Analysis and Design of Swarm-Based Robots using Game Theory

by

Sidney Givigi Jr.

A thesis

submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy in Electrical Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE)

Department of Systems and Computer Enginnering

Carleton University

Ottawa, Ontario, Canada

September 2009

Your file Votre référence
ISBN: 978-0-494-60104-4
Our file Notre référence
ISBN: 978-0-494-60104-4

The undersigned recommend to

The Faculty of Graduate Studies and Research

acceptance of the thesis

Analysis and Design of Swarm-Based Robots Using Game Theory

submitted by

Sidney Givigi Jr., M.Sc., B.Sc.

in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering

Thesis Supervisor

Professor Howard Schwartz

External Examiner

Professor Jason Gu

Chair, Department of Systems and Computer Engineering

Professor Howard Schwartz

Carleton University

September 2009

Abstract

This thesis addresses the problem of emergence of intelligence in a multi-robotic environment. Each robot is considered to be very simple in terms of hardware and software capabilities and, consequently, the kind of intelligence we are interested in is a very simple one, based only on few and simple behaviors such as the ones present in the individuals in social insect colonies. This is now known as swarm intelligence and has been applied successfully to many optimization problems. The goal is to have a large number of agents with just a few very simple rules and through their social interactions have them solve complicated tasks due to social emergent behaviors that are not directly derived from the individual behaviors of each agent. In this research we assume that the robots do not communicate directly to each other or, if they do so, it is not guaranteed that the other robot receives the message and decodes it the way intended. The main contribution of the present research to this field is the introduction of new design techniques based on Game Theory and their evaluation through experiments. Moreover, we also prove that when we represent the behaviour of robots by using personality traits, a zero-sum game between two robots will always converge to Nash equilibria. Also, we prove that in Markov games, the multiple pursuer-evader game with adaptation by learning automata converges to Nash equilibria as well.

Acknowledgements

I would like to thank some people for their help in the course of the research that resulted in this thesis.

Firstly, my wife Roberta. She has been my support in life for so long that I do not know how I would have dealt with all the difficulties I faced in the writing of this thesis without her help. She was also very understanding of all the time that I had to dedicate to this work, even when I was unable to do the dishes!

I would also like to thank my friend Jackson and his wife Tina for their encouragement. The conversations I had with them and with Gustavo and Louise were of great moral support value!

Xiaosong Lu has helped me in the execution of some of the experiments and for this I am grateful. Several fourth year Systems and Computer Engineering students have also provided very important help and deserve to be acknowledged in here.

Finally, Professor Howard Schwartz has all my gratitude. He has been part of this research since its inception through his encouragement and proposal of improvements that drove me to challenge myself at every step of the way. Now that I become a professor myself, I hope I can apply everything I learned from him, especially his honesty and moral standards, in helping my own students to produce the best work they are capable of.

Contents

List of Tables

List of Figures

List of Symbols

∇f	Gradient of function f
$\mathbb{R}$	Set of real numbers
$\mathbb{R}_+$	Set of non-negative real numbers
$\bar{x} \in \mathbb{R}^m$	A column vector of dimension m
$\bar{e}^k$ or e_k	Unit column vector with zeros in all positions except position k
$\Delta \subset \mathbb{R}^m$	The simplex set, i.e., the set $\{\bar{x} \in \mathbb{R}^m_+ \mid \sum_{k=1}^{m} x_k = 1\}$
Θ	The mixed-strategy space of the game, i.e., $\Theta = \Delta_1 \times \cdots \times \Delta_n$
$\underline{x}$	Element of the set Θ
$V(\cdot)$	Reward functions
$\mathcal{U}(\cdot)$	Reward functions dependent on the state
γ	Personality trait
η	Learning rate
$\mathcal{E}$	Payoff for each personality trait
σ	Threshold function or standard deviation
J_i	Payoff for player i in a differential game
ω	Angular velocity
$\mathbb{P}$	Probability
$\mathbb{E}$	Expected value
ι	Number of pursuers plus number of evaders

Chapter 1

Introduction

1.1 Overview

Robotics is already a very mature field in terms of mathematical and modelling tools and control theory has been successfully applied to a myriad of problems related to it. Actually, control and modelling of physical systems are the most important aspects of robotics. Models for robotic arms, vehicles, etc. are well known and several algorithms have been developed in order to operate them.

However, the field of multi-robotics has not yet been as well studied or understood. The reasons for that are mostly the complexity of modelling the interactions among the multiple robots and their relationship with their environment. Simplified models based on linear estimation have been proposed and work well for some specific applications, but when we consider more complex tasks or environments involving larger number of individual robots, the models become exceedingly complex, especially if the robots are desired to be rational and to follow several rules.

One way of avoiding such complexity is to design the robots to be simple in terms

of their rationality and capabilities. If the robots should have just a limited set of behaviours, like the ones that social insects show, their modelling would theoretically be simpler. However, in order for them to demonstrate any usual capabilities, a great number of them would be necessary. This type of assumptions and modelling techniques is known as *swarm intelligence.*

A large amount of effort has been put on the investigation of swarm intelligence. The most impressive results have been reported in the field of optimization. Little advances were reached in robotics, but it is certainly a very prolific field, and more effort is being directed towards this investigation.

1.2 The main problem and objective

The interaction among several robots in performing a task in the field is a very desirable feature of a multi-robotic system. However, as introduced in the previous section, the modelling of such a system is very complex and depends on several different control signals. Moreover, the equations are, as expected, highly coupled and nonlinear. Therefore, the tools for their analysis are limited.

One could try to simplify the equations by considering that the robots would show very simple behaviours. Moreover, one could expect that such simple behaviours would be translated in a somewhat more complex group behaviour. Such a view is, indeed, supported by the empirical observations of social insects such as ants, termites and bees.

The main objective of the thesis is to design a group (or swarm) of robots to solve some pre-specified tasks. These robots may possess some psychological traits that are referred in the course of the work by *personality traits* or simply *personalities.*

The interactions among robots may be represented as a game as defined by Game Theory. Furthermore, the choice of strategy that a robot may play at any given time will be determined by the interactions developed through the games. And, finally, in addition, robots' behaviours will be modeled using a Differential Game. In this case, real dynamics should be known and therefore modelling of real world processes must be done. Also, learning has to take place and modelling the dynamics of the learning (in terms of how the robots take decisions) must also be determined.

1.3 Thesis organization

This document consists of the following chapters:

- *Chapter 2* is a review of game theory and differential game theory. This includes what we call *Classical Game Theory* and another branch of the theory that deals with differential modelling of processes known as *Differential Game Theory.*

- *Chapter 3* is a brief review of the field of swarm intelligence in general and swarm robotics in particular. In this chapter we also discuss the problem of environment representation and introduce the idea of personalities borrowed from evolutionary psychology.

- *Chapter 4* presents the modelling of the robots and discusses the learning that must take place in order for the robots to behave properly given a task. Personality changes, reinforcement learning and learning in a cooperative game environment are discussed in this chapter, therefore, a combination of game theory and swarm intelligence takes place. Simulations are discussed in order

to provide some justification for our ideas as well as some theoretical development that justifies our research.

- *Chapter 5* introduces how a *Reinforcement Learning* approach may be used in order to solve a specific differential game by the means of a *Fuzzy Controller*. Also, we introduce the problem known a *Pursuer-Evader Game* and its more specific version known as the *Homicidal Chauffer* problem. It is then shown that the a fuzzy controller found using reinforcement learning is able to satisfactorily solve this problem. Simulations and experiments are discussed and show that the approach works.

- *Chapter 6* shows how the problem of swarm formation may be represented using differential games. Simulations are provided to show how the formulation works.

- *Chapter 7* tackles the problem of coalition formation in Markovian games. The pursuit-evasion problem is revisited and an algorithm is derived in order to solve the assignment of pursuers to evaders.

- *Chapter 8* concludes the thesis by stating the main contributions of the work.

1.4 Literature overview

In the thesis, two major bodies of knowledge will be investigated.

The first one is swarm intelligence. This term applies indistinctly to several investigation fields. The common point in them is that they deal with a very large number of entities that relate to each other. The entities may be of a variety of types that infer some analogies ranging from physics (particles or atoms) to biology (birds,

fishes or insects).

Biologists have been working on modelling swarming behaviour in animals and bacteria for a very long time. In [15], Breder introduces the idea of attractive and repulsive constant terms that would describe how aggregations should be formed. In his approach, attraction and repulsion would follow a "force" that varies with the square of the distance between the animals. A very broad investigation and description of such dynamical biological systems may be found in [81]. For a reference investigating bacterial grouping, see [22].

In physics, a series of very interesting work has also been developed. Some of them, based on biological ideas as, for example, [23]. While others focused on concepts in Physics itself [64, 60].

In computer science, the most usual one by far is the analogy to social insects. Some of the earliest references are [10, 28]. All the theories are based on the theory of self-organization [11], that affirms that collective (or group) behaviours are qualitatively (therefore, not just quantitatively) different from those that would be obtained by a single robot. The advantage of using several robots to perform a task is that some tasks are too complex or even impossible to be executed by a single robot [19]. In our implementation of swarm based robotics we also make use of personalities, introduced in [111].

In engineering, the works in swarming robotics are abundant. Most of them are based on the task of keeping a swarm group of robots (or agents) together. Some of the references are found in [36, 67, 5, 40]. This is not exacly the application that we have in mind, but we will make use of some of the stability analysis presented in them in our development.

One narrower area of swarm intelligence that we are interested in is called *swarm formation*. In engineering applications, there are some different ways of dealing with the problem. The most common approach is to determine the dynamic equations of the agents and then find a Lyapunov argument that will show the swarm is stable [36, 37]. Gazi [35] uses artificial potentials and sliding-mode control to model his swarm of agents. Olfati-Saber [82] uses heuristics to show how a multi-agent dynamic system would form a flock (or swarm in our notation). Others have tried to solve the problem by using graphs [63]. In the field of optimization, some have used the idea of a mass-spring system to model the forces between the agents [13]. Some of these approaches will be discussed in this work.

The second field is the one known as Game Theory. It was introduced by J. von Neumann and O. Morgenstern [104]. It has since been one of the major successes in the field of economics and social sciences. Its main application has been to the modelling of social interactions of rational entities for the prediction of outcomes of conflicts among them [74]. However, the applications are not only limited to economics. One of the most exciting applications, that incidentally is used in the development of this proposal, is to the Theory of Evolution [71, 108]. Another application also explored is in the military [55].

We will also focus on Differential Game Theory. Most of the applications are military and involve the pursuer-evader problem [33, 68, 84]. However, applications with evolutionary arguments [52] are also popular. References on robotics were not found by the author, except from publications of the author himself [43].

Furthermore, we shall discuss how cooperation may arise in games labeled "Markov games". In *Markov games* [103], the models of the environment and the interaction

among players are considered to be Markov chains [32]. Players are able to influence the state transitions in a Markov chain by taking actions [49]. It will be shown that using such a modelling tool will lead the system to cooperation.

We shall try in the next chapters to link both of these fields together. If we succeed, we will be able to, to some extent, predict the stability of the strategies used in the execution of pre-determined tasks.

Chapter 2

Game Theory

2.1 Introduction

Game theory may be defined as the study of mathematical models of conflict and cooperation between intelligent rational decision-makers [74]. A great deal of controversy has been generated by the use of the term *rational* in the above definition. Social scientists especially have debated the validity of the theory applied to an explanation of human behaviour [48], since humans are not inclined to behave rationally in their daily chores. However, such debate is useless for our present interests (and quite frankly, it seems to be superfluous altogether), since we will be dealing with entities (the robots) that are overtly irrational in terms of cognitive capacity. Our use of game theory is limited to the use of the mathematical techniques for analyzing situations in which individuals make decisions that influence the outcome of the others' welfare [74].

In the current chapter, we introduce the topic of game theory as a mathematical theory. Our presentation is not rigorous or formal from the point of view of a

mathematician. The chapter is structured in the following way:

- *Section 2.2* is a brief historical overview of the field.
- In *section 2.3* we introduce the main concepts involved in game theory. The section is named *Classical Game Theory* in order to distinguish it from the field of differential game theory, a branch of the modern game theory.
- *Section 2.4* is dedicated to the discussion of Differential Game Theory. The main differences between this branch and the conventional game theory are pointed out.
- *Section 2.5* introduces Cooperative Game Theory in its classical and differential meanings. Discussions on different *solution concepts* are the main part of the chapter and ways to reach them are presented.
- In *section 2.6* we discuss the application of the concepts introduced to our research topic.

2.2 History

One of the most striking consequences of game theory is that acting erratically may be (and often is) the best strategy for a rational smart player [86]. By erratically it is meant random, unpredictable behaviour.

It is generally agreed that the modern theory of games was born in 1944 with the monumental work *Theory of Games and Economic Behavior* by von Neumann and Morgenstern [104]. When it appeared, it was thought that a complete and consistent theory of rational strategic behaviour had come to be out of nothing. Soon it was

realized that it was not the case. Much more work was necessary in order to put the theory in shape.

The first step was undoubtedly taken by John Nash. In a series of four papers [76, 77, 78, 79] he unveiled the concept of Nash equilibrium, one of the touchstones of modern game theory.

John Nash was followed by John Harsanyi in the 1960's and 1970's. They were later followed by Reinhart Selten. It was no surprise that the three of them shared the Nobel Prize in Economic Sciences in 1994. The Bank of Sweden justified the prize *"for their pioneering analysis of equilibria in the theory of non-cooperative games"*.

Also in the 1960's another branch of game theory arose, the theory of differential games. It was found by Rufus Isaacs, who was working in the Rand corporation in modelling and solving military pursuit-evasion problems [65]. Actually, his work can be traced back to the late 1940's [72], but due to lack of financial support Isaac's work was just published in 1965 [55]. This theory is the most appropriate discipline for modelling conflict and problems in an analytical fashion. Research seemed to concentrate then on the extension (or derivation) of control theory problems.

2.3 Classical Game Theory

Classical or conventional game theory is the study of situations of conflict. Therefore, the situations of conflict will be called *games*. And the participants, by definition, will be named *players* [95]. It is assumed that each player wants to improve its circumstances, therefore they strive to gain as much as possible on its opponent's expense. Finally, each player has at his disposal several *strategies* or *options*, which he can exercise as an attempt to claim a portion of the available resources. Note that

the player can exercise only one strategy at a time and either in some stipulated order (the order of the game) or simultaneously (and independently of each other).

In the following subsections we shall analyze only *finite games in normal form.* In other words the game will be represented by a triplet

$$G = (I, S, \pi) \tag{2.1}$$

where I is the set of players, S is the pure strategy space and π is the payoff space.

2.3.1 Strategies and Payoff Functions

In Eq. 2.1, let $I = 1, 2, \cdots, n$ be the set of players where, obviously, $n \in \mathbb{N}$. For each player $i \in I$, we let S_i be its set (finite) of pure strategies. For simplicity, we will consider the players' pure strategies by positive integers. Therefore, the pure strategy set of any arbitrary player $i \in I$ is defined as $S_i = \{1, \cdots, m_i\}$, for $m_i \in \mathbb{Z} | m_i > 1$. A vector s of pure strategies, $s = \{s_1, s_2, \cdots, s_n\}$, where s_i is the pure strategy for player i, is called a *pure strategy profile* [108]. Hence the so-called pure strategy space is then defined as $S = S_1 \times S_2 \times \cdots \times S_{n-1} \times S_n$.

For any strategy $s \in S$, we may define a function $\pi_i(s) \in \mathbb{R}$ as the payoff associated to player $i \in I$. The payoff may have different meanings depending on the field of application of the theory. For example, in economics it is usually related to profits or losses; if the field is biology, the payoffs are associated to the gain (positive or negative) in fitness for each different strategy profile. In our robotic case, the payoffs are a little bit more complex, but we may think of them as the reward from the environment attached to the adoption of the strategy, or, in other words, as how much closer to the accomplishment of the task the strategy in question (and every one

adopted by every other robot - notice that the parameter of the payoff function is the pure strategy profile vector) took the robot. We may combine all the payoff functions for all the $i \in I$ players and define the payoff function of the game, $\bar{\pi} : S^n \rightarrow \mathbb{R}^n$. As it is clear from the definition of the function, it assigns to each strategy profile s the vector $\bar{\pi}(\bar{s}) = [\pi_1(\bar{s}), \cdots, \pi_n(\bar{s})]^T$ of payoffs.

Therefore, in Eq. 2.1, we may summarize the terms as: G is the game, I is the game's player set, S is its pure strategy space, and finally π is its combined payoff function. In the future, we will deal with the problem of the application of strategies.

2.3.2 An illustrative example

Certainly the best known game is the *Prisoner's Dilemma*. This is a two-player game in which each player has only two allowed strategies. The whole story behind the game is the following. Suppose that two people are arrested and accused of a crime. They are put in separate rooms and police officers interrogate them. Then, they have two choices (or strategies to play): *cooperate* or *defect*. By cooperation we mean that one accused person will cooperate with the other one and will not try to frame him of the crime. By defection we mean that one prisoner will put all the responsibility for the crime upon the other prisoner. If both of them cooperate with each other and do not confess the crime, they will get just a few months in jail (let's call this payoff R, reward for mutual cooperation). If the two of them defect, they will get a longer term in jail (call this payoff P, punishment for mutual defection). If one defects and the other cooperates, the defector gets no jail time (payoff T, temptation to defect) and the other gets life in prison (payoff S, sucker's payoff). In this version of the game, the objective for a player is to minimize its payoff, therefore the payoffs must

agree with the condition

$$T < R < P < S \tag{2.2}$$

Notice that in different but equivalent versions of the game, the objective could be maximize the payoff and the conditions in (2.2) would be reversed.

Let us consider a game $G = (I, S, \pi)$. The set of players is $I = 1, 2$, where player $i = 1$ is labeled player "A" and player $i = 2$ is labeled player "B". The pure strategies at each player's disposition are *Cooperate*, *Defect*. Finally, the payoffs are represented in Table 2.1. This table is the payoff in the form (Payoff for player "A", Payoff for player "B"). In it we have $T = 0$, $R = 2$, $P = 5$ and $S = 20$.

		Player B	
		Cooperate	Defect
Player A	Cooperate	R=2, R=2 Reward for mutual cooperation	S=20, T=0 Sucker's payoff, and temptation to defect
	Defect	T=0, S=20 Temptation to defect and sucker's payoff	P=5, P=5 Punishment for mutual defection

Table 2.1: Payoffs for the Prisoner's Dilemma game in the form (Payoff A, Payoff B)

The values in Table 2.1 mean that if both players cooperate they get 2 time units in jail; if one player defects and the other cooperates, the one who defected goes free and the one who cooperated gets 20 time units in jail; and finally, if both of them defect, they get 5 time units imprisonment. From the table it is easy to see that player "A"'s second strategy ("defect") yields a lower payoff than its first strategy ("cooperate"), irrespective of which strategy is used by player "B". The same thing is true for player "B". Hence, rationality leads each player to always select "defect". The dilemma from the name of the game comes from the fact that

if we consider the overall payoff $(\pi_A(\bar{s}) + \pi_B(\bar{s}))$ would be optimized if both players chose their first strategy, "cooperate". In the example just presented, $\bar{s} = \{2, 2\}$, i.e., both players play their second strategy, "defect". Consequently, their total payoff is $\pi_A(\bar{s}) + \pi_B(\bar{s}) = 5 + 5 = 10$. If they had played $\bar{s} = \{1, 1\}$, the total payoff would be $\pi_A(\bar{s}) + \pi_B(\bar{s}) = 2 + 2 = 4$, which is the overall optimum play for both players. We will come back to this when we analyze repeated games.

2.3.3 Mixed strategies

A mixed strategy for an arbitrary player $i \in I$ is defined as a probability distribution over his set of pure strategies S_i [77]. Since, that by definition for each player $i \in I$, the set of pure strategies S_i is finite, we may represent a mixed strategy $\bar{x}_i$ as a vector in a m_i-dimensional euclidean space, i.e., $\bar{x}_i \in \mathbb{R}^{m_i}$ [108], with the coordinate $x_{ik} \in \mathbb{R}$ being the probability assigned by $\bar{x}_i$ to the player's kth pure strategy.

