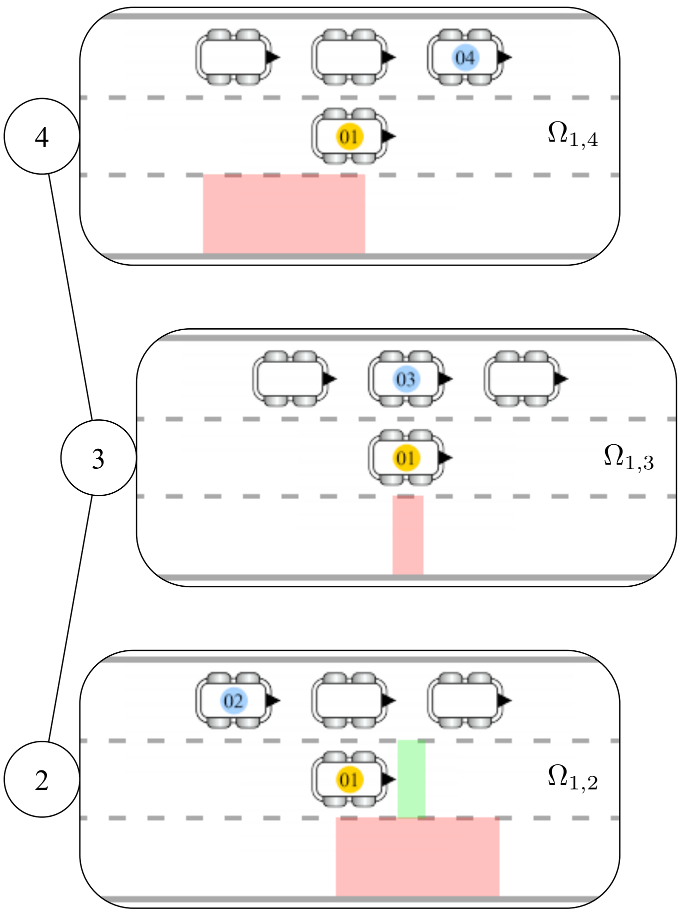


Figure 6.24: Consensus step 1 — Example 6.3.

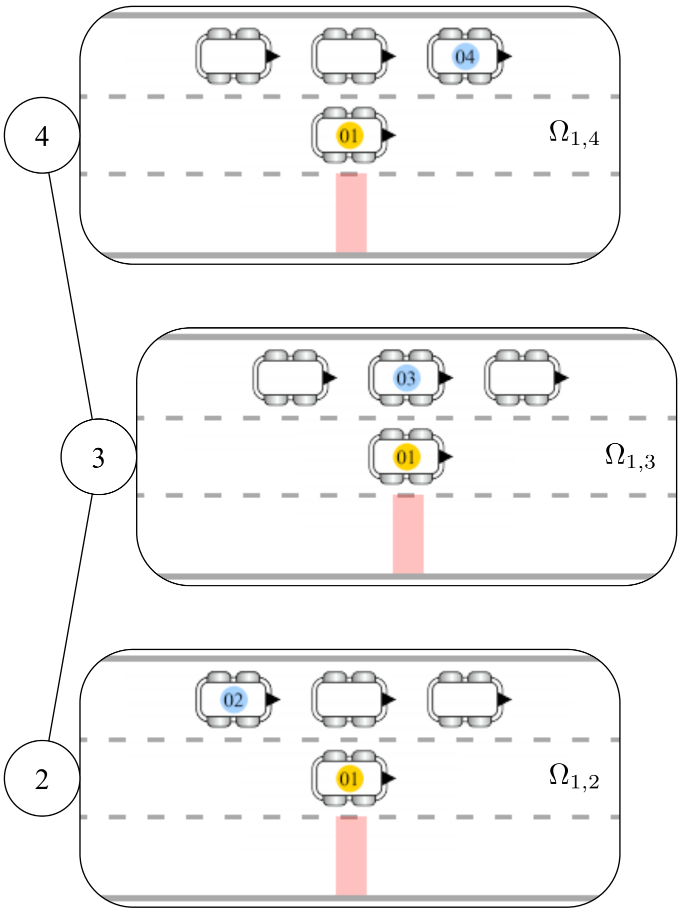


Figure 6.25: Consensus step 2 — Example 6.3.