Moreover, since the vector $\bar{x}_i$ represents probabilities we may establish that each of its components x_{ik} are positive, i.e., $x_{ik} \in \mathbb{R} | x_{ik} \geq 0$. More restrictively, we may also define that the subset $\Delta_i \subset \mathbb{R}^{m_i}$ is

$$\Delta_i = \{\bar{x}_i \in \mathbb{R}_+^{m_i} | \sum_{k=1}^{m_i} x_{ik} = 1\} \tag{2.3}$$

In the same way we defined a *pure-strategy profile*, we may also define a *mixed-strategy profile* as a vector $\underline{x} = [\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_n]^T$, where each component $\bar{x}_i \in \Delta_i$ is the mixed strategy for the player $i \in I$. Hence, we also define the *mixed-strategy space* of the game

$$\Theta = \Delta_1 \times \Delta_2 \times \cdots \Delta_{n-1} \times \Delta_n \tag{2.4}$$

In the case of mixed strategies, the payoff functions are the expectations of the players. The formulation of the problem is a linear system of inequalities and characterizes a linear programming problem. More specifically, the problem becomes linear forms in the probabilities with which the various players play their pure strategies [77]. In order to enunciate this proposition more formally, we shall first introduce some useful notation. First of all, let $u_i = \mathcal{E}(\underline{x})$ be the statistically expected value of the payoff of player $i \in I$ associated with a mixed strategy profile $\underline{x} \in \Theta$. Then, the expectation is mathematically defined as

$$u_i(\underline{x}) = \sum_{s \in S} \underline{x}(s)\pi_i(s) \tag{2.5}$$

Also note that the probability that a particular pure strategy profile $s \in S$ will be used, when a mixed strategy profile $\underline{x} \in \Theta$ is played, is simply the product

$$\underline{x}(s) = \prod_{i=1}^{n} \underline{x}_{is_i} \tag{2.6}$$

For example, lets say that our game has two players and they have just two pure strategies available and their related probabilities are $\bar{x}_1 = [0.4, 0.6]^T$ and $\bar{x}_2 = [0.7, 0.3]^T$. The probability that a strategy profile $\bar{s} = (s_1 = 2, s_2 = 1) \in S$ will be used in any arbitrary individual play is $\underline{x}(\bar{s}) = 0.6 \times 0.7 = 0.42$. Notice that we are using the assumption that both mixed strategy profiles are statistically independent [108] and that is the reason for the multiplication in Eq. 2.6.

Furthermore, let the notation $(\bar{x}_i, \underline{z}_{-i})$ be the strategy profile in which the player $i \in I$ plays strategy $\bar{x}_i \in \Delta_i$, while all other players j play according to the profile $\underline{z} \in \Theta$ [108]. Also, define the vertices of Δ_i as $\bar{e}_i^k$. Therefore, $\bar{e}_i^1 = [1, 0, \cdots, 0]^T$,

$\bar{e}_i^2 = [0, 1, \cdots, 0]^T$, and successively. This means that when $\bar{e}_i^1$ is chosen as the mixed strategy, the chance player $i \in I$ will play strategy one is 100%.

Now we may put the linear forms in the probabilities proposition in a formal mathematical notation. For any $\underline{x} \in \Theta$ and $i, j \in I$, we have

$$u_i(\underline{x}) = \sum_{k=1}^{m_j} u_i(\bar{e}_j^k, \underline{x}_{-j})\underline{x}_{jk} \leq v \tag{2.7}$$

where v is defined as the value of the game, i.e., the best payoff that may be achieved by all the players in the game (this concept will be better discussed in section 2.5).

Hence, starting from Eq. 2.7

$$\begin{aligned}
u_i(\underline{x}) &= \sum_{k=1}^{m_j} u_i(\bar{e}_j^k, \underline{x}_{-j})\underline{x}_{jk} \\
&= \sum_{k=1}^{m_j}\{[\sum_{\bar{s}\in S}(\prod_{i\neq j}^{n}\underline{x}_{is_i})\bar{e}_j^k\pi_i(\bar{s})]\underline{x}_{jk}\} \\
&= \sum_{\bar{s}\in S}\pi_i(\bar{s})[\sum_{k=1}^{m_j}(\prod_{i\neq j}^{n}\underline{x}_{is_i})\bar{e}_j^k\underline{x}_{jk}] \\
&= \sum_{\bar{s}\in S}\pi_i(\bar{s})[\prod_{i\neq j}^{n}\underline{x}_{is_i}(\sum_{k=1}^{m_j}\bar{e}_j^k\underline{x}_{jk})] \\
&= \sum_{\bar{s}\in S}\pi_i(\bar{s})[(\prod_{i\neq j}^{n}\underline{x}_{is_i})\underline{x}_{js_j}] \\
&= \sum_{\bar{s}\in S}\pi_i(\bar{s})[\prod_{i=1}^{n}\underline{x}_{is_i}] \\
&= \sum_{\bar{s}\in S}\underline{x}(s)\pi_i(\bar{s})
\end{aligned}$$

This result is important in the proof of the existence of Nash equilibrium. Although we just enunciate the definition of Nash Equilibrium, and not prove it, it is

interesting to keep in mind where it comes from.

Using the mixed strategy idea, one may replace the representation $G = (I, S, \pi)$ by $G = (I, \Theta, u)$. Let's now consider the special case for two-player games. As said before, one may use a payoff matrix to represent the results a player may expect when playing its strategies. These matrices may be labeled A and B. Therefore, for an arbitrary pair of mixed strategies $\bar{x}_1 \in \Delta_1$ and $\bar{x}_2 \in \Delta_2$, we define [108]

$$u_1(\underline{x}) = \bar{x}_1 \cdot A\bar{x}_2; u_2(\underline{x}) = \bar{x}_1 \cdot B\bar{x}_2 \tag{2.8}$$

An example is in order for the proper understanding of the concept. Considering the previous example introduced (Table 2.1), one may define two matrices of payoffs for player "A" and player "B" as depicted in Eq. 2.9.

$$A = \begin{bmatrix} 2 & 20 \\ 0 & 5 \end{bmatrix}, B = \begin{bmatrix} 2 & 0 \\ 20 & 5 \end{bmatrix} \tag{2.9}$$

Therefore, the expected payoff for player 1 is

$$u_1(\underline{x}) = \bar{x}_1 \cdot A\bar{x}_2 = (2\underline{x}_{21} + 20\underline{x}_{22})\underline{x}_{11} + 5\underline{x}_{22}\underline{x}_{12} \tag{2.10}$$

while the payoff for player 2 is

$$u_2(\underline{x}) = \bar{x}_1 \cdot B\bar{x}_2 = 2\underline{x}_{21}\underline{x}_{11} + (20\underline{x}_{21}) + 5\underline{x}_{22})\underline{x}_{12} \tag{2.11}$$

One may verify that the payoff for player $i = 1$ is minimized with $\bar{x}_1 = \bar{e}_1^2$, whatever strategy player $i = 2$ plays, i.e., for minimization, the second strategy, "defect", should always be played. For two-player games we know that $\underline{x}_{i1} + \underline{x}_{i2} = 1$,

therefore, from Eq. 2.10, we have

$$\begin{aligned} u_1(\underline{x}) &= (2\underline{x}_{21} + 20\underline{x}_{22})\underline{x}_{11} + 5\underline{x}_{22}\underline{x}_{12} \\ &= (2\underline{x}_{21} + 20\underline{x}_{22})(1 - \underline{x}_{12}) + 5\underline{x}_{22}\underline{x}_{12} \\ &= (2\underline{x}_{21} + 20\underline{x}_{22}) - 2\underline{x}_{21}\underline{x}_{12} - 20\underline{x}_{22}\underline{x}_{12} + 5\underline{x}_{22}\underline{x}_{12} \\ &= (2\underline{x}_{21} + 20\underline{x}_{22}) - (2\underline{x}_{21} + 15\underline{x}_{22})\underline{x}_{12} \\ &= (2\underline{x}_{21} + 20\underline{x}_{22}) - [2(1 - \underline{x}_{22}) + 15\underline{x}_{22}]\underline{x}_{12} \\ &= (2\underline{x}_{21} + 20\underline{x}_{22}) - (2 + 13\underline{x}_{22})\underline{x}_{12} \end{aligned}$$

We may see that $u_1(\underline{x}) = (2\underline{x}_{21} + 20\underline{x}_{22}) - (2 + 13\underline{x}_{22})\underline{x}_{12}$ is a decreasing function on $\underline{x}_{12}$. Therefore, the best strategy is always to play the "defect" strategy, or, in other words, have the probability $\underline{x}_{12} = 1$ or $\bar{x}_1 = [1, 0]^T$. The same result may be shown for player "B".

2.3.4 Nash Equilibrium

John Nash published the first proof of the existence of equilibrium points in [77]. In this two-page paper, he elegantly put forward one of the cornerstones of game theory. From [78], we extract the definition of equilibrium given by Nash:

Definition 2.3.1. *An n-tuple s is an* ***equilibrium point*** *if and only if for every i*

$$\pi_i(\bar{s}) = \max_{all\ r_i's}[\pi_i(\bar{s}; r_i)] \tag{2.12}$$

In Eq. 2.12, the notation $(\bar{s}; r_i)$ stands for $(s_1, s_2, \cdots, s_{i-1}, r_i, s_{i+1}, \cdots, s_n)$. And

the explanation of the equation is as follows. An equilibrium point is an n-tuple $\bar{s}$ such that each player's mixed strategy maximizes his payoff if the strategies of the others are held fixed. Thus each player's strategy is optimal against those of the others [78].

The importance of Nash equilibrium for the prediction of the outcome of a game cannot be overstated. The meaning of Eq. 2.12 in plain English is that if the outcome of a game does not satisfy the conditions for Nash equilibrium, then there is some individual (one or more) whose expected payoff could be improved by choosing another mixed strategy. Therefore, if we assume all individuals are rational and their outcome is not maximized, they must find their Nash equilibrium and play accordingly; and, furthermore, no change on the rules of the game is necessary for this improvement.

If we analyze Table 2.1, we see that the strategy "defect" is a Nash equilibrium (the max in Eq. 2.12 must be replaced by min). Consider the player A. Table 2.1 may be represented (for the considered player) as

$$A = \begin{bmatrix} 2 & 20 \\ 0 & 5 \end{bmatrix} \tag{2.13}$$

One may see that independently of which strategy player B adopts, the values in the second row of matrix A in Eq. 2.13 are always smaller than the values in the first row. Therefore, if $r_1 = 2$, $\pi_1(\bar{s}; r_1) < \pi_1(\bar{s})$. Consequently, strategy 2, "defect" is a Nash equilibrium.

2.3.5 Repeated games

A different situation arises when we consider repeated or iterated games. This has been largely studied by Axelrod [4]. Let's again consider the Prisoner's Dilemma problem. But at this time we will define a new payoff table (Table 2.2 according to [4]).

		Player B	
		Cooperate	Defect
Player A	Cooperate	R=3, R=3 Reward for mutual cooperation	S=0, T=5 Sucker's payoff, and temptation to defect
	Defect	T=5, S=0 Temptation to defect and sucker's payoff	P=1, P=1 Punishment for mutual defection

Table 2.2: Payoffs for the repeated Prisoner's Dilemma game in the form (Payoff A, Payoff B)

In the case described in Table 2.2, the objective is to maximize the payoff over a large finite number of plays. Observe that now, the payoffs constraints are

$$T > R > P > S \tag{2.14}$$

It may be clear for the reader that the rational strategy for only one instance of the game is not the optimal one anymore. In fact, there is no proof for a Nash equilibrium for the repeated Prisoner's Dilemma game. Axelrod proposed a competition based on this iterated game open for any person that would like to submit strategies [26]. The winner strategy was the one known as *Tit for Tat*. This is the policy of cooperating on the first move and then doing whatever the other player did on the previous move [4]. This does not mean that the *Tit for Tat* policy is the optimal one. There is no

proof of that. The only thing we know is that apparently for the repeated game, cooperation must be one of the played strategies, differently from the one instance game.

2.4 Differential Game Theory

As it has already been stated, differential game theory is a technique used to model dynamic conflicts. It may be thought as a midterm between conventional game theory and control theory. Actually, it may be considered as a hybrid of those theories [65]. And, as we shall see, the terms used in our analysis will sometimes resemble those from game theory and sometimes those from control theory.

In the next subsections we will introduce the subject and make a introductory analysis of its potential.

2.4.1 State and Control Variables

In the same way we defined the game in terms of the conventional game theory as finite games, we will formalize the mathematical description of a *finite horizon* differential game as

$$\max_{\phi_i \in R^{m_i}} \{J_i = \int_{t_0}^{T} G_i(t, \bar{x}, \bar{\phi}_1, \cdots, \bar{\phi}_N)\, dt + Q_i(T, \bar{x}(T))\},\ i = 1, \cdots, N \tag{2.15}$$

subject to

$$\dot{\bar{x}} = f(t, \bar{x}, \bar{\phi}_1, \cdots, \bar{\phi}_N);\ \ \bar{x}_0 = \bar{x}(t_0) \tag{2.16}$$

$$\bar{x} \in X \subseteq \mathbb{R}^n \tag{2.17}$$

Therefore, a finite horizon game Γ_T is defined by (2.15)-(2.16). In this game N players act over a bounded time interval $[t_0, T]$ [72]. We will postpone our discussion of Eq. 2.15 until the next subsection. Now, we shall focus on Eq. 2.16, the so-called *equation of motion* of the game. In this equation and in Eq. 2.17 we have a vector $\bar{x}$ that defines the space within which the N players can move. The vector

$$\bar{x} = [x_1, \cdots, x_n]^T \tag{2.18}$$

contains the *state variables* of the game (as it does in dynamical modelling of systems). We may assume that all the players have perfect information about the states of the game, however this is not a necessary assumption and some games may have partial and/or delayed information about the current state [65].

The evolution of the states $\bar{x}$ over time obeys the differential equation 2.16, where the vectors

$$\bar{\phi}_i = [\phi_{i1}, \cdots, \phi_{im_i}]^T \tag{2.19}$$

are *control variables* at the ith-player availability, and $\bar{x}_0$ is the game's *initial condition.* As stated in Eq. 2.15 the control signals $\bar{\phi}_i$ at time t are limited to a particular set $\Phi_i \subseteq \mathbb{R}^{m_i}$ that is constrained for each particular player.

2.4.2 Payoff Functions

As one may have already deduced, the payoffs are represented by Eq. 2.15.

In the previous section we have defined the control variables (Eq. 2.19). We may also say that each player is able to select a *control path* for its vector of control

variables, that is, a piecewise continuous function $\bar{\phi}_i(.)$ on $[t_0, T]$ satisfying

$$\bar{\phi}_i(t) \in \Phi_i \tag{2.20}$$

for all $t \in [t_0, T]$. We may now pursue a intuitive definition of payoff functions. Lets start with a definition [72].

Definition 2.4.1. *An N-tuple*

$$\phi(\cdot) = (\bar{\phi}_1(\cdot), \bar{\phi}_2(\cdot), \cdots, \bar{\phi}_N(\cdot)) \tag{2.21}$$

is called an outcome of the game if there exists a continuous and piecewise smooth function $\bar{x}(.)$ *on* $[t_0, T]$ *which is a unique solution to the initial value problem*

$$\dot{\bar{x}} = f(t, \bar{x}, \bar{\phi}_1, \cdots, \bar{\phi}_N); \;\; \bar{x}_0 = \bar{x}(t_0) \tag{2.22}$$

and satisfies the stated constraint

$$\bar{x}(t) \in X \;\; \forall t \in [t_0, T] \tag{2.23}$$

The function $\bar{x}(.)$ is the *state trajectory* generated by the control signal $\phi(.)$. With the related control and state trajectory one may derive the *payoff* for the ith-player as

$$J_i(\bar{\phi}_1, \cdots, \bar{\phi}_N) = \int_{t_0}^{T} G_i(t, \bar{x}, \bar{\phi}_1, \cdots, \bar{\phi}_N)\, dt + Q_i(T, \bar{x}(T)) \tag{2.24}$$

for $i = 1, \cdots, N$, provided the right-hand side of Eq. 2.24 is defined.

2.4.3 Strategies

In conventional game theory, a strategy is the set of decisions that a player has in its disposition, one for each situation that may arise. As we have seen, if each player chooses one strategy, the payoff is uniquely determined.

Something similar exists in a differential game. The decision at each time step t that a player must take is the choice of its control variables, $\bar{\phi}_i \in \Phi_i$ as function of the state variables $\bar{x} \in X$. When the players select their control variables $\bar{\phi}_i$ and these values are inserted in the motion equations (2.16), they become differential equations. And now, we may speculate that the paths (or orbits) will, under some circumstances, be uniquely determined (Eq. 2.24).

In order to go further, we may define the *Value* of the game. This quantity is also defined in conventional game theory and it is the best value of the payoff for the game. If we have a game with only two players, one that wants to minimize its payoff (with control variables $\bar{\phi}(\bar{x})$) and another that wants to maximize it (with control variables $\bar{\psi}(\bar{x})$), the *Value* of the game may be symbolically represented as [55]

$$V(\bar{x}) = \min_{\bar{\phi}(\bar{x})} \max_{\bar{\psi}(\bar{x})} (payoff) \tag{2.25}$$

where the min (max) extends over all allowable strategies $\bar{\phi}(\bar{x})$ ($\bar{\psi}(\bar{x})$). As in conventional game theory we might expect that

$$\min_{\bar{\phi}(\bar{x})} \max_{\bar{\psi}(\bar{x})} (payoff) = \max_{\bar{\psi}(\bar{x})} \min_{\bar{\phi}(\bar{x})} (payoff) \tag{2.26}$$

(we will not present a proof for this statement. For further discussion refer to [55]. We will not treat the difficulties on the adoption of strategies. We shall go straight

to the definition of a strategy).

A strategy for an arbitrary player i is defined as the choice of a function $\bar{\phi}_i(\bar{x})$ that may have some admissible range (constraints) and a sequence $\sigma_{t_i} = \{t_0 = 0, t_1, t_2 \cdots\}$ of increasing values of time with $\lim_{j\to\infty} t_j = \infty$. Notice that $t_1 - t_0$ is not necessarily equal to $t_2 - t_1$, although we may consider that so in this work. This type of strategy is known as a K-strategy (in honour of Samuel Karlin, who first defined such a strategy). When the i-th player plays the strategy, it finds the states $\bar{x} \in X$ to be at $\bar{x}^{(k)}$. At the interval $[t_k, t_{k+1})$, player i holds $\bar{\phi}_i(\bar{x}^{(k)})$ constant. Every other player will have at its disposition a similar strategy.

Therefore, the sequence of decisions taken by a player in a differential game is discrete, as it is in conventional game theory. The difference is that a strategy in a differential game is implemented through the control variables that will uniquely (given some conditions) alter the state of the game.

2.5 Cooperative Games

Until now we have been dealing with what is called *competitive games*. However, this does not encompass the whole field of game theory. One very important part of the field is the one that deals with how cooperation may arise in a game. This possibility has already been briefly considered in our discussion of repeated games. But we need to go a little further. In this section we will deal with some different assumptions that will give rise to *Cooperative Games*. We will then discuss cooperative games in classical notation as well as in differential notation.

2.5.1 Some Definitions

Before we state the model for a cooperative game, some definitions are necessary. However, even before we do that, we introduce some notation that is used in the next sections.

The set of all players involved in a game is N. In most applications we will be able to ordinate the players. In these cases, the set will be $N = \{1, 2, \cdots, n\}$ where we will have n players. The set of all possible coalitions will be represented as 2^N, i.e., all the subsets of N. Subsets of N will be represented by capital letters, therefore, we will write $T \subset N$. The cardinality of set T will be represented by $|T|$. Also, for simplicity, the unitary set $\{i\}$ will be represented by i whenever there is no place for confusion.

Now, going back to the definitions, the first one that we approach is *coalition*. A coalition may be seen as a binding agreement or partnership [59] among players. Based on this agreement, players decide to coordinate their efforts so they may accomplish a task with better payoffs than they would get if they played separately. Again, if we designate N as the set of players involved in a game, a coalition is a subset S of it. Therefore, the formation of the coalition S presupposes an agreement and coordination among the $|S|$ players in set S and independent of the approval of any player $i \in N \backslash S$ [59]. Also, no side agreement of any member of S with any other member of $N \backslash S$ is allowed. This means that a player cannot be in two coalitions at the same time.

Moreover, the *grand coalition* (to be better explained in the next subsection) is defined as the coalition of all players $i \in N$ acting in agreement. In order to keep the theory mathematically consistent, the *empty coalition* has also to be defined.

However, it should be clear that an empty coalition can never happen, for even if a player i does not enter in a coalition with other players, its set is i, i.e., it has one element. These coalitions (of only one player) in which a player acts alone is named a *1-person coalition* [59].

We also define a *coalition structure* as how the players in N divide themselves in mutually exclusive and exhaustive coalitions [59]. Suppose that m coalitions are formed. This structure is described by

$$\Upsilon = \{S_1, S_2, \cdots, S_m\} \tag{2.27}$$

Where the set Υ is a partition of N and has to satisfy the following conditions:

$$\begin{cases} S_j \neq \emptyset, j = 1, \cdots, m; \\ S_i \bigcap S_j = \emptyset \ \forall i \neq j; \\ \bigcup\limits_{S_j \in \Upsilon} = N \end{cases} \tag{2.28}$$

And now we define what a *payoff* is. As already discussed in the general case of game theory and differential game theory, the payoff is the outcome a player receives for playing a strategy. In the case of cooperative game theory, the strategy is related to a player being part of a coalition. The payoff is then the one given to a coalition that is then distributed among the players committed to that particular agreement. The way this division is done will be discussed later. For now, we only define how we represent the payoffs.

When the game terminates, each player $i \in N$ receives a payoff, which we denote

as x_i [59]. The payoff $x_i \in \mathbb{R}$ represents, roughly, how much the player contributed to the success of its coalition. The payoffs for all the players $i \in N$ is represented by a vector $\bar{x} = [x_1, \cdots, x_n]$, where n is the number of elements of N. This is termed the *payoff vector*. Usually it is assumed that each x_i is non-negative, i.e., $x_i \geq 0$. However in some cases this condition may be dropped in order to generalize the definition.

Now that we have already introduced some terms, we are ready to enunciate a *cooperative game*. We will limit our discussion to games in characteristic form. But, in real applications the notation changes slightly. We will always justify any changes in the standard notation.

2.5.2 Games in Characteristic Function Form

The common way to represent a game is in *characteristic function form* [59]. We are going to start by giving the general definition for such a game [30].

Definition 2.5.1. *Let $n \in \mathbb{N}$. A* **cooperative n-player game in characteristic form** *is an ordered pair (N, v),where N is the set of n elements and $v : 2^N \to \mathbb{R}$ is a real-valued set-function on the set 2^N of all subsets of N such that $v(\varnothing) = 0$.*

As before, we will designate as a player each element of N. And again observe that we can enumerate the players sequentially (assuming that they are finite) and, unless stated otherwise, $N = \{1, \cdots, n\}$.

Function v is called the *characteristic function* of the game. Therefore, if we have a subset S of N ($S \subset N$) that we will call, as introduced previously, a *coalition*, $v(S) \in \mathbb{R}$ will be the payoff for the coalition when the game is played.

Let us now try to make explicit some of the implicit assumptions that a game such the one defined in Definition 2.5.1 takes [59].

1. The players prefer to have a bigger payoff than a smaller one. As in the case of classical game theory, players are interested in increasing their payoff.

2. The payoff $v(S)$ of a coalition S is distributed among its members. Any distribution is possible assuming that all members agree on it.

3. The payoff $v(S)$ does not depend on the actions taken by the players in $N - S$. It also does not depend on the other coalitions that players $N - S$ might form. Moreover, no part of the payoff $v(S)$ may be given to any player in $N - S$.

4. The characteristic function v is known to all players. All the agreements among players are also public.

5. We may assume that all decisions on forming coalitions are taken based on the characteristic function v unless otherwise specified. In some cases, we may have some propensities so players will have some preference to form coalitions. In these cases, such preferences must be made explicit.

6. If the formation of some coalitions are unwanted, this is implemented in the definition of the characteristic function v. Another way of doing this is to define the game as a 3-tuple $(N, v, \mathcal{C})$, where $\mathcal{C} \subset 2^N$ are all the possible coalitions and $v : \mathcal{C} \to \mathbb{R}$.

Together, these assumptions make it possible for games to be described theoretically. Assumptions $(1) - (3)$ have already been discussed in the previous subsection. Assumptions $(4) - (5)$ are necessary for most models to work. And finally assumption (6) is a technical requirement, but we will mostly disregard it as unnecessary.

In the following subsection we discuss some usual ways in which cooperative games are defined. Otherwise stated, these are the way we define this kind of game in the course of this thesis.

How Games are Defined

We now define some of the assumptions that are usually taken in the field of cooperative games in order to specify how a game is played. Assumptions are necessary because for any game to make sense, the way we approach it must be stated [21]. The assumptions discussed in this thesis focus on the formation of coalitions as well as the allocation of the coalition payoff. They are by far the most common in the field.

We start by stating the *superadditivity* requirement.

Definition 2.5.2 (Superadditivity). *A game is called superadditive if given two coalitions S and T, $v(S \cup T) \geq v(S) + v(T)$, $\forall S, T \subseteq N$, such that $S \cap T = \oslash$.*

A great amount of effort has been put in the study of superadditive games. It is a desirable characteristic because players usually coordinate for the betterment of their own condition and not in its detriment [59]. However, in some cases it is desirable that it is dropped in order to reflect better the real world, including physical restrictions [3]. For example, it is possible that after a merge, two companies become less profitable than each individual one was before. In this case, the game would not be superadditive.

One assumption that has already been informally introduced is the one called *perfect information.* It is as follows.

Assumption 2.5.1 (Perfect Information). *The game is one of perfect information, i.e., all players know the moves previously made by all other players [98]. This is not*

to be confused with the assumption that one player knows the payoff function of all the others.

We also take the assumption that one player knows the characteristic function $v(\cdot)$ of all others. However, the important point in here is that all players *know* how the others acted in the past and then may use it as basis for the future. As for the payoff, it may be known just inside the coalition. This would be the case in some distributed problems that will be discussed in a later chapter.

Another assumption that has already been informally stated is the *rationality principle.*

Assumption 2.5.2 (Individual Rationality Principle). *A player has an incentive to participate in a coalition only if its payoff is increased. Mathematically, if x_i is player i's payoff allocation,*

$$x_i \geq v(i)$$

where i is short for $\{ i \}$.

Therefore, the payoff each coalition gives to each one of its participant has a *lower bound.* So, for a player $i \in N$, if the value drops beyond a certain value (the payoff $v(i)$), the player automatically leaves the coalition.

It is also desirable that a coalition is *efficient.*

Definition 2.5.3 (Efficiency Principle). *The payoff for the coalition must be completely divided among the players in the coalition. In other words, no payoff is wasted. Mathematically, $v(S) = \sum_{i \in S} x_i$.*

And we also formally state an assumption regarding the grand coalition.

Assumption 2.5.3 (Grand Coalition). *The payoff for the grand coalition (N) is greater than or equal to the payoff for any other coalition, i.e.,* $v(N) \geq v(S), \forall S \subset N$.

However, observe that the individual payment is not necessarily the largest in the grand coalition. Therefore, the players have an incentive to keep trying to find a better payoff.

Before we introduce an example to clarify the ideas discussed so far, we need to define two extra concepts: *sidepayments* and *payoff configurations.*

In cooperative games a sidepayment is a transfer of payoff from one member of the coalition to another [59]. In the simplest possible game, the payoff is equally distributed among all players of the coalition so that individual payoffs are $\frac{v(S)}{|S|}$. However, this does not reflect all the possibilities of the game. A player may have an *upper-hand* in a dispute and its payoff should be greater than the other ones. But the procedure is not without difficulties, for it usually assumes that the utility is transferable, which may not be the case. Therefore, some works are based on a more complex definition of payoff functions. When we make use of a model based on cooperative games, we shall determine *ad hoc* in which way the payoffs are transferable (if this is the case).

Payoff configuration is how we express the general payoff of the game. It has the following format

$$(\bar{x}; \Upsilon) = (x_1, \cdots, x_n; S_1, \cdots, S_n)$$

With these definitions, let us discuss a brief example.

Example - A Small Market

This example was introduced by von Newmann and Morgenstern [104] and the presentation given here is based on the account of Kahan and Rapoport [59].

Let us say we have three players. Player 1 owns a house which is worth a to him and he is willing to sell it. Player 2 and 3 are prospective buyers who would pay respectively b and c for the house, where $b \leq c$ without loss of generality. Let us also assume that $a \leq b$.

Now, we define the characteristic function as the profit of each player in the case of a deal. If the players do not reach a deal, there is no profit to be distributed and $v(\{1\}) = v(\{2\}) = v(\{3\}) = 0$. As mentioned before, we will drop the brackets and represent $v(\{i\})$ by only $v(i)$.[1] Therefore we have $v(1) = v(2) = v(3) = 0$. Now, if the seller (player 1) reaches a deal with buyer player 2, the ownership of the house changes and a price p_{12} is paid by it. We may assume that $a \leq p_{12} \leq b$. And player 1 profits $p_1 = (p_{12} - a)$ and player 2 profits $p_2 = (b - p_{12})$. The joint profit is then

$$\begin{aligned} v(12) &= p_1 + p_2 \\ v(12) &= (p_{12} - a) + (b - p_{12}) \\ v(12) &= b - a \end{aligned}$$

[1] Also the payoff $v(\{1,2\})$ will be represented as $v(12)$ and so forth.

Similarly, when buyer 3 reaches a deal with the seller, 1, we have

$$\begin{aligned} v(13) &= p_1 + p_2 \\ v(13) &= (p_{13} - a) + (c - p_{13}) \\ v(13) &= c - a \end{aligned}$$

And since players 2 and 3 both are buyers, they can never reach a deal among themselves and $v(23) = 0$.

Another possible situation involves a three-way deal. Let us suppose that player 3 is to pay some money to player 2 to give up the deal and then negotiate freely with player 1 for the house. In this case, the sidepayment (bribe) paid to 2 has to be less than $b - a$ which is the upper bound for player 2's profit. This is so because any offer greater than $b - a$ made to player 2 by player 3 would be disadvantageous for player 3 and he would be better off negotiating directly with player 1. Also, for the same reason, the sidepayment to 2 plus the price paid to 1 for the house cannot exceed c, the total money player 3 has. If p is the price paid for the house, q the bribe paid to 2, the grand coalition $\{1, 2, 3\}$ would have the payoff

$$\begin{aligned} v(123) &= (p - a) + (q) + (c - p - q) \\ v(123) &= c - a \end{aligned}$$

Therefore, the characteristic function is

$$\begin{cases} v(1) = v(2) = v(3) = 0 \\ v(12) = b - a; v(13) = c - a; v(23) = 0 \\ v(123) = c - a \end{cases} \tag{2.29}$$

With these results, we may conclude that the game is supperadditive (see Definition 2.5.2) and efficient (see Definition 2.5.3). Moreover, the payoff configurations for each one of the possible coalitions is as follows

$$\begin{cases} (p - a, b - p, 0; 12, 3) \\ (p - a, 0, c - p; 13, 2) \\ (p - a, q, c - p - q; 123) \\ (0, 0, 0; 1, 2, 3) \end{cases} \tag{2.30}$$

2.5.3 Solution Concepts

A solution concept tries to describe the coalitions that will be formed as well as the coalition formation for a cooperative game. It is intimately related to the concept of a stable set and it is very important for the realization of possible outcomes of the game.

The concept of a stable set was formulated by von Neumann and Morgenstern [104]. They proposed that we consider not individual payoff vectors $\bar{x} = [x_1, \cdots, x_n]$ (also called imputations), but sets of them. But first, it is necessary to define *domination* in the context of cooperative games [98].

Definition 2.5.4 (Domination). *An imputation* $\bar{x} = [x_1, \cdots, x_n]$ *dominates an imputation* $\bar{y} = [y_1, \cdots, y_n]$ *if there is some coalition* S *such that*

i. $x_i > y_i \; \forall i \in S$ *and*

ii. $\sum_{i \in S} x_i \leq v(S)$.

Now, a stable set would be.

Definition 2.5.5 (Stable Sets). *A stable set for a game* (N, v) *is a set of imputations* Φ *such that*

i. Φ *is internally stable: no imputation in* Φ *is dominated by any other imputation in* Φ*;*

ii. Φ *is externally stable: every imputation not in* Φ *is dominated by some imputation in* Φ.

Therefore, we may hypothesize that starting from an imputation not in Φ we will end up in some imputation that is in Φ in the case Φ exists, which is not guaranteed.

Starting with this concept, we will enunciate two common solution concept, the *core* and the *Shapley value* [21, 93]. These are broadly used in the literature. However, some problems require the use of other solution concepts. Thus, we will always make explicit which concept will be used.

The Core

The core is the set of all undominated imputations. It was suggested as a powerful concept for games in characteristic form. Its formal enunciation is

Definition 2.5.6 (Core). *An allocation lies in the core of a cooperative game if it is efficient and is such that for all $S \subset N$, we have that $\sum_{i \in S} x_i \geq v(S)$.*

One point that has been largely discussed is that the core may be an empty set. However, for a large number of games it is not empty. If an imputation is in the core and if the players play accordingly, there is no incentive for them to change the way they play. An example may clarify the concept [98].

Consider the game

$$\begin{cases} v(1) = v(2) = v(3) = 0 \\ v(12) = \frac{1}{4}; v(13) = \frac{1}{2}; v(23) = \frac{3}{4} \\ v(123) = 1 \end{cases} \tag{2.31}$$

For an imputation $[x_1, x_2, x_3]$ to be in the core, it has to satisfy the following conditions

$$\begin{cases} x_1 + x_2 \geq v(12) = \frac{1}{4} \\ x_1 + x_3 \geq v(13) = \frac{1}{2} \\ x_2 + x_3 \geq v(23) = \frac{3}{4} \\ v(123) = 1 \end{cases} \tag{2.32}$$

Therefore, for example, the imputation $[\frac{1}{4}, \frac{1}{2}, \frac{1}{4}]$ is in the core!

There are several practical ways to find the core [98], but they are dependent of how we model the problem (the types of the inputs, the range of the payoffs, etc.). We are not going to pursue these in detail in this thesis.

Shapley Value

The Shapley value is a solution concept that is approached from a different point of view. Instead of asking which results are possible (stable sets or core, for example), we may ask if there could be an imputation that would be the *fairest* distribution of the payoffs. The general answer to this question came to be known as the *Shapley Value* (in honour of Lloyd Shapley, who proposed the solution). He came to the result from an axiomatic path. He suggested three axioms which he claimed captured the concept of a fair distribution of payoffs. In order to be clear, let us state what is the result we want to achieve. In an n-player cooperative game as defined above, we want to assign an imputation $\bar{\phi} = [\phi_1, \cdots, \phi_n]$ that is fair to all players. The axioms Shapley started with are [98]

Axiom 2.5.1. *$\bar{\phi}$ should depend only on v and should respect any symmetries in v. That is, if players i and j have symmetric roles in v, then $\phi_i = \phi_j$.*

Axiom 2.5.2. *If $v(S) = v(S - i)$ for all coalitions $S \subseteq N$, that is, if player i does not add any value to any coalition, then $\phi_i = 0$. Also, a player like this does not change the value of ϕ_j for other players j in the game.*

Axiom 2.5.3. *Consider two games (N, v) and (N, w) over the same set of players N but with different characteristic functions v and w. Suppose that the fair payoff distribution for each game is represented by $\bar{\phi}[v]$ and $\bar{\phi}[w]$ respectively. If we now define the sum game $(N, v + w)$ where the characteristic function is $v + w$, the fair payoff distribution for this new game is*

$$\bar{\phi}[v + w] = \bar{\phi}[v] + \bar{\phi}[w]$$

With the definition of these axioms, Shapley proved the following theorem [92].

Theorem 2.5.1 (Shapley Value). *There is only one method of assigning an imputation $\bar{\phi}$ to a game (N, v) which satisfies the three axioms above and it is given elementwise by*

$$\phi_i = \sum_{S \subseteq N} \frac{(n-s)!(s-1)!}{n!}[v(S) - v(S-i)]$$

where $s = |S|$.

2.5.4 Stochastic Cooperative Games

So far, we have considered only the case of games (N, v) in which the characteristic functions v are deterministic.

However, this is not necessarily the case. Actually, for robotics applications, we do not know with certainty the payoffs for a robot. This is due to several circumstances. First of all, the nature of the payoffs in real life are more commonly than not stochastic. Also, there are uncertainties in the modeling of the robots and the environment that would be better modeled with a stochastic representation. Finally, the relationships among robots and the relationships of robots with the environment present themselves in a stochastic fashion.

Therefore, a possible definition for cooperative stochastic games is.

Definition 2.5.7. *Let $n \in \mathbb{N}$. A* **stochastic cooperative n-playser game in characteristic form** *is an ordered pair (N, V),where N is the set of n elements and $V : 2^N \rightarrow \mathbb{R}$ is a stochastic-process on the set 2^N of all subsets of N.*

In here, we are only going to state the definition of a stochastic cooperative game without focusing on the supporting assumptions and definition of solution concepts. When this is necessary, we are going to state the concepts for further investigation.

Stochastic Cooperative Differential Games

As we have done with stochastic cooperative games in general, we are only going to state what a stochastic cooperative differential game is [110].

Definition 2.5.8. *Consider a general n-person differential game with state dynamics*

$$\dot{\bar{x}} = f(t, \bar{x}, \bar{\phi}_1, \cdots, \bar{\phi}_N); \;\; \bar{x}_0 = \bar{x}(t_0)$$

$$\bar{x} \in X \subseteq \mathbb{R}^n$$

and payoff for player i as

$$\int_{t_0}^{T} G_i(t, \bar{x}, \bar{\phi}_1, \cdots, \bar{\phi}_N)\, dt + Q_i(T, \bar{x}(T))\}, \; i = 1, \cdots, N$$

A stochastic cooperative n-person differential game in characteristic form is the one in which the solution principle includes

i. an agreement on a set of cooperative strategies and

ii. a mechanism to distribute total payoff among players.

Moreover, $G_i(.)$ and $Q_i(.)$ may be consider as stochastic processes.

Note that the definition for a cooperative differential game would be virtually the same without the last statement about $G_i(.)$ and $Q_i(.)$.

Here we also do not make extra statements about solution concepts and mechanisms of distribution of payoffs. When these make themselves necessary, we will define them in the context of the application.

2.6 Perspectives

Game theory may be used in the modeling of conflict and cooperation. Since the multiple robots problem also involves conflict and cooperation, it seems that the application of this mathematical tool would fit the representation of the tasks.

However, some problems arise when one tries to apply game theory to robots. The first one is the one of learning. Since the representation of the environment, the representation of the other players and the payoffs that each task would give to each robot are not very clear, learning becomes challenging. Also, it is not very straightforward to model the robots' relationships by games. Sometimes, more than one game is necessary for this modeling and the solutions may very well spiral out of control.

In chapters to come, we will use game theory to model our problems and also to propose solutions to them. When necessary, we will adapt the notation accordingly so the statement of the problem is more clear. Game theory will be used throughout this thesis to solve several different multiple robotic problems. Classical game theory will be used in chapter 4 to model conflicts among robots and also in chapter 7 as an auxiliary technique in the solution of the cooperation problem. Differential game theory will be used in the modeling and solution of the problems discussed in

chapters 5 and 6. And cooperative game theory will be used as the main solution tool in chapter 7. Simulations and experiments will show the feasibility of the use of the techniques.

Chapter 3

Swarm Intelligence

3.1 Introduction

Swarm intelligence could be defined as the study of the social insect metaphor for solving problems in computer science or engineering [9]. It is a very powerful idea and a very interesting topic of research.

In this chapter, we study some of the ideas involved in the subject. We start by discussing the evolution of swarm intelligence and swarm-based robotics. We continue by discussing the necessity of the representation of the environment. Thereafter we introduce the concept of personality traits as it is applied to swarm-based robotics. And we end the chapter by discussing some problems with the technique and also the applications of swarm intelligence in this thesis.

This chapter is organized in the following manner:

- In *section 3.2* we briefly discuss the historical evolution of swarm intelligence.
- *Section 3.3* is dedicated to the topic of representation of the environment. We

limit ourselves to the discussion of just one possible representation technique known as the *potential field representation.*

- *Section 3.4* introduces the idea of personalities and its application to a swarm-based robotic environment.
- Finally, *section 3.5* points to the future use of the ideas presented in the whole chapter.

3.2 The evolution of swarm intelligence

Swarm intelligence grew out of the observation of social insect colonies. It emphasizes the distributedness and direct or indirect interaction among (relatively) simple agents [9]. The approach is also designed in order to be flexible and robust, for a large number of agents will make the whole system resistant to individual failures.

The first and main application of swarm intelligence is in combinatorial optimization. It may be thought of as an alternative to classical approaches and, more specifically, genetic programming. Therefore, it is also successfully applied to communication networks [9]. A more exciting approach is the trial of achieving artificial intelligence based on simple agents - a type of collective intelligence. In this sense, intelligence is perceived as the capacity of solving problems and not pure rationality.