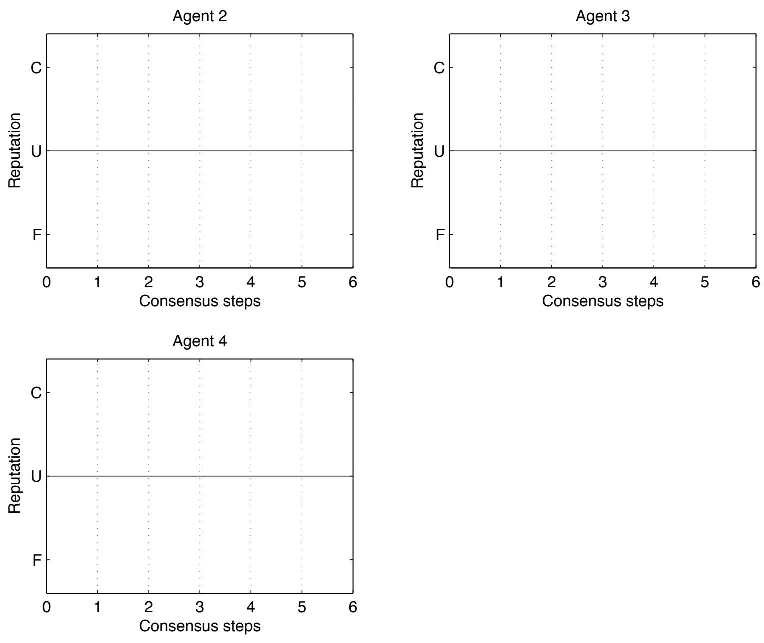


Figure 6.26: Reputation values — Example 6.3.

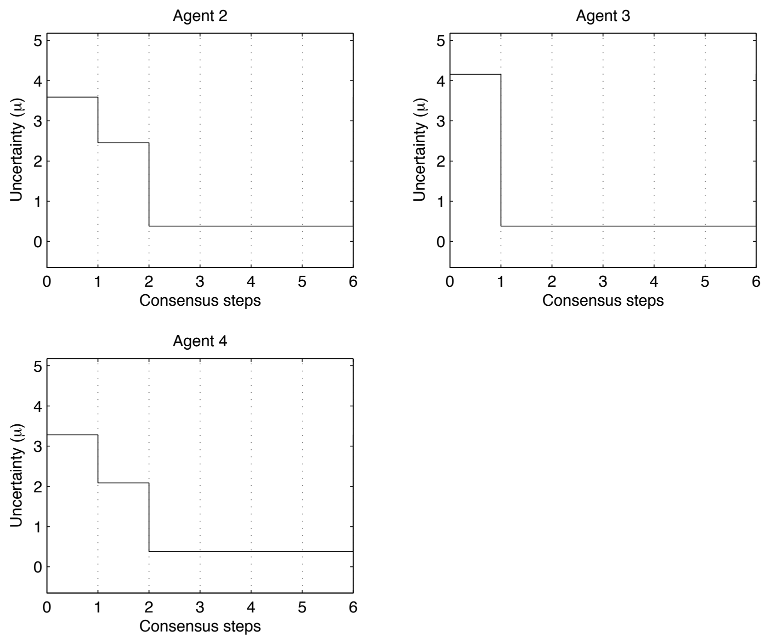


Figure 6.27: Uncertainty measure μ — Example 6.3.

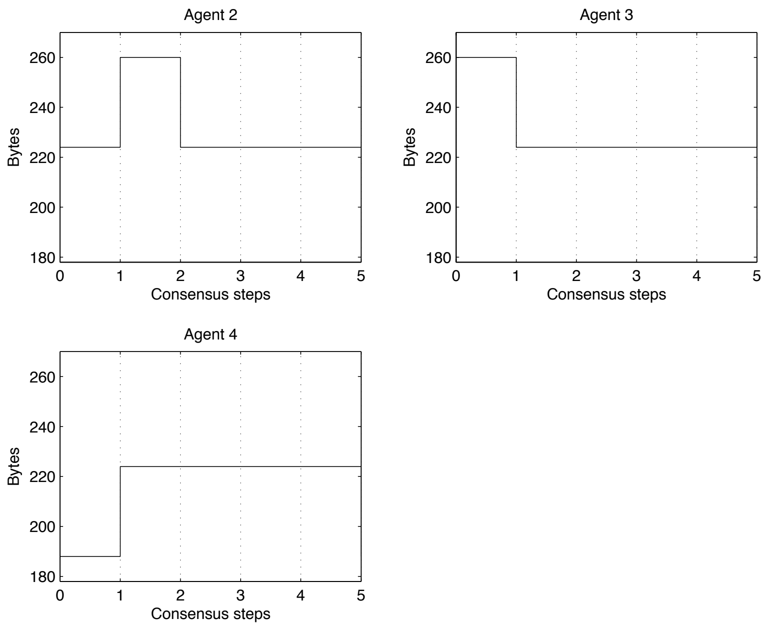


Figure 6.28: Communication overhead — Example 6.3.