One of the main ideas of swarm intelligence is the small amount of (formal) communication among the many individuals in the society. The concept is that the individuals would be able to get information directly from the environment just by observing the behaviour of the other individuals. This was named stigmergy (from the Greek *stigma*: sting, and *ergon*: work) [44]. The implications of this idea are

enormous. First of all, the need for the bandwidth of communication among the agents is largely reduced. Second of all, the representation of the environment turns out to be very important, for it is through this representation (and not the world itself) that the robot communicates. And finally, when we walk towards a robotic representation, sensors become a key aspect of the implementation.

In robotics, the key advantage of swarm intelligence is its simplicity. Since hardware has advanced so much in the past decade or so, it becomes possible to implement simple robots and test them in operation. Most important is the possibility to implement the algorithms in very simple chips, such as FPGA's.

In this thesis we deal mostly with the approach of swarm intelligence applied to robotics. We try to understand the nature of cooperation among them and try coordination algorithms that will provide a basis for future development. We did not build the robots, but simulation is provided in order to support our ideas.

3.3 Representation of the environment

The way the environment is represented has a great influence on which techniques may be used to control the robots as well as in how the payoff tables used by Game Theory are implemented. In order to reduce the robots computational requirements, the environment is represented by potential fields in a technique known as Co-Fields [69].

Co-fields are based upon cooperation through social potential fields [20]. Each robot is considered to be a particle with a given position in a fixed time. The environment, containing the other robots, obstacles and enemies, is represented individually by potential fields. The potential fields represent attraction or repulsion among the

objects. In the approach presented in here, the meaning of the field is determined by the robot through the adaptation of personality traits as discussed in next section. For now, we may say that the algorithm the robot is running might prefer a down-hill - where the robot looks for a low potential valley within its representation - or an up-hill - where the robot pursues a peak - approach depending on the task it has and the personalities it developed.

For each robot, according the potential field created by any object it perceives, we may write the "force" [12] acting over it by

$$\vec{F}_T = \sum_{attractive} \vec{F}_i + \sum_{repulsive} \vec{F}_j \tag{3.1}$$

where, again, the attractive and repulsive forces are defined differently for each robot. As a result of that, each individual robot perceives the world in a potentially very different way of the others, much like people. In Fig. 3.1 it is shown the actual configuration of the world and the ways each one of two robots perceives it. As can be seen, robot A does not know anything about robot B, who, on the other hand, has a representation of its own state (position) that does not fit the actual data and the other three objects are represented as a cluster (in the form of a big attractor) and not individually. The differences in the representations may result from the learning of new traits of personality or from noise in the sensors. Either way, the robot will have some outcome from its reading. Even if it means it will do nothing, this will come as a result of a computation. In Eq. 3.1, $\vec{F}_T$ is the resultant of the individual forces (attractive and repulsive) acting on the robot.

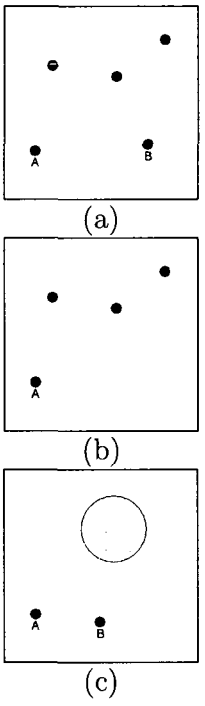

Figure 3.1: Actual configuration of the world (a). The way robot A perceives it (b). The way robot B perceives it (c).

The direction of the robots movement is the direction of the resultant force. Observe that the force may be represented as

$$\vec{F_T} = |F_T|\vec{d_F} \tag{3.2}$$

where $\vec{d_F}$ is the unit vector in the direction of the resultant force. In other words it is

$$\vec{d_F} = \frac{\vec{F_T}}{|F_T|} \tag{3.3}$$

We use this notation in order to emphasize that the fields just define the direction

of the movement, but the final decision to move in that direction and the velocity of the movement is determined by the personality traits as defined in next section. The goal of the robot (equalize the sum of forces or get to a maximum/minimum value of them) is highly dependent on the task it has to perform.

3.4 Swarm-based robotics in terms of personalities

One of the suggestions of the Theory of Evolution is that animals have emotions [25]. Moreover, these emotions are shared by the same species. Also, traits of personality (term that is used interchangeably with emotions) are important to the maintenance of objectives and collaboration [75]. Using these ideas, we can define a way robots may react to their environment [57, 111].

In our problem, the robots are assumed, initially, to be homogeneous in configuration and capabilities (understood as the set of traits of personality available to each one of them). However, like ants in a colony, they should differ from each other in order to better perform a complex task. But we do not want to add complexity to our algorithms. In order to solve this dilemma, we make use of personality traits. Therefore, the algorithms are the same for every robot, but changing these numerical values (the "traits") will change the behaviour of individual robots. Although simple, this idea is very powerful and joined with reinforcement learning may lead to a heterogeneous population in which some robots are able to specialize in executing certain tasks but at the same time they may learn how to execute a different action if it is necessary.

Personality traits are represented by real numbers γ_i. They are used in order to represent the individual intentions when faced with changes in the environment. At

each time t, the robot may take one action α_i chosen from a set of possible actions A. If we define X as the state of the robot and Y as the state of the environment, we may establish a function $f : X \times Y \times A \rightarrow X \times Y$, i.e., a function that maps the current states of a robot and the environment to different states through the execution of an action. The problem then becomes to determine a way to represent the states X and Y and relate them to the actions A. Clearly, if we increase the number of represented states and possible actions; the number of possibilities for the function f increases exponentially. This could be regarded as the main problem when a symbolic representation of the world is used.

The choice of actions is made considering the traits of personality a robot has and the payoffs related to each action at a given moment in time. By payoffs we mean the reward or penalty that may come from executing a specific action. For example, if by performing an action the robot gets closer to its objective, it receives a reward (a positive number that represents the possible success of the action); on the other hand, if as a consequence of an action the robot is damaged or increased its risk of being damaged, it receives a penalty (a negative number that represents the failure of the action). Notice that the payoffs may change according to a change in the representation of the environment. We may give a human example to explain that. In our diet we usually would not consider eating worms. However, if we were lost in a jungle, we would do anything that would keep us alive. In the same way, a robot can change the values of its payoffs if the environment as it perceives it changes.

Let us make some definitions.

Definition 3.4.1. *For a player i, we define the personality traits to be*

$$\bar{\gamma}_i = [\gamma_1, \cdots, \gamma_n]^T \tag{3.4}$$

the reward functions defined for each one of the n personality traits are represented by the vector

$$\bar{\mathcal{E}}_i = [\mathcal{E}_1, \cdots, \mathcal{E}_n]^T \tag{3.5}$$

However, in this section we are going to drop the subscript i, since the algorithm to be introduced in the next chapter will be explained for just one robot.

The reward functions $\mathcal{E}_j$ represent how well a personality trait contributed to the success of the robot. These functions are arbitrary and defined based on the problem under consideration [111]. When an action α_k is chosen to be executed, all personality traits $\bar{\gamma}$ are then updated.The effect of the action taken is evaluated using the equation

$$\mathcal{U}(s, \bar{\gamma}, \alpha_k) = h(\sum_{j=1}^{n} \gamma_j \mathcal{E}_j(s_t, \alpha_k, t)) \tag{3.6}$$

where the individual reward functions $\mathcal{E}_j$ are related to how beneficial for the robot the execution of action α_k in the presence of state s_t is according to each personality trait γ_j and, therefore, determines the reward and/or penalty related to the trait of personality. Function $h(\cdot)$ is a suitably defined function that weight the cost function inside the summation in a way particular to each problem under consideration. Notice that the reward functions $\mathcal{E}_j$ are some feedback from the environment. Since there could potentially be more robots acting on the environment, the reward that one

particular robot gets depends indirectly on the actions taken by the other agents present. Furthermore we assume that the weights γ_j (i.e., the personality traits) are normalized, so

$$\sum_{j=1}^{n} \gamma_j = 1; \gamma_j \geq 0 \tag{3.7}$$

However notice that the personality traits vector $\bar{\gamma}$ is not a probability vector. In other words, γ_j is not the probability of the robot to take action j. Indeed, the dimension of the personality traits vector and the number of actions are, in general, not the same.

Equation (3.6) takes into account all the traits of personality γ_j and the environment as represented at time t (the current time step). Moreover, action α_k is the action under consideration and s_t is the state that the robot perceives the environment to be in at time t.

Furthermore, some procedure for evolving the personalities is necessary. The main purpose of that is to diminish the dilemma between stability and adaptability (plasticity) of learning.

The dynamics of the personality vector are described by the following general difference equation

$$\bar{\gamma}_{t+1} = \bar{\gamma}_t + \eta \bar{F}(\bar{\gamma}_t, \bar{\mathcal{E}}_t) \tag{3.8}$$

where the function $\bar{F}(\bar{\gamma}_t, \bar{\mathcal{E}}_t)$ depends on the application under consideration.

Equation (3.8) implies, because the utility function is a function of all personality traits, that each trait of personality influences the others (notice that the term $\bar{F}(\bar{\gamma}_t, \bar{\mathcal{E}}_t)$ may include the reward functions for all the other personality traits). Therefore, changing a single trait will affect all the others. Furthermore, since the utilities

include the plays of all other players (as explained above), the utilities and plays of other players also influence how the personality traits for one robot changes.

Equation (3.8) can be characterized by an ordinary differential equation. This is done in order to prove convergence of the algorithm. We may rewrite (3.8) as

$$\frac{\bar{\gamma}_{t+1} - \bar{\gamma}_t}{\eta} = \bar{F}(\bar{\gamma}_t, \bar{\mathcal{E}}_t)$$

If we let $\eta \rightarrow 0$ we have

$$\dot{\bar{\gamma}} = \bar{F}(\bar{\gamma}, \bar{\mathcal{E}}) \tag{3.9}$$

Notice that this approximation may be very convenient in representing the process under study, but when we implement the algorithms we use (3.8).

For sake of completeness, we discuss here the ideas above tied with the representation of the environment. Since the states of the system are represented using potential fields as described in the previous section, we need to avoid the proliferation of actions that the robot can take. For example, in a search and rescue application, the robot would choose only between an up-hill or a down-hill algorithm (plus the choice to not move). One thing that should be stressed here is that the up-hill or down-hill move does not need to be on the exact direction of the gradient, but, according to the personality traits will vary, keeping only the orientation of the movement. Then, when the state is recognized, the action could be chosen according to the formula, known as the randomized strategy [58], which is useful for leading a robot to *explore* new actions and not just *exploit* learnt sequence of actions. The next equation

demonstrates how a randomized strategy would look.

$$P(\alpha_i|s) = \frac{k^{\mathcal{U}/T}}{\sum_{j=1}^{n} k^{\mathcal{U}(s,\bar{\gamma},\alpha_j)/T}} \tag{3.10}$$

In Eq. 3.10 P is the probability that action α_i will be executed when the state is s. The term k is a coefficient that defines how often the robot will explore new solutions or exploit the ones it already knows as better ones. When k increases, the probability that the robot will explore new choices decreases and vice-versa. T is a *temperature* parameter inspired in *Boltzmann* theory of statistical mechanics. It is desired that over time, T decreases to decrease exploration [58]. The value $\mathcal{U}(.)$ is the estimated function related to the action under consideration and the current state (Eq. 3.6). Notice that $\mathcal{U}(.)$ is the long term expected reward and not just the instantaneous one, for it carries the evolution of personality traits γ_j. This means that the decisions we take are based in the expectation to solve the problem and not just in the instantaneous reward. And finally n is the number of possible actions that a robot may perform.

3.5 Perspectives

The implementation of swarm-based robotics in terms of personalities is a very promising research topic. The implementation of such a technique is straight-forward. It may also be shown that such a procedure results in coordination among a large number of robots [41].

However, there are some problems related with the approach described above.

First of all, the number of actions is directly proportional to the number of cost functions that are evaluated (according to Eq. 3.6). Moreover, the actions are related to the number of different states that may be perceived by the robots, a fact that is made explicit by Eq. 3.10. Note that in practice, we cannot possibly predict all the states of the world. The solution for that is to represent the world by a finite number of states S. Furthermore, we will need what is called a *feature space* [97], which is defined as a function $g : S \times A \rightarrow U$ where U is the feature space. This representation has several advantages as reported in [97]. It may be shown that the number of values $V(.)$ is exponentially proportional to the cardinality of the space S. In other words, if the cardinality of the feature space is represented by $|U|$ and that of the state space by $|S|$, in order to build the rewards of Eq. 3.6 we will need

$$|U| = |S|^{|A|} \tag{3.11}$$

Therefore, any increase in the number of possible states (or in the number of actions) will have a great influence in how fast we could decide what is the best action at each time t. For example, let say that the robot may perceive the environment to be in 8 different states, i.e., $|S| = 8$. Now let the robot have 2 actions at its disposal, i.e., $|A| = 2$. Therefore the cardinality of the feature space is $|U| = 8^2 = 64$. If we increase the number of actions by one, the cardinality of the feature space becomes $|U| = 8^3 = 512$.

In the next chapter, we show some applications of the ideas presented so far. Game theory and swarm intelligence are linked in the solution of some problems that may be related to multiple robotic environments and Algorithms and heuristics based on the idea of personality traits will be developed.

Chapter 4

Evolution of Personality Traits

4.1 Introduction

As seen in the previous chapter, swarm intelligence may be defined as a distributed problem-solving technique for multi agent dynamic systems based on self-organization theory and inspired by collective social behaviour [9].

Swarm robotics has been defined by Sahin as *the study of how large number of relatively simple physically embodied agents can be designed such that a desired collective behaviour emerges from the local interactions among agents and between the agents and the environment* [88]. In our approach we use a looser definition in what concerns the term *desired* collective behaviour. We are more interested in what Beni [7] called *unpredictability* of the system. In other words, we are interested in the patterns and behaviours that will arise from the group interactions.

An important work that may be cited on swarm robotics was performed by Dorigo et. al. [31]. In this paper, they provide an overview of their project (SWARM-BOTS). They report experimental work that show how robots may cooperate and coordinate

tasks. The main difference with our approach is in the definition of the relationship among robots and the environment. In this thesis we make extensive use of Game Theory.

In practice, swarm intelligence is not intended to generate a rational individual entity, as classical Artificial Intelligence proposes [9]. However, through investigation of simple computational models that may be implemented in simple machines, swarm intelligence tries to explore how complex tasks might be performed by a social entity due to behaviours that are not directly predicted by the particular characteristics of each individual [29]. For example, if we have a society with a majority of peaceful agents, when some aggressive individuals enter the community, we cannot say that the majority will make the new individuals become peaceful. It is our purpose to try to predict such behaviours through modeling and simulation in order to justify techniques that may be used in a swarm-based robotic environment.

For the purpose of this chapter, we defined the following design guidelines:

- The algorithms the robots must execute are simple (computationally) and may be run by simple DSP (or even FPGA) chips.
- The robots sensory capacity is limited and supposed to be the same.
- The message exchanging is very limited or nonexistent. When (and if) the robots communicate with each other, no guarantee is given that the message will be received or, if it is received, that the content of the message was understood.
- The robots are able to sense the presence of other robots and, for simulation sake, the position of targets and/or enemies.

In this chapter we tie together some of the ideas presented so far. As discussed

in Chapter 3, in order to observe the emergence of group behaviours, we rely on the concept of "personality traits" [75, 57]. Using the adaptation of personality traits discussed, the underlying behaviour of each robot changes and by changing each robot's behaviour the group's behaviour also changes [9].

The environment is modelled as a game and techniques of learning in games are used. The field of learning has been extensively investigated. For example, [34] brings a comprehensive discussion of the theory behind learning in games. In [91, 108, 47], we find discussions related to the topic as well as very interesting simulations of possible applications. In the modeling developed in this chapter, we use a version of fictitious play. This learning technique was introduced in [16] and its convergence was proven in [85]. In addition to fictitious play (used in the case of a zero-sum game and derived from our general algorithm), we also make use of reinforcement learning, a very powerful tool with several different applications to the field of artificial intelligence [100].

In order to demonstrate the ideas presented so far, we now introduce three simulations that incrementally become more complex and, together, show how powerful the approach suggested is.

This chapter is divided as follows. We start, in section 4.2, with a description of the general framework for simulation. In section 4.3 we introduce a simulation of a zero-sum (matrix) game wherein no saddle point exists and therefore mixed strategies must be used. A theoretical proof of convergence is given and we compare the theoretical optimal solution with the results obtained by the learning procedure depicted in this thesis. In section 4.4 we define some concepts that will be used in the solution of the problems presented in the sections to come. Section 4.5 presents

a more complex application than the one discussed in section 4.3. In this section, we model a robotic conflict using a non-zero-sum game. Moreover, this time, we will have more players (3 robots) and the reward will not only be the payoff table, but a combination of payoffs and goal achievement. There is no proof of convergence for this case and its existence is an open question. The results shown are based on heuristics. In section 4.6 we introduce a situation in which the robots do not perceive the environment completely. Therefore, we will make use of potential fields for the representation of the environment and a still more complex learning procedure. This is also an open problem. However the results are very promising. Finally, *section 4.7* concludes the chapter with a discussion of all simulations, bringing them together in a detailed fashion.

4.2 Simulation framework

All the simulations presented in this chapter will follow a single framework. However, the meaning of some of the terms will change as we go forth in considering more complicated situations. The general framework is described in Algorithm 4.2.1 below.

For each of the simulations presented in this chapter, the steps in Algorithm 4.2.1 could be expanded as needed. Specially, in the simulations shown in section 4.6, each of the steps above is implemented as a series of several items in the instantiated algorithm.

Algorithm 4.2.1 General algorithm

1: Define, if necessary, the variables necessary to represent the environment and initialize them. This may take several steps of the algorithm.
2: Initialize learning rate η. This value is dependent on the problem under consideration.
3: Define how many personality traits are necessary for the problem under consideration and initialize them.
4: Define the payoffs for the game. These payoffs may be a matrix (or set of matrices) as in sections 4.3 and 4.5, or reward functions $\mathcal{E}_j$ representing the contribution of a trait of personality to the success of a mission combined with a matrix describing the state of the environment as in the simulation in section 4.6.
5: Play the game. The rules for the game will be introduced in each simulation. In general, we use an equation in the form of (3.6).
6: Update the personality traits according to (4.20) and (4.21) in the case of the robots leaving the room and the robots tracking a target. In the case of the zero-sum game, the updating takes a slightly different form that will be described shortly.
7: Normalize (if necessary) personality traits γ_i for each robot according to (3.7).

4.3 A zero-sum game example

The game that we will analyze in this section is reported in [98] and is represented in table 4.1. This is a zero-sum game, meaning that the payoffs for each player always add to zero. For example, if player A plays the strategy A1 and player B plays the strategy B1, player A gets rewarded 4 units while player B gets a penalty of 4; whereas, if player B decides to play strategy B2, then player B gets a reward of 4 units while player A gets penalized 4 units.

As may be seen in table 4.1, there is no saddle point for the game, therefore there must be a mixed strategy set for both players. The optimal strategies, calculated using a linear programming solver such as the Simplex, for both players, are shown in table 4.2.

		Player B Strategies					
		B1	B2	B3	B4	B5	B6
Player A Strategies	A1	4	-4	3	2	-3	3
	A2	-1	-1	-2	0	0	4
	A3	-1	2	1	-1	2	-3

Table 4.1: Zero-sum game example

Player	Strategy	Optimal Frequency
Player A	A1	24%
	A2	21%
	A3	55%
Player B	B1	0%
	B2	36%
	B3	0%
	B4	57%
	B5	0%
	B6	7%

Table 4.2: Optimal mixed strategies

4.3.1 Convergence

A general proof of convergence of strategies (or actions) to a Nash Equilibrium is virtually impossible to be obtained for equation (3.9) of section 3.4 or, equivalently, for the algorithm presented in section 4.2. Therefore, we now provide a proof of convergence for the special case of zero-sum games [1]. Our proof will follow closely the one found in [90]. However, one fundamental difference is made. Since we use personality traits, the strategies dynamics depend on them and additional considerations must be made. Therefore, the personality dynamics are introduced in order to derive the strategies dynamics.

[1] In the particular case of zero-sum games, the convergence is to a Nash equilibrium.

For a zero-sum game, the utility functions for the two players involved would be:

$$\mathcal{U}_1(\bar{p}_1, \bar{p}_2) = \bar{p}_1^T M \bar{p}_2 \tag{4.1}$$

$$\mathcal{U}_2(\bar{p}_2, \bar{p}_1) = -\bar{p}_2^T M^T \bar{p}_1 \tag{4.2}$$

where $\bar{p_1} \in \mathbb{R}^n$ and $\bar{p_2} \in \mathbb{R}^m$ are arrays of probabilities where p_{i_n} is the probability that strategy n for player i be executed. The matrix $M \in \mathbb{R}^{n \times m}$ is the payoff matrix for both players. For the simulation at hand, matrix M is

$$M = \begin{bmatrix} 4 & -4 & 3 & 2 & -3 & 3 \\ -1 & -1 & -2 & 0 & 0 & 4 \\ -1 & 2 & 1 & -1 & 2 & -3 \end{bmatrix} \tag{4.3}$$

In order to proceed with our proof, we need to define some notation. This is done in the following definition.

Definition 4.3.1. *Consider $\bar{x} = [x_1, \cdots, x_n]$ where n represents the number of strategies*

- $\Delta(n)$ *denotes the simplex [108] in $\mathbb{R}^n$ (figure 4.1), i.e.,*

$$\Delta(n) = \{\bar{x} \in \mathbb{R}^n : \bar{x}_i \geq 0 \; \forall i = 1, \cdots, n; and \sum_{j=1}^{n} \bar{x}_i = 1\}$$

- $Int(\Delta(n))$ *is the set of interior points of a simplex [108], i.e.,*

$$int(\Delta(n)) = \{\bar{x} \in \Delta(n) : \bar{x}_i > 0 \; \forall i = 1, \cdots, n\}$$

- *$bd(\Delta(n))$ is the boundary of the simplex [108] (figure 4.1), i.e.,*

$$bd(\Delta(n)) = \{\bar{x} \in \Delta(n) : \bar{x} \notin int(\Delta(n))\}$$

- *$\bar{v}_i \in bd(\Delta(n))$ is the i^{th} vertex of the simplex $\Delta(n)$, i.e.,*

$$\bar{v}_i = \{\bar{x} \in \Delta(n) : \bar{x}_i = 1 \text{ and } \bar{x}_j = 0 \; \forall j \neq i\}$$

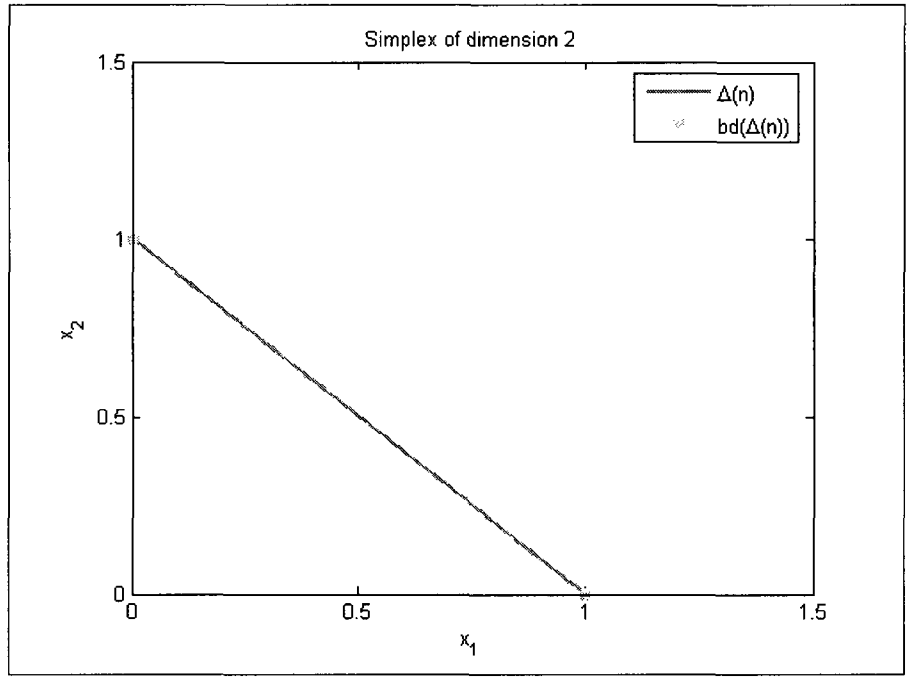

Figure 4.1: Simplex of a player with two strategies

Now define the best response mappings as

$$\bar{\beta}_1(\bar{p}_2) = \arg\max_{\bar{p}_1 \in \Delta(n)} \mathcal{U}_1(\bar{p}_1, \bar{p}_2) \tag{4.4}$$

$$\bar{\beta}_2(\bar{p}_1) = \arg\max_{\bar{p}_2 \in \Delta(m)} \mathcal{U}_2(\bar{p}_2, \bar{p}_1) \tag{4.5}$$

The utilities in (4.1) and (4.2) are implementations of the utility function (3.6). We also need to define the reward functions related to each personality trait (the functions $\mathcal{E}_j$) and the function used to update the personality traits (function $\bar{F}(\bar{\gamma}, \bar{\mathcal{E}})$ used in (3.9)) for each of the two players. Note that the arguments for the utility functions are apparently different from the parameters found in (3.6), however, the parameters are embedded in the definitions of $\bar{p}_1$ and $\bar{p}_2$, for they are dependent on $\bar{\gamma}_1$ and $\bar{\gamma}_2$ respectively (4.7). Function $h(\cdot)$ is the identity.

Let us now define the concept of empirical frequency (expectation of the opponent executing each of its actions). The empirical frequency $\bar{q}_i$ is calculated as the running average [90] of the observed actions of the opponent (recall that we have access to the action the opponent has played at each time step)

$$\begin{aligned} \bar{q}_1(k) &= \bar{q}_1(k-1) + \frac{1}{k}(\bar{v}_{a_1(k-1)} - \bar{q}_1(k-1)) \\ \bar{q}_2(k) &= \bar{q}_2(k-1) + \frac{1}{k}(\bar{v}_{a_2(k-1)} - \bar{q}_2(k-1)) \end{aligned} \tag{4.6}$$

where $a_i(k-1)$ is the action executed by player i at time step $k-1$ and $\bar{v}_i$ is a vertex of the simplex as defined in definition 4.3.1. We may assume that as $k \to \infty$, $\bar{q}_i(k) \to \bar{p}_i$. Therefore, in the proof, we are going to use both terms interchangeably.

Let us define a mapping from the personality space to the strategy space $d : \mathbb{R}^r \to$

$\Delta(n)$ as

$$\begin{aligned} \bar{q}_1 &= A_1\bar{\gamma}_1 \\ \bar{q}_2 &= A_2\bar{\gamma}_2 \end{aligned} \tag{4.7}$$

Notice that A transforms from personalities to likelihood of actions. We can now define the reward functions for each personality trait defined as (for each player)

$$\bar{\mathcal{E}}_1 = {A_1}^T(A_1{A_1}^T)^{-1}\bar{\beta}_1(\bar{q}_2) \tag{4.8}$$

$$\bar{\mathcal{E}}_2 = {A_2}^T(A_2{A_2}^T)^{-1}\bar{\beta}_2(\bar{q}_1) \tag{4.9}$$

where, $A_1 \in \mathbb{R}^{n\times r_1}$ and $A_2 \in \mathbb{R}^{m\times r_2}$. These reward functions make use of the pseudo-inverse and are used in the convergence proof. In our example (with matrix M defined in (4.3)), $n = 3$ strategies and $m = 6$ strategies. Therefore, we select $r_1 = 5$ personality traits and $r_2 = 10$ personality traits. Notice that according to (4.8) and (4.9) $r_1 \geq n$ and $r_2 \geq m$. Moreover, $rank(A_1) = n$ and $rank(A_2) = m$. Other than those, no assumption is taken. Furthermore, the functions used to update the personality traits (the $\bar{F}(\bar{\gamma}, \bar{\mathcal{E}})$ in (3.9)) are

$$\begin{aligned} \bar{F}_1(\bar{\gamma}_1, \bar{\mathcal{E}}_1) &= \bar{\mathcal{E}}_1 - \bar{\gamma}_1 \\ \bar{F}_2(\bar{\gamma}_2, \bar{\mathcal{E}}_2) &= \bar{\mathcal{E}}_2 - \bar{\gamma}_2 \end{aligned} \tag{4.10}$$

The resulting personality dynamics are

$$\begin{aligned} \dot{\bar{\gamma}}_1 &= A_1{}^T(A_1A_1{}^T)^{-1}\bar{\beta}_1(\bar{q}_2) - \bar{\gamma}_1 \\ \dot{\bar{\gamma}}_2 &= A_2{}^T(A_2A_2{}^T)^{-1}\bar{\beta}_2(\bar{q}_1) - \bar{\gamma}_2 \end{aligned} \tag{4.11}$$

In this simulation we relaxed the condition of normalization presented in (3.7). However, this is just for simplicity of calculations. Should we have wanted to do so, this could be easily implemented. Matrices A_1 and A_2 used were

$$A_1 = \begin{bmatrix} 0.3267 & 0.5071 & 0.7707 & 0.0478 & 0.3606 \\ 0.5406 & 0.7828 & 0.9703 & 0.1291 & 0.4767 \\ 0.1427 & 0.2456 & 0.3197 & 0.9082 & 0.2506 \end{bmatrix} \tag{4.12}$$

and

$$A_2 = \begin{bmatrix} 0.8686 & 0.6813 & 0.0693 & 0.2760 & 0.5695 & 0.5676 & 0.6390 & 0.6081 & 0.1034 & 0.1500 \\ 0.6264 & 0.6658 & 0.8529 & 0.3685 & 0.1593 & 0.9805 & 0.6690 & 0.1760 & 0.1573 & 0.3844 \\ 0.2412 & 0.1347 & 0.1803 & 0.0129 & 0.5944 & 0.7918 & 0.7721 & 0.0020 & 0.4075 & 0.3111 \\ 0.9781 & 0.0225 & 0.0324 & 0.8892 & 0.3311 & 0.1526 & 0.3798 & 0.7902 & 0.4078 & 0.1685 \\ 0.6405 & 0.2622 & 0.7339 & 0.8660 & 0.6586 & 0.8330 & 0.4416 & 0.5136 & 0.0527 & 0.8966 \\ 0.2298 & 0.1165 & 0.5365 & 0.2542 & 0.8636 & 0.1919 & 0.4831 & 0.2132 & 0.9418 & 0.3227 \end{bmatrix} \tag{4.13}$$

The values of matrices in 4.12 and 4.13 were generated randomly. The point is that if the matrices satisfy the conditions listed above, the algorithm will converge. More importantly, if A_1 and A_2 are the identity matrices of necessary dimensions, the

algorithm reduces to fictitious play and convergence is still guaranteed [85].

Using these definitions, we find that the strategy dynamics for player 1 is

$$\begin{aligned} \dot{\bar{q}}_1(t) &= A_1\dot{\bar{\gamma}}_1(t) \\ &= A_1({A_1}^T(A_1{A_1}^T)^{-1}\bar{\beta}_1(\bar{q}_2(t)) - \bar{\gamma}_1(t)) \\ &= \bar{\beta}_1(\bar{q}_2(t)) - \bar{q}_1(t) \end{aligned} \tag{4.14}$$

In the same way, the strategy dynamics for player 2 is

$$\dot{\bar{q}}_2(t) = \bar{\beta}_2(\bar{q}_1(t)) - \bar{q}_2(t) \tag{4.15}$$

We now define a function that measures the maximal possible reward for the players

$$V_1(\bar{q}_1, \bar{q}_2) = \max_{\bar{x}\in\Delta(n)} \mathcal{U}_1(\bar{x}, \bar{q}_2) - \mathcal{U}_1(\bar{q}_1, \bar{q}_2) \tag{4.16}$$

where $\max_{x\in\Delta(n)} \mathcal{U}_1(\bar{x}, \bar{q}_2)$ is the best "strategy" [2] that may be used by player 1. Using (4.4), we know that

$$\max_{x\in\Delta(n)} \mathcal{U}(\bar{x}, \bar{q}_2) = (\bar{\beta}_1(\bar{q}_2))^T M \bar{q}_2 \tag{4.17}$$

Therefore, from (4.16), using the definitions of utility functions in (4.1) and (4.2) together with the definition of best response mappings in (4.4) and (4.5), and collecting terms one gets

$$V_1(\bar{q}_1, \bar{q}_2) = (\bar{\beta}_1(\bar{q}_2) - \bar{q}_1)^T M \bar{q}_2 \tag{4.18}$$

[2] For a discussion on pure and mixed strategies, refer to [108]

In the same way

$$V_2(\bar{q}_2, \bar{q}_1) = -(\bar{\beta}_2(\bar{q}_1) - \bar{q}_2)^T M^T \bar{q}_1 \tag{4.19}$$

Finally, we may say that $V_1(\bar{q}_1, \bar{q}_2) \geq 0$ and $V_2(\bar{q}_2, \bar{q}_1) \geq 0$ with equality if and only if $\bar{q}_1 = \bar{\beta}_1(\bar{q}_2)$ and $\bar{q}_2 = \bar{\beta}_2(\bar{q}_1)$.

Now we prove that the learning procedure will converge to the optimal solution. We start by the following Lemma.

Lemma 4.3.1. *Define* $\tilde{V}_1(t) = V_1(\bar{q}_1(t), \bar{q}_2(t))$ *and* $\tilde{V}_2(t) = V_2(\bar{q}_2(t), \bar{q}_1(t))$. *Then* $\dot{\tilde{V}}_1(t) = -\tilde{V}_1(t) + \dot{\bar{q}}_1^T M \dot{\bar{q}}_2$ *and* $\dot{\tilde{V}}_2(t) = -\tilde{V}_2(t) - \dot{\bar{q}}_1^T M^T \dot{\bar{q}}_2$.

Proof. By definition (4.16)

$$\begin{aligned}
\dot{\tilde{V}}_1(t) &= \frac{d}{dt}[\max_{\bar{x}\in\Delta(n)} \mathcal{U}_1(\bar{x}, \bar{q}_2(t)) - \mathcal{U}_1(\bar{q}_1(t), \bar{q}_2(t))] \\
&= \frac{d}{dt}[\max_{\bar{x}\in\Delta(n)} \mathcal{U}_1(\bar{x}, \bar{q}_2(t))] - \frac{d}{dt}[\bar{q}_1^T(t) M \bar{q}_2(t)] \\
&= \nabla_{q_2}[\max_{\bar{x}\in\Delta(n)} \mathcal{U}_1(\bar{x}, \bar{q}_2(t))]\dot{\bar{q}}_2(t) - \dot{\bar{q}}_1^T(t) M \bar{q}_2(t) - \bar{q}_1^T(t) M \dot{\bar{q}}_2(t)
\end{aligned}$$

We now use the fact that [8]([90], Lemma 3.2)

$$\nabla_{q_2} \max_{\bar{x}\in\Delta(n)} \mathcal{U}_1(\bar{x}, \bar{q}_2(t)) = \nabla_{q_2} \max_{\bar{x}\in\Delta(n)} [\bar{x}^T M \bar{q}_2(t)] = \bar{\beta}_1^T(\bar{q}_2(t)) M$$

that yields (using (4.14) and (4.18))

$$\begin{aligned}
\dot{\tilde{V}}_1(t) &= \bar{\beta}_1^T(\bar{q}_2(t)) M \dot{\bar{q}}_2(t) - (\bar{\beta}_1(\bar{q}_2(t)) - \bar{q}_1(t))^T M \bar{q}_2(t) - \bar{q}_1^T(t) M \dot{\bar{q}}_2(t) \\
&= -(\bar{\beta}_1(\bar{q}_2(t)) - \bar{q}_1(t))^T M \bar{q}_2(t) + (\bar{\beta}_1(\bar{q}_2(t)) - \bar{q}_1(t))^T M \dot{\bar{q}}_2(t) \\
&= -\tilde{V}_1(t) + \dot{\bar{q}}_1^T(t) M \dot{\bar{q}}_2(t)
\end{aligned}$$

Therefore $\dot{\tilde{V}}_1(t) = -\tilde{V}_1(t) + \dot{\tilde{q}}_1^T(t)M\dot{\tilde{q}}_2(t)$. A similar derivation may be used for $\tilde{V}_2(t)$, yielding $\tilde{V}_2(t) = -\tilde{V}_2(t) - \dot{\tilde{q}}_2^T(t)M^T\dot{\tilde{q}}_1(t)$. □

And, finally, we enunciate the convergence theorem

Theorem 4.3.1. *The solutions of the system of differential equations (4.14) and (4.15) satisfy* $\lim_{t\to\infty}(\bar{q}_1(t) - \bar{\beta}_1(\bar{q}_2(t))) = 0$ *and* $\lim_{t\to\infty}(\bar{q}_2(t) - \bar{\beta}_2(\bar{q}_1(t))) = 0$.

Proof. We know from Lemma 4.3.1 that $\dot{\tilde{V}}_1(t) = -\tilde{V}_1(t) + \dot{\tilde{q}}_1^T(t)M\dot{\tilde{q}}_2(t)$ and $\dot{\tilde{V}}_1(t) = -\tilde{V}_2(t) - \dot{\tilde{q}}_2^T(t)M^T\dot{\tilde{q}}_1(t)$. Define the function $V_{12}(t) = \tilde{V}_1(t) + \tilde{V}_2(t)$. Taking its derivative, we have

$$\begin{aligned}\dot{V}_{12}(t) &= \dot{\tilde{V}}_1(t) + \dot{\tilde{V}}_2(t) \\ &= -\tilde{V}_1(t) - \tilde{V}_2(t)\end{aligned}$$

Since $\tilde{V}_1(t) \geq 0$ and $\tilde{V}_2(t) \geq 0$ with equality only at the equilibrium point of (4.14) and (4.15), $\tilde{V}_{12}(t)$ is a Lyapunov function and the theorem follows the arguments. □

4.3.2 Simulation results

We now present the results for the problem presented in the matrix in (4.3).

Suppose that both players are initialized with personality traits set to random values (in the interval $[0, 1]$). This means that both players start with a random probability of playing each strategy that is different from the optimal showed on table 4.2. The question we want to answer is: if we use personalities as described above, will the players learn to play the best mixed strategy? If not, will there be any improvement over time?

Algorithm 4.3.1 is followed by both players. As described in (4.12) and (4.13), we have created 5 personality traits for player 1 and 10 personality traits for player 2. Observe that this is not necessary and we could have used a much larger number of personalities (or, on the other hand, a smaller number greater or equal to the number of strategies) to characterize our player.

The utility functions are defined in (4.1) and (4.2). If we discretize the equation of dynamics for the personality traits ((4.14) and (4.15)) we end up with equations of the form found in (3.8) where, again, the value η may be called the learning rate and is set to a positive number and functions $\bar{F}(\bar{\gamma}, \bar{\mathcal{E}})$ are defined as in (4.10).

Algorithm 4.3.1 Zero-sum game

1: $\eta \leftarrow 0.1$
2: $t \leftarrow 0.01$
3: Set the number of personality traits for player A equal to 5 and for player B equal to 10.
4: Initialize the personality traits $\bar{\gamma}_A$ and $\bar{\gamma}_B$ randomly.
5: Define the payoffs for the game to be according to the matrix in (4.3). Also, define the mappings from the personality traits spaces to the strategies (or actions) spaces according to the matrices in (4.12) and (4.13).
6: Player A initialize the empirical frequency of player B, p_2, as 0. Player B initialize the empirical frequency of player A, p_1, as 0.
7: **for** $i = 1$ to 5000 **do**
8: Player A calculates the strategy to play according to its probability distribution $\bar{q}_1 = A_1\bar{\gamma}$. The action chosen is called a.
9: Player B calculates the strategy to play according to its probability distribution $\bar{q}_2 = A_2\bar{\gamma}$. The action chosen is called b.
10: Both players play their actions.
11: The payoff for player A is M_{ab} and the payoff for player B is $-M_{ab}$.
12: Update personality traits as described in (3.8).
13: Player A updates empirical frequency of player B according to (4.6).
14: Player B updates empirical frequency of player A according to (4.6).
15: Record the payoff for player B $(-M_{ab})$.
16: **end for**

We then ran the simulation described in algorithm 4.3.1. The loop in line 7 was

run for 5000 iterations and we then collected the final probabilities of using each one of the 3 strategies at player A's disposal and each one of the 6 strategies at player B's disposal as well as the value that player B obtained in each simulation (the value obtained by player A is just the negative of the one obtained by player B). Such results are summarized in table 4.3. One may notice that, indeed, the procedure made the personalities converge to their optimal values. Furthermore, we observe that the disadvantageous strategies B1 and B3 were almost never used. Moreover, strategy B5, which is dominated by strategy B2, was never used. The learning procedure even caught the subtlety that B2 dominates B5. Furthermore, the average payoff for player B was 0.0710, very close to the theoretical value of the game, 0.07 [98]. The

Player	Strategy	Optimal Frequency	Experimental Frequency
Player A	A1	24%	22.04%
	A2	21%	23.93%
	A3	55%	54.03%
Player B	B1	0%	1.05%
	B2	36%	36.10%
	B3	0%	0.02%
	B4	57%	56.39%
	B5	0%	0%
	B6	7%	6.44%

Table 4.3: Experimental results obtained for both players

importance of this example resides in the fact that the robots are able to learn from experience when the task is represented in terms of a game. Notice that for a zero-sum game, the approach reduces to fictitious play. Therefore, convergence to the Nash Equilibrium will always occur. This is not the case for the other simulations presented in this chapter.