Chapter 7

Conclusions

Our attention was focused on collaborative multi–agent systems in which each agent plans its behavior based on a decentralized policy, i.e. consider only its locally available pieces of information.

We described a theoretical strategy for a decentralized IDS which is not an ad–hoc solution, but instead is independent from the collaborative policy used in the system. We also described a more efficient implementation of the IDS strategy and we developed a very flexible simulator in C++ which is able to prove the effectiveness of our method.

Moreover, we improved the presented IDS by adding a *distributed algorithm* based on *consensus* and we also proved the effectiveness of the solution by a general *consensus convergence theorem* and by in-depth simulations.

Chapter 8

Publications

The ideas exposed in this thesis have been summed up and developed in the following papers:

[1] A. Fagiolini, G. Valenti, L. Pallottino, G. Dini, and A. Bicchi, “Decentralized intrusion detection for secure cooperative multi–agent systems,” in *Proc. IEEE Int. Conf. on Decision and Control*, 2007, (accepted).

[2] A. Fagiolini, G. Valenti, L. Pallottino, G. Dini, and A. Bicchi, “Local monitor implementation for decentralized intrusion detection in secure multi–agent systems,” *3rd IEEE Conference on Automation Science and Engineering*, 2007, in press.

[3] A. Fagiolini, M. Pellinacci, G. Valenti, G.Dini, and A. Bicchi, “Consensus–based distributed intrusion detection for multi–robot systems,” *IEEE International Conference on Robotics and Automation*, 2008, (submitted).

Appendix A

Multi–Agent System Simulator

The Multi–Agent System Simulator (MASS) is a software simulator developed by us in order to evaluate the effectiveness of our IDS by reproducing attacks of several kinds. Specifically, the MASS is entirely written in C++.

A.1 Smulator overview

The MASS has been developed such that each module of the IDS is represented by a particular object in the code. Therefore the MASS can be considered as a quite faithful reproduction of our proposed monitoring strategy.

A simple scheme of the MASS structure is shown in Fig. A.1. The environment is the main object which contains the whole system and which let us access its variables. Inside the environment object there is a vector of vehicles, which on their turn are composed of a physical layer, a logical layer, and a monitor layer. The physical layer and the logical layer have been described in the previous chapters. The monitor layer is a container for a list of local monitors, each of them is able to monitor a single neighboring vehicle and is instantiated or deleted as soon as a new vehicle appears or disappears from monitor layer's view.

A.2 Brief class description

Area. This class represents a region of an N-dimensional space, composed of a list of Rectangle objects. Also a few basic mathematical operations

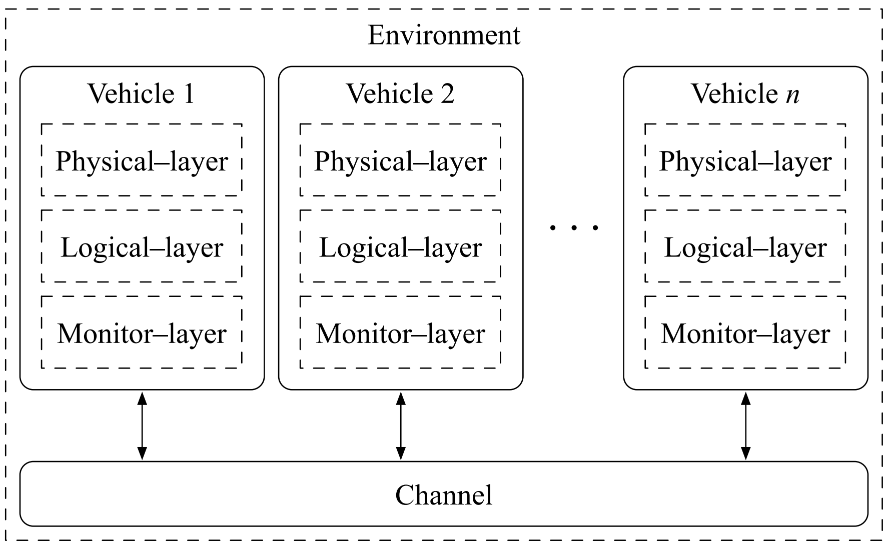


Figure A.1: A simple scheme of the MASS structure.

are defined:

- intersection,
- union,
- subtraction,
- membership.