4.4 Implementation for next sections

In the next two simulations we are going to use the following implementation.

Going back to (3.8), we have to define how the function $\bar{F}(\bar{\gamma}, \bar{\mathcal{E}})_t$ will behave. Define $\Delta\mathcal{E}_i(t) = \mathcal{E}_i(s_t, \alpha_j, t) - \mathcal{E}_i(s_{t-1}, \alpha_k, t-1)$ where α_j is the action taken at time t and α_k was the action taken at time $t-1$, we define the step as

$$\Delta\gamma_i(t) = \frac{\Delta\mathcal{E}_i(t)}{\sum_{j=1}^{n} \Delta\mathcal{E}_j(t)} \tag{4.20}$$

where n is the number of personality traits of a robot. Now we define the adaptation law as

$$\gamma_i(t) = \gamma_i(t-1) + \eta\Delta\gamma_i(t),\ 0 < \eta < 1 \tag{4.21}$$

Equations (4.20) and (4.21) imply, because of the presence of all personality traits in the denominator, that each trait of personality influences the others. Therefore, changing a singular trait will affect the way all the others work. Furthermore, the convergence of (4.21) is highly dependent on the value of the learning rate η: if this is small, the convergence is too slow and the robots take too long to adapt to new situations; if it is too high, the system oscillates a lot around some value before it converges and when it does, it has a large probability to go to a local maximum (minimum). In our simulations, we use a small value for this term so that we achieve a smooth convergence. Equation (4.21) is in the same form used in the reinforcement learning literature. More details may be found in [58]. Finally, notice that nothing can be said about the convergence of the algorithm at this point, for the coefficient of $\bar{\gamma}_t$ is not known.

When the state is recognized, the action is chosen according to the formula, known as the randomized strategy [58], which is useful for leading the robot to "explore" new actions and not just "exploit" learnt sequence of actions. For completeness sake, we will repeat the next equation, already defined in the previous chapter, which demonstrates the randomized strategy.

$$P(\alpha_i|s) = \frac{k^{\mathcal{U}(s,\bar{\gamma},\alpha_i)/T}}{\sum_{j=1}^{m} k^{\mathcal{U}(s,\bar{\gamma},\alpha_j)/T}} \tag{4.22}$$

In (4.22), P is the probability that action α_i will be executed when the state is s in the presence of the personality traits $\bar{\gamma}$, where $\bar{\gamma} = [\gamma_1, \cdots, \gamma_n]$ is a vector of personality traits (as introduced in section 3.4). The term k is a coefficient that defines how often the robot explores new solutions or exploits the ones it already knows as better ones. When k increases, the probability that the robot explores new choices decreases and vice-versa. T is a temperature parameter inspired in *Boltzmann* theory of statistical mechanics. It is desired that over time, T decreases to diminish exploration [58]. The utility function $\mathcal{U}(.)$ is related to the action under consideration and the current state. $\mathcal{U}(.)$ is the long term expected reward and not just the instantaneous one. This means that the decisions the robot takes are based in the expectation to solve the problem (find a target or perform a predefined task) and not just in the instantaneous reward. This difference is implemented in the personalities, since robots learn over time, they will develop the capability to "predict" the payoffs of their actions. Moreover, n is the number of personality traits. And finally m is the number of possible actions that a robot may perform (in our present case it is the number of different strategies available for the robot). In this chapter, in all simulations we will use $k = e$

(i.e., $exp(1) = 2.7183$) and $T = 1$. $T = 1$ means that we do not reduce exploration as time goes by. $k = e$ means that the robots have a preference for exploitation of already learnt strategies but are also open to exploration [58].

4.5 Robots leaving a room

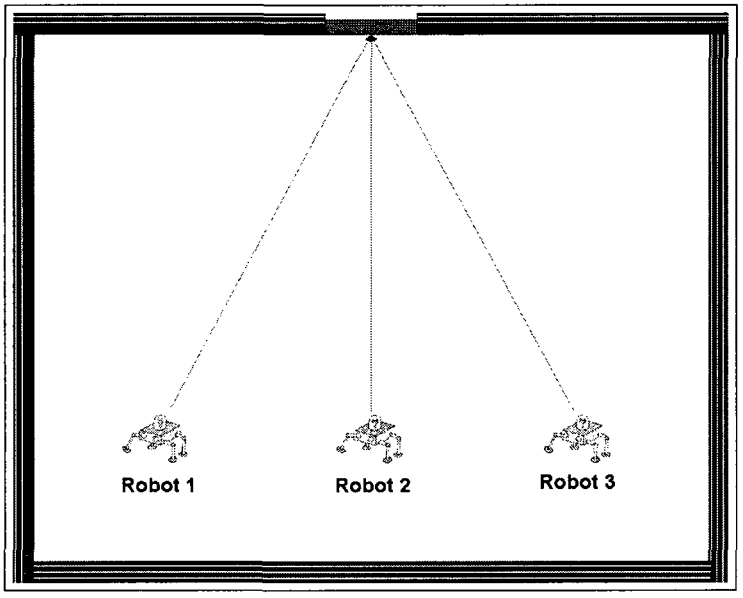

Figure 4.2: Artistic depiction of the problem of robots leaving a room

Our second simulation is similar to the one introduced and reported in [111]. The setup is the following. Three robots of 1 unit in diameter are located in a room with dimensions 8×8 (corners at $(0,0)$, $(0,8)$, $(8,8)$ and $(8,0)$). There is a door centered at $(3,8)$ and with dimensions so that just one robot may pass. Inside the room, there are three robots located at positions $(3+3\sqrt{2}, 8-3\sqrt{2})$, $(3,6)$ and $(3-3\sqrt{2}, 8-3\sqrt{2})$, i.e., at a distance 6 units from the door (figure 4.2). It is assumed that one robot knows the positions of the other two without any noise, and it is also assumed that

the robots only move in a straight line from their initial position toward the center of the door with fixed speed, 1 unit/s. The problem may be described as a game shown in Table 4.4, which has the payoffs for player A. The values X and Y in the table must follow the rule

$$X \in \mathbb{Z}^+ \text{ and } Y \in \mathbb{Z}^- \tag{4.23}$$

where the values for X and Y will depend on how the designer chooses to represent

		Player B Strategies	
		Walk	Wait
Player A Strategies	Walk	-1	X
	Wait	Y	0

Table 4.4: Modeling of a game between two robots trying to leave a room

the environment. If the designer wants to enhance the action "Walk" for player A, then set $|X| > |Y|$. On the other hand, if the designer wants to enhance the action "Wait", then set $|Y| > |X|$. And finally, if both are considered to be at the same level, set $|X| = |Y|$.

Before we state the algorithm, we need to make some definitions.

Definition 4.5.1. *Definitions for algorithm 4.5.1.*

i. The payoffs for the robots 1 and 3 in Figure 4.2 are according to the matrix

$$M_1 = \begin{bmatrix} -1 & 1 \\ -1 & 0 \end{bmatrix} \tag{4.24}$$

i.e., where $|X| = 1$ *and* $|Y| = 1$ *(see Table 4.4 and equation (4.23)). For player*

2, the payoff table is

$$M_2 = \begin{bmatrix} -1 & 3 \\ -1 & 0 \end{bmatrix} \tag{4.25}$$

where $X = 3$ and $Y = -1$ (see Table 4.4 and equation (4.23)). The values are different such that robot 2's expected reward is greater than its expected penalty.

ii. The probability for a robot to move is given by

$$P(Walk) = \frac{e^{\gamma_1}}{e^{\gamma_1} + e^{\gamma_2}} \tag{4.26}$$

where γ_1 is related to action 1, i.e., "Walk" and γ_2 is related to action 2, "Wait".

The results were obtained after 100 repetitions of the game with a learning rate $\eta = 0.01$. First of all, one of the robots on the sides (robots 1 and 3) converged to a purely "cooperative" robot, i.e., its personality trait for waiting for the others became 1; whereas the other robot on the side converged to a purely "competitive" robot, i.e., its personality trait for always walking became 1. Second of all, the robot in the middle chooses its actions on a $50 - 50$ basis. As result of all this, the average of the 100 games is 10.04 s to leave the room and the standard deviation is 1.82s. Also, 24 out of the 100 repetitions obtained the best solution of 8 s [111].

One may notice that the robots did not work only for their own advantage. Table 4.4 shows that the strategy "Wait" is dominated by the strategy "Walk". However, behaving the way they did made the overall result much better for the group. This is one of the interesting results that will be exploited in the next simulation. Here we see the spontaneous emergence of altruistic behaviour that enhances the performance of

Algorithm 4.5.1 Robots Leaving the Room

1: Each robot will have two personality traits, initialized as $\gamma_i = \frac{1}{2}$, $i = 1, 2$, which define which strategy (Walk or Wait in Table 4.4) the robot will play.
2: Define the payoffs for the robots 1 and 3 according to (4.24) and the payoffs for robot 2 according to (4.25).
3: $Robots \Leftarrow [1, 2, 3]$.
4: **while** there are robots in the room **do**
5: For each robot calculate the probability to move according to the personality traits. The probability to move is given by (4.26), i.e., $A_l \in \{Walk, Wait\}$ (where l is the robot's id).
6: **for** $l \in Robots$ **do**
7: **if** no other robot is the the room **then**
8: There is no conflict. Set action to "Walk", i.e., $A_l \Leftarrow Walk$.
9: **else**
10: **if** $l = 1$ or $l = 3$, $M \Leftarrow M_1$ (4.24), otherwise $M \Leftarrow M_2$ (4.25)
11: **for** $j \in Robots$, $j \neq l$ **do**
12: $F(\gamma_{A_l}, \mathcal{E}_{A_l}) \Leftarrow M(A_l, A_j)$ {Payoff for robot l playing against robot j.}
13: Update the personality trait related to the action chosen according to the equation $\gamma_{A_l}(t) \Leftarrow \gamma_{A_l}(t-1) + \eta F(\gamma_{A_l}, \mathcal{E}_{A_l})$.
14: **end for**
15: Normalize all personality traits γ_i so that $\sum_{k=1}^{2} \gamma_k = 1; \gamma_k \geq 0$.
16: **end if**
17: Add action A_l to list of actions L_l taken so far.
18: **if** action is to walk and there is no collision **then** robot l moves **else** robot l keeps its current position for one time step.
19: **if** robot l reached door **then** remove l from list $Robots$
20: **end for**
21: **end while**

the group. The emergence of altruism is due to the calculations done in steps 13 and 15 of the algorithm. Notice that the matrices in (4.24) and (4.25) do not have a positive payoff for the strategy "Wait". However, since the traits that determine the execution of the actions are normalized (step 15), the negative payoffs that the strategy "Wait" gets combined with the negative payoffs of the collisions when strategy "Walk" is chosen will drive one of the robots to be altruistic.

4.6 Tracking a target

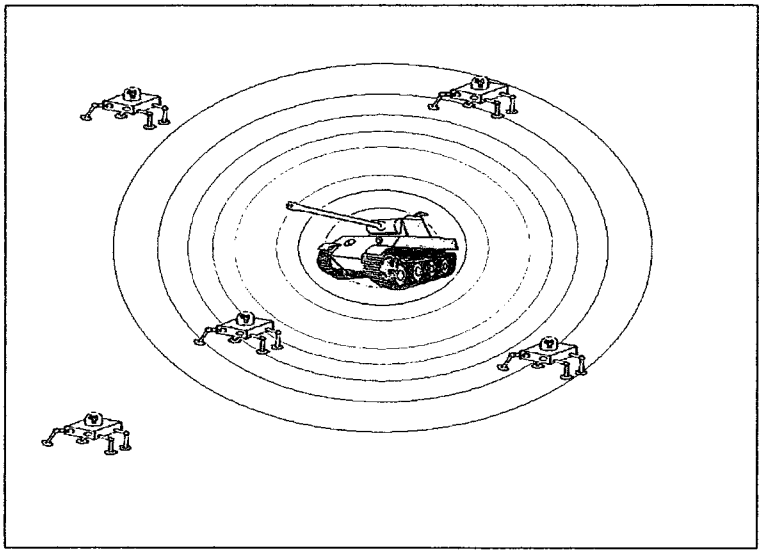

Figure 4.3: Artistic depiction of the simulation environment.

We now make use of all the ideas presented so far and define a more complex and challenging simulation mission to be accomplished by several robots working together. We set up the simulation environment as shown in Figure 4.3. In this figure, we depict a target (a tank) and several robots that are moving around it. Their objectives

are to find the target and go back to the base. In our simulation environment, the position of the target changes from simulation to simulation and the robots perceive the environment as potential fields (Gaussian potential fields). Each single robot is able to identify the target potential field, the other robots fields and the bases field. No noise is added to the readings and some delay is possible in the measurements. We also assume that the low-level dynamics of the robots and the control loops necessary to stabilize them are already implemented.

Each robot has three traits of personality: "courage" (γ_1), "fear" (γ_2) and "co-operation" (γ_3), which influence which action the robot will take. For example, a courageous robot may pursue the gradient of the target, while a cooperative and fearful one may tend to huddle together with other robots in order to look for the target as a group. Again, these behaviours are derived from our assumptions on the definition of the "emotions" of the robots.

For this simulation, the environment is supposed to be in only two states: θ_1, meaning high risk for the robot (of being shot) and θ_2, which means that the robot is in a low risk of being shot. The decision about which state the robot is in is psychological, i.e., it depends on the values of traits of personality of each robot. In this way, if a robot is "courageous" high risk has a different meaning compared to a "fearful" robot.

Let $\sigma(.)$ be a function determining the threshold in separating states θ_1 and θ_2. Let also γ_1 be the trait "courage", γ_2 be "fear" and γ_3 be "cooperation". Define F_{Max} as the maximum potential field found so far. We then define the probability for the

Utility Payoffs		States	
		θ_1	θ_2
Actions	α_1	-1	5
	α_2	4	-2

Table 4.5: Utility payoffs for states

robot to identify the environment as state θ_1 (high risk) as

$$P(\theta_1|s) = \frac{|F_T|}{|F_{Max}|} - \sigma(\gamma_1, \gamma_2, \gamma_3) \tag{4.27}$$

Since the traits of personality are normalized (as explained in the last sections), we chose the threshold function to be

$$\sigma(\gamma_1, \gamma_2, \gamma_3) = \gamma_1 - \gamma_2 - \gamma_3 \tag{4.28}$$

Therefore, if the trait of personality γ_1, "courage", is dominant, the probability the robot will identify the environment as being "high risk" will decrease. On the other hand, since $P(\theta_2|s) = 1 - P(\theta_1|s)$, the probability increases when "fear" (γ_2) and "cooperation" (γ_3) are dominant. Notice that $P(.)$ could be out of the interval $[0, 1]$, if that happens we simply truncate it.

In the same way, only two actions are possible. We shall call them α_1, which means to follow an uphill approach (getting closer to the dangerous target); and α_2, which means to follow a downhill path (according to danger). Table 4.5 describes the payoffs related to each decision when the robot identifies the environment to be in each specific state. The values in Table 4.5 are empirical and by choosing different payoffs the robots would end up with different behaviours. Also notice that the table

is not exactly a payoff table as we had in the previous examples; in this case we do not have a conflict among the robots. The numbers in the table mean that when the robot perceives the state to be in the "high risk" state θ_1, it is more "profitable" to execute action α_2 (downhill path) and when the robot finds itself in the low risk state θ_2, the robot would prefer to execute action α_1 (uphill path). Later on (in equation (4.29)), we will see that the choice is not so straightforward, but in general the rules just explained will be applied.

After the robot identifies the target, it gets back to the base with its estimation of the target location. The closer the robot gets to the target, the greater the danger of being shot (at each time step we divide the potential field where the robot is by the maximum value of the field - the position of the target - and according to this number, randomly shoot at the robot simulating an action taken by the enemy). When the robot is shot, we assume that it is still operational, but has to go back to the base in order to avoid malfunctioning. Actually, since we may have a large number of robots, this assumption is not necessary, but by making use of it, we simplify our simulation environment. When the robot is shot, we artificially increase its "fear" trait of personality in order to avoid being shot in the future. The task "get back to base" is hardwired in this approach and after the robot identifies the target it just follows the track back to safety. Notice that this behaviour is artificial and not desired, for the robot must be able to help other robots in need even if it is on its way back to the base. However, we do not implement this feature for the sake of simplicity.

The traits of personality are defined as follows:

1. Courage (γ_1) the robot goes in the direction of danger, i.e., in the direction of

the increasing potential field, therefore, this trait makes it more likely for the robot to identify the environment as in the "low risk" state (state θ_2 in Table 4.5).

2. Fear (γ_2) the robot goes in the opposite direction of danger, i.e., in the direction of the decreasing potential field, therefore, this trait makes it more likely for the robot to identify the environment as in the "high risk" state (state θ_1).

3. Cooperation (γ_3) robots tend to huddle together in order to decrease the possibility of being shot. This trait makes the robots work together.

The behaviour in 3 is explained by the assumption in the simulation that the chance of the robot being shot is inversely proportional to the number of robots huddled together. This is not a deliberate hypothesis; in fact the same assumption has been taken when studying the formation of patterns of animals in the wild (flock formation, fish schooling, etc.) [26].

To choose an action, we use the value function $\mathcal{U} : X \times A \rightarrow R$ in (4.29) that maps the state of the environment and the action under consideration to a reward. In the case of game theory we need to calculate the expected value of the value function. Therefore, define $J(s_t, \bar{\gamma}, \alpha_i) = \sum_{j=1}^{3} \gamma_j \mathcal{E}_j(s_t, \alpha_i, t)$, i.e., the summation in (3.6). Now, define $\mathcal{U}(s_t, \bar{\gamma}, \alpha_i) = \mathbb{E}\{J(s_t, \bar{\gamma}, \alpha_i)|(s, \alpha_i)\}$ as the expected value for the payoff for all possible actions α_i. We can think of this as a game against nature [98], in which the environment is supposed to play with a mixed strategy $P(\theta_i|s)$. Therefore, the expected outcome of the game in table 4.5 is

$$\begin{aligned}\mathcal{U}(s_t, \bar{\gamma}, \alpha_1) &= \mathbb{E}\{J(s_t, \bar{\gamma}, \alpha_1)|(s_t, \alpha_1)\}=[-1(P(\theta_1|s) + 5(P(\theta_2|s)]J(s, \bar{\gamma}, \alpha_1) \\ \mathcal{U}(s_t, \bar{\gamma}, \alpha_2) &= \mathbb{E}\{J(s_t, \bar{\gamma}, \alpha_2))|(s_t, \alpha_2)\}=[4(P(\theta_1|s) - 2(P(\theta_2|s)]J(s_t, \bar{\gamma}, \alpha_2) \quad (4.29)\end{aligned}$$

Equation (4.29) is the application of game theory expectation calculation to the framework introduced in section 3.4. Actually, this equation is just the implementation of (3.6) in terms of game theory, where $h(\cdot)$ is the expectation function.

Then, the action is chosen randomly based on the probability ((4.22), where $k = e$ $(exp(1) = 2.7183)$, $T = 1$)

$$P(\alpha_i) = \frac{e^{\mathcal{U}(s,\bar{\gamma},\alpha_i)}}{\sum_{j=1}^{2} e^{\mathcal{U}(s,\bar{\gamma},\alpha_j)}} \tag{4.30}$$

Before we state the algorithm, we need to introduce some definitions:

Definition 4.6.1. *Definitions for algorithm 4.6.1.*

i. The target is identified by a Gaussian field. If (X_T, Y_T) designates the position of the target, let

$$T(x,y) = Ke^{-\frac{1}{2}\frac{(x-X_T)^2}{\sigma^2}} e^{-\frac{1}{2}\frac{(y-Y_T)^2}{\sigma^2}} \tag{4.31}$$

be the Gaussian field irradiated by it. σ is the standard deviation of the field and K is a term to scale the sensitivity of the robots.

ii. The robots also irradiate Gaussian fields. If (X_R, Y_R) is the position of a robot, let

$$R(x,y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\frac{(x-X_R)^2}{\sigma^2}} e^{-\frac{1}{2}\frac{(y-Y_R)^2}{\sigma^2}} \tag{4.32}$$

be the Gaussian field around it. σ is the standard deviation of the field.

iii. The uphill unit vector for robot i located at (x_i, y_i) is

$$\vec{u}_i = \frac{\nabla T(x_i,y_i) + \sum_{j\neq i} \nabla R_j(x_i,y_i)}{|\nabla T(x_i,y_i) + \sum_{j\neq i} \nabla R_j(x_i,y_i)|} \tag{4.33}$$

iv. The downhill unit vector for robot i located at (x_i, y_i) is

$$\vec{d_i} = -\vec{u_i} \tag{4.34}$$

v. The probability for the robot identifying that it is in state θ_1, high risk, is

$$P(\theta_1|s_t) = \frac{|\nabla T(x_i, y_i) - \sum_{j \neq i} \nabla R_j(x_i, y_i)|}{|F_{Max}|} - \gamma_1 + \gamma_2 + \gamma_3 \tag{4.35}$$

where $|F_{Max}|$ is the maximum field found so far for each robot. Accordingly, the probability to be in state θ_2, low risk, is

$$P(\theta_2|s_t) = 1 - P(\theta_1|s_t) \tag{4.36}$$

vi. The probability of executing action α_1 is

$$P(\alpha_1) = \frac{e^{\mathcal{U}(s,\bar{\gamma},\alpha_1)}}{e^{\mathcal{U}(s,\bar{\gamma},\alpha_1)} + e^{\mathcal{U}(s,\bar{\gamma},\alpha_2)}} \tag{4.37}$$

Accordingly, $P(\alpha_2) = 1 - P(\alpha_1)$. Equation (4.37) is (4.22), where $k = e$, $T = 1$ and $n = 2$.

vii. The personality traits are updated using the adaptation law

$$\gamma_i(t) = \gamma_i(t-1) + \eta \Delta \gamma_i(t) \tag{4.38}$$

where

$$\Delta\gamma_i(t) = \frac{\Delta\mathcal{E}_i(t)}{\sum_{j=1}^{3}\mathcal{E}_j(t)} \tag{4.39}$$

viii. The probability of a robot being shot is

$$P(shot) = \frac{|\nabla T(x_i, y_i) - \sum_{j\neq i}\nabla R_j(x_i, y_i)|}{\max(|T(x_j, y_j)|)} \cdot 0.01 \tag{4.40}$$

where (x_j, y_j) are all the points visited by the robot previously.

As stated in the algorithm 4.6.1, the choice for the reward functions $\mathcal{E}_i(.)$ is dependent on the application. It may be argued that the reward functions would have to include some kind of external payoff based on the success of the task. However, this is not considered in the model of algorithm 4.6.1.

The interpretation of the reward functions used in the algorithm 4.6.1 is:

- $\mathcal{E}_1(.)$ is the function for the personality trait γ_1, "courage". For action α_1 "follow the uphill gradient", the reward is $\mathcal{E}_1(s_t, \alpha_1, t) = \nabla T(x_i, y_i)\cdot\vec{u_i}$. Notice that this value is positive if the angle between the gradient ∇T and the uphill unit vector $\vec{u_i}$ is in the interval $(-90°, 90°)$ and negative otherwise. For action α_2, "follow the downhill gradient", the reward is $\mathcal{E}_1(s_t, \alpha_2, t) = \nabla T(x_i, y_i)\cdot\vec{d_i}$. Notice that this value is positive if the angle between the gradient ∇T and the downhill unit vector $\vec{d_i}$ is in the interval $(-90°, 90°)$ and negative otherwise. In other words, the personality trait "courage" returns a larger value if the direction of the movement is closer to the gradient of the target. Since $\vec{u_i}$ and $\vec{d_i}$ are constituted by a summation of the gradient of the target and the gradients

Algorithm 4.6.1 Tracking of a target

1: {Initializations} Define a base and set the initial position of all robots to it. Randomly select the position of the target (X_T, Y_T) and its standard deviation σ. Set $K = \frac{100}{\sigma\sqrt{2\pi}}$ and create the gaussian field according to (4.31). Initialize each personality trait to a random value between $[0, 1]$. Normalize them so that $\sum_{j=1}^{3} \gamma_j = 1$; $\gamma_j \geq 0$.
2: For each robot located at the position (X_R, Y_R), define the field around it to be according to (4.32) with $\sigma = 4$.
3: Initialize a list *Robots* $\Leftarrow [1, 2, \cdots, n]$ with all robots.
4: **repeat**
5: **for** $i \in$ *Robots* {for all robots} **do**
6: Calculate the gradient $\nabla T(x_i, y_i)$ at the current position of the robot.
7: **for** $j \in$ *Robots*; $j \neq i$ **do** calculate the gradient of each robot's field $\nabla R_j(x_i, y_i)$.
8: Calculate probabilities of identifying the robot in states θ_1 and θ_2 according to (4.35) and (4.36).
9: Calculate the rewards for each personality trait $\mathcal{E}_1(.)$, $\mathcal{E}_2(.)$ and $\mathcal{E}_3(.)$.
10: Calculate the expected values for the execution of each action according to (4.29).
11: Calculate the uphill unit vector (4.33) and the downhill unit vector (4.34). Calculate the probability of executing action α_1 (uphill) and α_2 (downhill) as described in (4.37). Randomly select the action to be executed using these probabilities.
12: Calculate the step for the adaptation of traits of personality according to (4.39).
13: Update the personality traits using the adaptation law in (4.38).
14: Calculate the probability of a robot being shot as in (4.40).
15: **if** robot i is shot **then** remove robot i it from list *Robots* and go back to base.
16: **end for**
17: **until** all robots are back to base

of the robots (equations (4.33) and (4.34)), the direction that is closer to the danger is preferred.

- $\mathcal{E}_2(.)$ is the function for the personality trait γ_2, "fear". For action α_1 "follow the uphill gradient", the reward is $\mathcal{E}_2(s_t, \alpha_1, t) = (\sum_{j \neq i} \nabla R_j(x_i, y_i)) \cdot \vec{u}_i$. Notice that this value is positive if the angle between the summation of gradients $(\sum_{j \neq i} \nabla R_j(x_i, y_i))$ and the uphill unit vector $\vec{u}_i$ is in the interval $(-90°, 90°)$ and negative otherwise. For action α_2, "follow the downhill gradient", the reward is $\mathcal{E}_2(s_t, \alpha_2, t) = (\sum_{j \neq i} \nabla R_j(x_i, y_i)) \cdot \vec{d}_i$. Notice that this value is positive if the angle between the summation of gradients $(\sum_{j \neq i} \nabla R_j(x_i, y_i))$ and the downhill unit vector $\vec{d}_i$ is in the interval $(-90°, 90°)$ and negative otherwise. In other words, the personality trait "fear" returns a larger reward for the action that moves the robot closer to other robots.

- $\mathcal{E}_3(.)$ is the function for the personality trait γ_3, "cooperation". For both actions, the reward is calculated as $\mathcal{E}_2(s_t, \alpha_k, t) = \sum_{j \neq i} \nabla R_j(x_k, y_k)$, for $k = 1, 2$ where (x_k, y_k) is the future position of the robot. Let $\vec{p}_i$ be the robots current position and $|\vec{v}_i|$ the speed of the robot, which, in our case is 1 unit/s. For action α_1 "follow the uphill gradient", the reward function $\mathcal{E}_3(.)$ is evaluated at $(x_1, y_1) = (\vec{p}_i + \vec{u}_i \cdot |\vec{v}_i|)$. For action α_2, "follow the downhill gradient", the reward function $\mathcal{E}_3(.)$ is evaluated at $(x_2, y_2) = (\vec{p}_i + \vec{d}_i \cdot |\vec{v}_i|)$. The personality trait "cooperation" assumes that when the robot moves closer to other robots, the survival of the groups is enhanced.

Indeed, the concept of success in the definition of reward functions is very subjective. Depending on the information we have available and the complexity of the

model we establish, success would have a completely different meaning. For example, with the same setup of Algorithm 4.6.1, we may assume that the robots know where the target is, and the task could be just to get close to it. In this case, the reward functions $\mathcal{E}_i(.)$ could include external information on how dangerous the environment becomes at each step, say how close a shot came to hit the robot, or how close we got to the target. Notice that in Algorithm 4.6.1, we did not assume that this information is available.

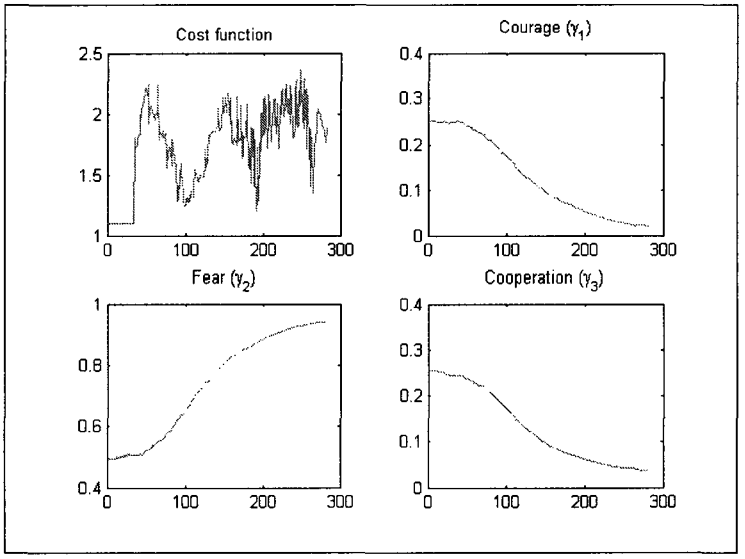

Figure 4.4: Utility function and personality traits of one robot

Since we chose a low value for the learning rate ($\eta = 0.01$), it is expected that there will be a slow convergence of the traits of personality to a steady state value. Results for one arbitrarily chosen robot are depicted in Figure 4.4. This figure indicates that the traits of personality do converge to a steady state value. In this figure, the value function is $\mathcal{U}(s_t, \bar{\gamma}, \alpha_k)$, where α_k is the action executed at time t. Notice that the value function varies around some range (this is not necessarily the case; until further

Number of robots	Courage (γ_1)		Fear (γ_2)		Cooperation (γ_3)	
	Average	Standard deviation	Average	Standard deviation	Average	Standard deviation
10	0.1648	0.1377	0.1158	0.1435	0.7194	0.2743
20	0.2451	0.1006	0.2381	0.1713	0.5167	0.2316
30	0.1299	0.0930	0.3066	0.1827	0.5636	0.1692

Table 4.6: Convergence of the personality traits

proof, this should be taken as a particularity of the simulation analyzed). We may notice that the robot becomes a "fearful" robot (γ_2 increases, while the other traits decrease). Therefore, we may hypothesize that this particular robot is in some kind of cluster of robots, which makes variations on the cost functions for the particular traits more difficult. Moreover, the particular values of the personalities are characteristic of the one simulation at hand. Had we had a different initialization, we could get to different steady state values for the traits of personality, since the environment changes considerably as well as the robots initial conditions (the initial values for the traits of personality). Table 4.6 shows that in a given run, all the robots do converge to a steady state value and they are related to each other. This is not necessarily true for different payoff tables (like Table 4.5) and reward functions ($\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3$) and must be considered (until further proof) as a particularity of the simulation setup under analysis.

In order to evaluate the quality of the simulations, we measure the quality of the target location by the robots. When the i^{th} robot goes back to base, it records the position (x_{S_i}, y_{S_i}) where it was shot (recall that we suppose that the robot just goes back to base when it is shot). Therefore, if (x_T, y_T) is the actual position of the target, the error of the best target location is $(\|(x_T, y_T) - (x_{S_i}, y_{S_i})\|), i = 1, \cdots, n$, where n is

Number of robots	Target location error		Total Time		Location error for all robots	
	Average	Standard deviation	Average	Standard deviation	Average	Standard deviation
10	12.2000	6.5201	93.3000	55.6698	17.3810	7.7339
20	9.5880	3.8975	136.1000	45.4715	15.5337	5.4713
50	8.4136	3.9315	322.5000	117.8740	15.3012	5.5135

Table 4.7: Simulation Results

the number of robots in the simulation. We also measure the total time that it takes for all the robots to get back to base, and the average location error for all robots in the simulation. The results are shown in Table 4.7, wherein we have the average and standard deviation of the target location error, total time of the missions and average location error for all robots. All the results are obtained through 10 executions of the target-tracking mission.

The results indicate that some robot behaviours are independent of the number of robots in the fleet. There is also a tendency to get a better location of the target with an increasing number of robots. This is due to our assumption that the robots are less likely to get shot when they are in larger numbers (equation (4.40)) by means of huddling together, which has been observed in the simulations. In fact, in order to visualize better the effect of the other robots in how a robot decides to act, we considered the enemy (the tank) to be more accurate and replaced (4.40) by

$$P(shot) = \frac{|\nabla T(x_i, y_i) - 10 \sum_{j \neq i} \nabla R_j(x_i, y_i)|}{\max(|T(x_j, y_j)|)} \cdot 0.1 \tag{4.41}$$

i.e., the robots are 10 times more likely to be shot than predicted in the algorithm (therefore the probability is multiplied by 0.1 instead of 0.01). Also, the presence of

other robots in the neighbourhood makes more unlikely for a robot to be shot (this is the meaning of the factor 10 in the equation above). In this way, robots will take advantage of the increase in the number of robots in the neighbourhood. Table 4.7 also indicates that as the number of robots increases, the total time for target location also increases, although not linearly. This happens for two different reasons. First of all, the robots take longer to leave the base (we assume that just one robot leaves the base at each time step). Second of all, as we have a larger number of robots, the chance of being shot decreases (again, (4.41)) and, therefore, they take much longer to get back to base.

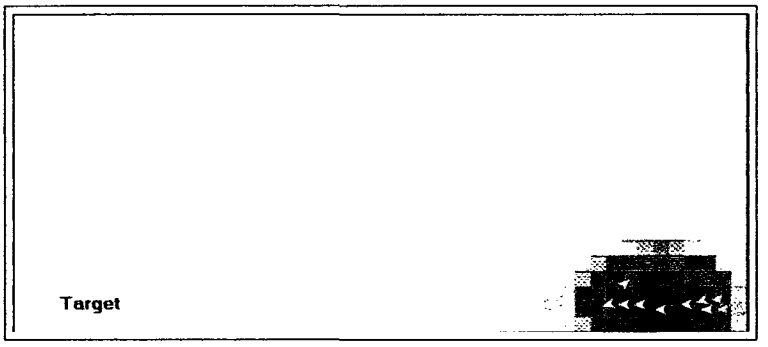

Figure 4.5: State of the robots during the simulation

When we used the probability in (4.40) (as in the simulation illustrated in Figure 4.4), the traits "fear" and "cooperation" are always more important, given rise to the most interesting behaviour observed in the simulation: the tendency for the robots to huddle together. In most simulations, they formed a big group and kept like that until the individual robots were being shot by the enemy. This may also be seen in Table 4.7, for as we increase the number of robots, the average distance to the target slightly decreases. This is also a result from the emergent huddling behaviour. Figure 4.5 shows a picture of the state of the robots in the simulation. We can see

that the robots do huddle together, but some of them (the more courageous ones) move farther from the centre of the swarm. However, they are more likely to be shot (a result seen from (4.40)).

Another aspect observed in the simulations was the behaviour of robots after some of them were shot. Observe that, since the number of active robots decrease, the reward $\mathcal{E}_2(.)$ for the personality trait γ_2, "fear", calculated on step 19 of Algorithm 4.6.1, and $\mathcal{E}_3(.)$ for the personality trait γ_3, "cooperation", calculated on step 20, decrease. Therefore, the reward $\mathcal{E}_1(.)$ for the trait of personality γ_1, "courage", gets more important for the remaining robots and they tend to "attack" the target more directly. This was a behaviour observed when just some few robots were left. Since there is no other robot to help them, the remaining robot takes more risks and move toward the target, therefore increasing the risk of being shot.

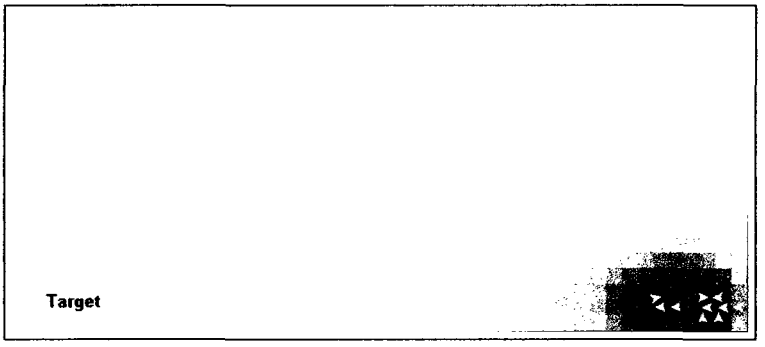

Figure 4.6: State of the simulation when two robots are turned courageous

In order to examine how resilient to spurious behaviours the swarm is, we then fixed some robots as courageous and ran the simulation again. The idea is to check out when the group will start showing different group behaviours than the ones observed so far. Figures 4.6, 4.7 and 4.8 are snapshots of the simulation over the period of time when some robots were set to "courageous". We also reduced the probability of

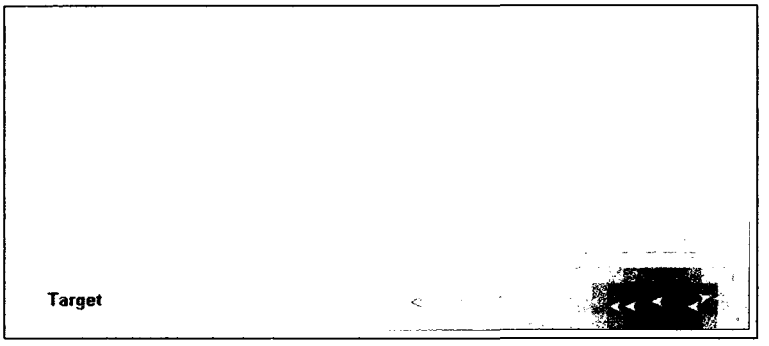

Figure 4.7: State of the simulation when five robots are turned courageous

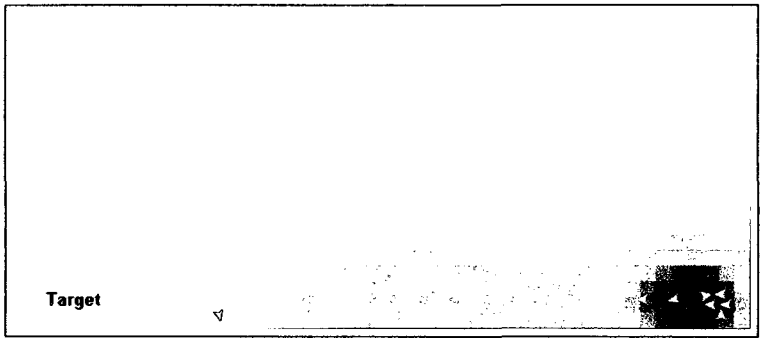

Figure 4.8: State of the simulation when ten robots are turned courageous

getting shot even more, to just 10% of the value in (4.41). Figure 4.6 shows the state of the simulation when 2 out of 20 robots are made courageous. It does not look very different from the state of the simulation in Figure 4.5. Figure 4.7 shows the state of the simulation when 5 out of 20 robots are made courageous and Figure 4.8 shows the state of the simulation when we turn 10 out of 20 robots courageous. We see that in Figure 4.7, the group of robots start to break and, in Figure 4.8, when half the robots turn courageous, the group of robots is completed broken. This suggests that the swarm of robots is resilient to outlier individuals up to some limit, but as the number of robots with some specific trait of personality increases, the swarm dynamics can

dramatically change. In the case depicted in Figures 4.6 to 4.8 we see that when more robots become courageous, they drive the entire group to a courageous state. We note that for this behaviour to become noticeable we artificially and arbitrarily set the trait of personality γ_1, "courage", for the courageous robots to 1 and the other two traits were set to zero.