Automaton. This class represents the automaton, which is a fundamental block of the decision–making process installed on–board all robotic agents. The automaton is characterized by a set of discrete states and a tree data structure made of a set of transitions, events, and sub-events.

Channel. This template represents a wireless channel, used by the agents of the system to communicate with each other. The Channel class provides a few basic communication services as broadcast/reception of packets. In addition, it gives a wide range of options in terms of how the channel musts work (communication range, reception probability, and so on).

Configuration. This class provides an object able to read a configuration file and store the contained information into its fields.

Environment. This class represents the scenario where the simulation is set. An Environment object contains the set of simulated vehicles, the Channel class, and provides a collection of methods to stimulate the evolution of the system and save the simulation outputs.

Event. This class represents the mathematical condition which, if verified, induce the automaton to change its logical state. A few methods to evaluate and reset the logical value of the Event are provided.

ExtValue. This class represents an extended boolean type which can assume three logical values:

- true,
- false,
- uncertain.

Some basic logical operation are also defined.

Failure. This class represent a kind of failure by which a vehicle is affected. The failure can be a *physical* or *logical* one, and the list of broken vehicles can be defined in the configuration file.

Hypothesis. This class represents a hypothesis that a monitor make on a target agent's neighborhood. A Hypothesis object contains

- the id of the probably occurred event,
- a List of SubHypothesis,
- a "negative" Area object representing a region where there should not be any neighboring vehicle.

Image. This is one of the main output class. An Image object is able to generate and save png images representing

- the omniscient vision of all vehicles,
- the subjective vision of a particular observer,
- the target–agent's reconstructed neighborhood.

List. This template represents a list of ‘T’ type elements. Such template provides several typical operations on lists:

- look for,
- head insertion,
- head extraction,
- tail insertion,
- appending,
- joining,
- element deletion,
- list deletion,
- element updating,
- sorting.

Logger. This class is used for logging. A global Logger object is instantiated in the MASS and used like a log server. It creates a log file at each simulation steps and it saves there all the received log messages.

Message. This template represents a radio message in which is sent a ‘T’ type object. Agents communicate with each other by exchanging this kind of message objects.

Monitor. This class represents the *local monitor* component. When instantiated, a Monitor object focuses its attention on a target vehicle and observes its behavior estimating its currently performed maneuver and its occurred events.

MonitorLayer. This class represents a module, installed on–board each vehicle, containing and managing all the Monitor objects, since a single Monitor is able to observe only one target. It provides a few methods e.g. for

- adding a monitoring activity for a recently appeared vehicle,
- interrogating a Monitor about a target–agent’s reputation,
- getting a target–agent’s estimated Neighborhood.

Neighborhood. This class represents the target–agent's neighborhood. Such neighborhood is estimated by the Monitor and is exchanged with neighoboring agents under the working hypothesis of virtuous communication. An object of this type is composed of:

- the target–agent's state,
- a set of measured states inside the target–agent's active region,
- a set of Hypothesis objects.

Output. This class is used to save the simulation results to an ASCII file. It is also able to organize output data into tab–separated columns in order to be easily imported in Matlab.

PhysicalLayer. This class represents the physical structure and the dynamics of a vehicle. A PhysicalLayer object, calling the provided methods, can be controlled like a continuous system.

Rectangle. This class represents a cuboid in a N–dimentional continuous space. The cuboids represented by this class are those whose edges are parallel to the direction of one of the orthonormal basis vectors. Each edge can be projected onto the direction of the basis vector with which it is aligned, and can be represented by the two bounds of its projection. Some basic operations are provided, like intersection, overlapping test, etc.

Reputation. This class represents a target–agent's reputation. In particular, a reputation level can take one of the following values:

- faulty,
- uncertain,
- correct,
- unset.

ReputationManager. This class represents the reputation manager. This is a component of the local IDS embedded on the agent's architecture. This class provides operations that allow the agent to determine the reputation of all its neighboring agents. Specifically, an object of this type bases its reputation evaluation on information supplied by the MonitorLayer object. Moreover, the ReputationManager object executes the consensus algorithm communicating with the neighboring agents.

State. This class represents a vehicle's continuous n–dimentional state.

SubEvent. This class represents the most elementary part of an Event, and provides the same evaluation methods as the Event class.

Transition. This class represents a discrete state transition in the Automaton. A Transition contains a set of Events, and it occurs only when at least one of these events occurs. This class provides all the evaluation methods as the Event and SubEvent class.

Vector. This template represents a vector of elements of type 'T'. Such template provides almost all vector typical updating, reading and sorting operations.

Vehicle. This class represents the whole vehicle. A Vehicle object contains and stimulates

- a PhysicalLayer object,
- an Automaton object,
- a ReputationManager object,
- a MonitorLayer object.

Bibliography

[1] C. Tomlin, G. J. Pappas, and S. Sastry, "Conflict resolution for air traffic management: a study in multiagent hybrid systems," *Automatic Control, IEEE Transactions on*, vol. 43, no. 4, pp. 509–521, 1998.

[2] R. Ghosh and C. Tomlin, "Maneuver design for multiple aircraft conflict resolution," *American Control Conference, 2000. Proceedings of the 2000*, vol. 1, no. 6, pp. 672–676 vol.1, 2000.

[3] L. Pallottino, V. G. Scordio, E. Frazzoli, and A. Bicchi, "Decentralized cooperative policy for conflict resolution in multi-vehicle systems," *IEEE Trans. on Robotics and Automation*, 2007, (accepted).

[4] L. Pallottino, A. Bicchi, and E. Frazzoli, "Probabilistic verification of decentralized multi-agent control strategies: a case study in conflict avoidance," in *American Control Conference (ACC)*, 2007, pp. 170–175.

[5] J. Blum and A. Eskandarian, "The threat of intelligent collisions," *IT Professional*, vol. 6, no. 1, pp. 24–29, 2004.

[6] J. Lygeros, "Lecture notes on hybrid systems," *Notes for an ENSIETA Short Course*, 2004.

[7] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems.* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[8] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. C. Teneketzis, "Failure diagnosis using discrete-event models," *Control Systems Technology, IEEE Transactions on*, vol. 4, no. 2, pp. 105–124, 1996.

[9] T. Yoo and S. Lafortune, "Polynomial-time verification of diagnosability of partially observed discrete-event systems," *Automatic Control, IEEE Transactions on*, vol. 47, no. 9, pp. 1491–1495, 2002.

[10] Y. Wang, T. Yoo, and S. Lafortune, "Decentralized diagnosis of discrete event systems using unconditional and conditional decisions," *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, pp. 6298–6304, 2005.

[11] A. Bicchi, "Dinamica e controllo dei veicoli robotici," *Lecture Notes for the course of "Robotica Industriale"*, 2003.

[12] C. M. Özveren and A. S. Willsky, "Observability of discrete event dynamic systems," *Automatic Control, IEEE Transactions on*, vol. 35, no. 7, pp. 797–806, 1990.

[13] A. Balluchi, L. Benvenuti, M. Di Benedetto, and A. Sangiovanni-Vincentelli, "Design of observers for hybrid systems," *Hybrid Systems: Computation and Control*, vol. 2289, pp. 76–89, 2002.

[14] P. Ramadge and W. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[15] C. M. Özveren, A. S. Willsky, and P. J. Antsaklis, "Stability and stabilizability of discrete event dynamic systems," *J. ACM*, vol. 38, no. 3, pp. 729–751, 1991.

[16] S. Zad, R. Kwong, and W. Wonham, "Fault Diagnosis in Discrete-Event Systems: Framework and Model Reduction," *Automatic Control, IEEE Transactions on*, vol. 48, no. 7, p. 1199, 2003.

[17] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete-event systems," *Automatic Control, IEEE Transactions on*, vol. 40, no. 9, pp. 1555–1575, 1995.

[18] S. Buchegger and J. Le Boudec, "Performance Analysis of the CONFIDANT Protocol: Cooperation Of Nodes–Fairness In Dynamic Ad-hoc NeTworks," *Proceedings of IEEE/ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC)*, pp. 226–236, 2002.

[19] ——, “A Robust Reputation System for Mobile Ad-hoc Networks,” *Proceedings of P2PEcon, June*, 2004.

[20] R. Olfati-Saber, J. Fax, and R. Murray, “Consensus and cooperation in networked multi-agent systems,” *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[21] M. DeGroot, “Reaching a Consensus,” *Journal of the American Statistical Association*, vol. 69, no. 345, pp. 118–121, 1974.

[22] W. Tutte, *Graph theory.* Cambridge University Press New York, 2001.

[23] J. Bondy and U. Murty, *Graph theory with applications.* Macmillan London, 1976.

[24] M. Barile, “Graph distance,” from MathWorld — A Wolfram Web Resource. *http://mathworld.wolfram.com/GraphDistance.html.*

[25] E. W. Weisstein, “Graph diameter,” from MathWorld — A Wolfram Web Resource. *http://mathworld.wolfram.com/GraphDiameter.html.*