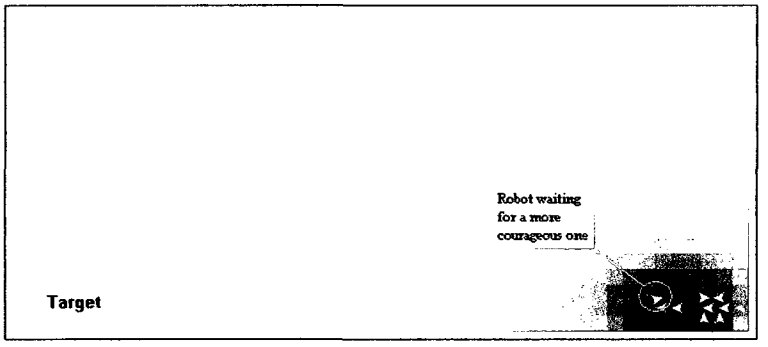

Figure 4.9: Robot waiting for a more courageous robot

The last interesting behaviour that we want to discuss is how some robots turn around and follow other more courageous robots, i.e., they wait until the courageous robots take the lead and then follow them. Figure 4.9 shows one robot that turned back and is waiting until a more courageous one passes by so she may follow her. The reason this happens is that the traits of personality "fear" and "cooperation" are much bigger than the trait "courage". Therefore, the robot is (1) afraid of being shot and (2) wanting to share the risk with other robots.

4.7 Conclusion

This chapter has presented a unique method of modeling and controlling a swarm of robots. It integrates ideas from game theory and incorporates the novel use of adaptive personality features to achieve an intelligent swarm. Three different simulations have been presented. Each simulation scenario highlights a different aspect of swarm intelligence using game theory and adaptive personalities.

The first simulation illustrated how two agents or robots can play a zero-sum game and how the agent/robot personalities would converge to the Nash equilibrium. A proof of convergence theoretically validates the method. The second simulation is an example of three robots that must cooperate in leaving a room. It is shown how the proposed method can achieve optimal performance. Furthermore, one robot converges to the always walk condition another converges to the altruistic always wait condition and the third robot converges to the mixed 50% wait and 50% walk strategy.

The third simulation illustrates how the proposed method can be used to locate a target. The effect of different robot personalities on the performance of the swarm is shown. Cooperative robots tend to huddle into a tight swarm, whereas more courageous robots leave the swarm and lead the pack. We also demonstrate that the swarm is resilient to spurious individuals. By fixing some individuals as "aggressive", we showed that it takes up to half of the swarm to change the arising group behaviour. This is an important result, since malfunctioning robots must be dealt with.

In the next chapter we will discuss how differential game theory may be used in order to model a swarm formation problem. We also introduce a simulation showing the optimal solution for the problem stated using analytical optimization methods.

Chapter 5

Swarm Formation as a Differential Game

5.1 Introduction

The problem of swarm formation has been largely discussed in the literature. It basically consists of some agents (robots, animals, vehicles, etc.) moving closely spaced to each other in some direction. They should not collide and if some obstacle is placed in their path they should be able to (1) identify it and (2) take measures to avoid the obstacle and at the same time avoid collisions with the other agents.

Many different approaches have been tried to overcome this problem. The most common approach is to determine the dynamic equations of the agents and then find a Lyapunov argument that will show the swarm is stable [36, 37]. Gazi [35] uses artificial potentials and sliding-mode control to model his swarm of agents. Olfati-Saber [82] uses heuristics to show how a multi-agent dynamic system would form a flock (or swarm in our notation). Others have tried to solve the problem by using

graphs [63]. In the field of optimization, some have used the idea of a mass-spring system to model the forces between the agents [13].

The motivation for this type of research has certainly been the observation of how certain animals behave [36]. Fish tend to group in fish schools. Several animals (zebras, antelopes, etc.) form herds in order to make themselves safer. Birds migrate in flocks. However, the most compelling example of animal organization is the insects' communities [9]: bee hives, ant colonies, etc.

We do not intend to dwell too much in the biological metaphor in this chapter. We want to provide some insight in the way robots could work together in order to solve problems. The biological metaphor just defines strategies of how this may be done. We will use the fact that most animals, supposedly, use very simple rules in order to achieve these outcomes. Also, we consider that these relationships involve conflicts (among different agents) and cooperation. For this reason, we will try to use game theory for the modelling of the system. One example of how to use game theory to model swarms may be found in [42].

As discussed in chapter 2, game theory is basically the study of decision making [74] in order to solve conflicts. In this chapter we are going to use, more specifically, the representation of *differential games*. As seen in section 2.4, differential games investigate how decision making takes place over time [110]. In order to represent the game, we need to model the dynamic equations that are related to the process under investigation. In our case, these equations are differential equations and will reflect the dynamics of the robots.

Once again, in this chapter we build upon the concepts introduced in chapter 2. We will use game theory as a tool in order to model conflicts [74] and promote

cooperation [110]. The main contribution of this chapter is in the modelling of swarm formation as a differential game. We hope that it will be clear by the end of the presentation that the problem can be effectively and efficiently represented in the domain of game theory.

The chapter is divided as follows. In section 5.2 we describe the model of a swarm of robots using the notation of differential game theory (section 2.4). Section 5.3 discusses one particular case of the model. And, finally, section 5.4 shows the conclusions that may be determined from the model.

5.2 Model

Before proceeding any further, let us restate equations (2.15) and (2.16) from section 2.4. If the game is a nonzero-sum, N-player differential game, a player i tries to choose a control signal u_i to minimize the cost equation [96]

$$J_i = q_i(\bar{x}(T)) + \int_{t_0}^{T} g_i(\bar{x}(s), u_1(s), \cdots, u_N(s), s)ds \tag{5.1}$$

subject to the state dynamics

$$\dot{\bar{x}}(s) = f(\bar{x}(s), \bar{u}_1(s), \cdots, \bar{u}_N(s), s), \quad \bar{x}(t_0) = \bar{x}_0 \tag{5.2}$$

where $\bar{x}(s) \in \mathbb{R}^m$ is the state vector of dimension m, T is the terminating time (or the time where the terminal state is reached), $q_i(\cdot)$ is the payoff of the terminal state and $g_i(\cdot)$ is the integral payoff for player $i \in N$.

Functions $q_i(\cdot)$ and $g_i(\cdot)$ are chosen in order to achieve an objective. Observe that

these functions are the same as $Q_i(\cdot)$ and $G(\cdot)$ in section 2.4. Function $f(\cdot)$ determines the dynamics of the system. They are also called the *constraints* of the system and could be represented by inequalities [96]. We also assume that one agent (or player) has access to the states of the other players involved in the game at all times. This is called the *perfect information* assumption. For our case, we also assume that each player knows the cost functions of all the others. We assume as well that all the robots have the same dynamics. Finally, one must notice that the game may have different solutions and the achievement of the optimal solution is not guaranteed for the general case.

Then, the first step for the solution of the game is the definition of the state dynamics functions $f(\cdot)$. Some assumptions must be taken into account.

i. The robots move forward at varying speeds and are subject to inertia.

ii. The robots' kinematic equations are linear.

These assumptions are taken to simplify the case study in the next section. If we had decided to use a different model, the solution of the game would be different, although the procedure would be the same.

The states of the robots are their cartesian coordinates. For robot $i \in N$ (where, again, N is the set of all robots)

$$\bar{x}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{5.3}$$

Let us assume that the motion dynamics, which evolve in continuous time, for robot $i \in N$ (where, again, N is the set of all robots) are represented by the following

differential equation for each robot's coordinates

$$\dot{\bar{x}}_i = A_i \bar{x}_i + B_i \bar{u}_i \tag{5.4}$$

where $\bar{x}_i \in \mathbb{R}^2$ is the position of the robot in the cartesian plane (5.3); $A_i \in \mathbb{R}^{2\times 2}$; $B_i \in \mathbb{R}^{2\times 2}$; and $\bar{u}_i \in \mathbb{R}^2$.

Equation (5.4) is similar to the system presented in [36], which makes explicit the nature of the forces (attraction and repulsion) among the robots

$$\dot{\bar{x}}_i = \sum_{j=1, j\neq i}^{N} \mathfrak{F}(\bar{x}_j - \bar{x}_i) \tag{5.5}$$

where $\mathfrak{F}(\cdot)$ are the forces acting on over robot i due to the other robots $j \in N$. In this case, $A_i = \mathbf{0}$ and $B_i = \boldsymbol{I} \in \mathbb{R}^{2\times 2}$, and $u_i = \mathfrak{F}(\cdot)$. One should notice that our model is related to this model. We, however, do not make any assumption about the format of the attraction/repulsion forces and they are represented by the input control signals $\bar{u}_i$ of each robot $i \in N$.

We may decide to represent all the robots' states in just one dynamic equation. This takes advantage of the fact that the game is of perfect information. In this case, the state vector becomes

$$\bar{x} = \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \vdots \\ \bar{x}_N \end{bmatrix} \tag{5.6}$$

In section 5.3 we need to deal with accelerations. In order to do that, we need to include some other states to the state vector (5.6). Let us now consider the velocity vector $\bar{v}_i$

$$\bar{v}_i = \begin{bmatrix} (v_x)_i \\ (v_y)_i \end{bmatrix} \tag{5.7}$$

The velocity state vector for the system would then be

$$\bar{v} = \begin{bmatrix} \bar{v}_1 \\ \bar{v}_2 \\ \vdots \\ \bar{v}_N \end{bmatrix} \tag{5.8}$$

The full states for the system would become

$$\bar{X} = \begin{bmatrix} \bar{x} \\ \bar{v} \end{bmatrix} \tag{5.9}$$

The dynamics would then be

$$\dot{\bar{X}} = A\bar{X} + \sum_{j=1}^{N} B_j \bar{u}_j \tag{5.10}$$

where $A \in \mathbb{R}^{4N \times 4N}$ and $B_j \in \mathbb{R}^{4N \times 2}$ are chosen such that the overall system is the same as the one defined in equation (5.4).

Furthermore, we define a terminal manifold where the robots must lie at time $t = T$

$$\psi(\bar{X}(t), t) = 0\ ,\ t = T \tag{5.11}$$

We, then, select a linear quadratic function to be our cost function. Therefore, adapting and simplifying the notation of (5.1), it becomes [87, 96]

$$J_i(t)=\frac{1}{2}[(\bar{X}^T S_{if} \bar{X})_{t=T} + \int_{t_0}^{T} (\bar{X}^T Q_i \bar{X} + \sum_{j=1}^{N} \bar{u}_j^T R_{ij} \bar{u}_j) ds] \tag{5.12}$$

where

$$\begin{aligned} q_i(\bar{X}(T))&=\frac{1}{2}(\bar{X}(T)^T S_{if} \bar{X}(T)) \\ g_i(\cdot)&=\frac{1}{2}(\bar{X}^T Q_i \bar{X} + \sum_{j=1}^{N} \bar{u}_j^T R_{ij} \bar{u}_j) \end{aligned} \tag{5.13}$$

As it is well known, the task of minimizing (5.12) in the presence of the constraints (5.4) and (5.11) is equivalent to the problem of minimizing the function

$$\begin{aligned} J_i(t)=&\frac{1}{2}(\bar{X}(T)^T S_{if} \bar{X}(T)) + \bar{\nu}_i^T \psi(\bar{X}(T), T) + \\ &\frac{1}{2} \int_{t_0}^{T} [(\bar{X}^T Q_i \bar{X} + \sum_{j=1}^{N} \bar{u}_j^T R_{ij} \bar{u}_j) + \bar{\lambda}_i^T (A\bar{X} + \sum_{j=1}^{N} B_j \bar{u}_j - \dot{\bar{X}})] ds \end{aligned} \tag{5.14}$$

subject to no constraints [87]. In (5.14), $\bar{\lambda}_i$ and $\bar{\nu}_i$ are Lagrange multipliers.

Let us now define the *Hamiltonians* $H_i(\bar{X}, \bar{u}_1, \cdots, \bar{u}_N)$ to be

$$H_i(\bar{X}, \bar{u}_1, \cdots, \bar{u}_N)=\frac{1}{2}[\bar{X}^T Q_i \bar{X} + \sum_{j=1}^{N} \bar{u}_j^T R_{ij} \bar{u}_j] + \bar{\lambda}_i^T (A\bar{X} + \sum_{j=1}^{N} B_j \bar{u}_j) \tag{5.15}$$

Then, our problem is to minimize the function

$$J_i(t)=\frac{1}{2}(\bar{X}^T S_{if}\bar{X})_{t=T} + \bar{\nu}_i^T\psi(\bar{X}(T),T) + \int_{t_0}^{T} H_i(\bar{X},\bar{u}_1,\cdots,\bar{u}_N) - \bar{\lambda}_i^T\dot{\bar{X}}ds \tag{5.16}$$

Now, integrating the last term of (5.16) by parts, we get

$$J_i(t)=\frac{1}{2}(\bar{X}^T S_{if}\bar{X}) + \bar{\nu}_i^T\psi(\bar{X}(T),T) - \bar{\lambda}_i^T\bar{X} + \int_{t_0}^{T} H_i(\bar{X},\bar{u}_1,\cdots,\bar{u}_N) + \dot{\bar{\lambda}}_i^T\bar{X}ds \tag{5.17}$$

Solving (5.17) by variational calculus, we find the control [87] that minimizes the cost functions for the i^{th} player to be

$$\bar{u}_i^*(t) = -R_{ii}^{-1}B_i^T\bar{\lambda}_i \tag{5.18}$$

Moreover, we assume that the terminal manifold (5.11) is a linear combination of the states, such that

$$\psi(\bar{X}(t),t) = F_i\bar{X}(t) - \bar{r}(T) = 0 \tag{5.19}$$

If we now return to equation (5.17), we can solve it for $\bar{\lambda}_i$ by using a sweep method [54]. Assume that $\bar{\lambda}_i = S_i\bar{X} + C_i^T\bar{\nu}$, then (5.18) becomes

$$\bar{u}_i^*(t) = -R_{ii}^{-1}B_i^T(S_i\bar{X} + C_i^T\bar{\nu}) \tag{5.20}$$

with the final conditions $S_i(T) = S_{if}$ and $C_i(T) = F_i^T$.

Taking into account the transversality conditions [87], one may solve (5.20) in terms of $\bar{X}$ to get

$$\bar{u}_i^*(t)=-(R_{ii}^{-1}B_i^TS_i - R_{ii}^{-1}B_i^TC_iV_i^{-1}P_i)\bar{X} - R_{ii}^{-1}B_i^TC_i^TV_i^{-1}\bar{r}(T) \tag{5.21}$$

where all R_{ii} are positive definite (otherwise the problem makes no sense [96]).

Matrices S_i, C_i, P_i and V_i are found according to the equations

$$\begin{aligned}
\dot{S}_i&=-S_iA - A^TS_i - Q_i - \sum_{j=1}^{N}(S_jB_jR_{jj}^{-1}R_{ij}R_{jj}^{-1}B_j^TS_j - S_iB_jR_{jj}^{-1}B_j^TS_j - S_jB_jR_{jj}^{-1}B_j^TS_i)\\
\dot{C}_i&=-A^TC_i - \sum_{j=1}^{N}(S_jB_jR_{jj}^{-1}R_{ij}R_{jj}^{-1}B_j^TC_j - S_iB_jR_{jj}^{-1}B_j^TC_j - S_jB_jR_{jj}^{-1}B_j^TC_i)\\
\dot{P}_i&=-P_iA + \sum_{j=1}^{N}P_iB_jR_{jj}^{-1}B_j^TS_j\\
\dot{V}_i&=\sum_{j=1}^{N}P_iB_jR_{jj}^{-1}B_j^TC_j
\end{aligned} \tag{5.22}$$

subject to the final constraints

$$\begin{aligned}
S_i(T)&=S_{if}\\
C_i(T)&=F_i^T\\
P_i(T)&=F_i\\
V_i(T)&=\mathbf{0}
\end{aligned} \tag{5.23}$$

In section 5.3, we are going to show how the model developed in this section could be applied in order to have three robots forming a swarm. As it may be clear by now, the control signals $\bar{u}_i(t)$ may be thought of as forces (accelerations). This, relates the

results presented in here with the results in [36].

5.3 Study case

Let us consider three identical robots randomly positioned in a small plane (20×20) centered at point $(0, 0)$. Since they are identical, their motion (kinematic) equations are the same. Let us start by ignoring external forces. Observe that that would not be a problem in itself, it would just make the problem more cumbersome.

The kinematic equations for each robot $i \in N$ is

$$\dot{\bar{x}}_i = \bar{v}_i = \begin{bmatrix} (v_x)_i \\ (v_y)_i \end{bmatrix} \tag{5.24}$$

Now, let us assume we are interested in the distances among some of the robots plus the position of the center of the swarm. Define then the transformation

$$\bar{z} = A_z \bar{x} \tag{5.25}$$

where

$$A_z = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \tag{5.26}$$

In the same way, we define the velocity vector $\bar{w}$ to be $\bar{w} = A_z \bar{v}$. Since we want

only one state vector, we have

$$\bar{X} = \begin{bmatrix} \bar{z} \\ \bar{w} \end{bmatrix} = \begin{bmatrix} A_z & \mathbf{0} \\ \mathbf{0} & A_z \end{bmatrix} \begin{bmatrix} \bar{x} \\ \bar{v} \end{bmatrix} \tag{5.27}$$

Defining, at last, $\bar{a}_i = \bar{u}_i$ to be the acceleration (control signal) for the i^{th} robot, the kinematics equation become

$$\dot{\bar{X}} = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix} \bar{X} + \sum_{i=1}^{N} \mathbf{B_i} \bar{u}_i \tag{5.28}$$

Matrices $\mathbf{B_i}$ are

$$\mathbf{B_1} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{I} \\ \mathbf{I} \\ \mathbf{I} \end{bmatrix}, \mathbf{B_2} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{I} \\ -\mathbf{I} \\ \mathbf{I} \end{bmatrix}, \mathbf{B_3} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{I} \\ \mathbf{0} \\ -\mathbf{I} \end{bmatrix} \tag{5.29}$$

where $\mathbf{0}$ and $\mathbf{I}$ are matrices in $\mathbb{R}^{2\times 2}$.

The next step is to choose the cost criteria for each robot. Observe that this choice will determine the behaviour of the robots. We then choose

$$J_i = \frac{1}{2}\sigma_i^2 \bar{X}_f^T \bar{X}_f + \bar{\nu}_i^T \psi(\bar{X}_f, T) + \frac{1}{2}\int_0^T \left(\frac{\bar{u}_i^T \bar{u}_i}{r_{ii}} + \sum_{j=1; j\neq i}^{N} \frac{\bar{u}_j^T \bar{u}_j}{r_{ij}}\right) dt \tag{5.30}$$

where $\bar{X}_f = \bar{X}(T)$ and the final time T is fixed. The terminal manifold is chosen to

be

$$\psi(\bar{X}(T), T) = \bar{X}_f - \bar{r}(T) = 0 \tag{5.31}$$

where $r(T)$ is the desired final position.

The parameters σ_i and, r_{ii} are positive numbers and represent the energy the robots have at their disposal, which is supposed to be finite [51]. The numbers r_{ij} may be positive or negative. They relate to the terms of (5.12) in the following way

i. $S_{if} = \sigma_i^2 \boldsymbol{I}$

ii. $Q_i = \mathbf{0}$

iii. $R_{ij} = r_{ij}^{-1}$

One of the several different ways we may solve (5.30) (as mentioned in the last section) is by variational calculus [62] together with a sweep method [66]. By doing that, we end up with the following Hamiltonians for the robots $i \in N$

$$H_i = \frac{1}{2}\left(\frac{\bar{u}_i^T \bar{u}_i}{r_{ii}} + \sum_{j=1; j\neq i}^{N} \frac{\bar{u}_j^T \bar{u}_j}{r_{ij}} + \bar{\lambda}_i^T \left\{ \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix} \bar{X} + \sum_{j=1}^{N} B_j \bar{u}_j \right\}\right) \tag{5.32}$$

Then, following (5.21) and (5.22), the control signals are

$$
\begin{aligned}
\bar{u}_i(t) &= -(r_{ii}B_i^T S_i - r_{ii}B_i^T C_i V_i^{-1} P_i) \begin{bmatrix} \bar{z} \\ \bar{w} \end{bmatrix} - r_{ii}B_i^T C_i^T V_i^{-1} r(T) \\
\dot{S}_i &= -S_i \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ I & 0 \end{bmatrix} S_i + r_{ii} S_i B_i B_i^T S_i - \sum_{j=1;j\neq i}^{N} \left(\frac{r_{jj}^2}{r_{ij}} S_j B_j B_j^T S_j - 2{r_{jj}}^2 S_i B_j B_j^T S_j\right) \\
\dot{C}_i &= -\begin{bmatrix} 0 & 0 \\ I & 0 \end{bmatrix} C_i + r_{ii} S_i B_i B_i^T C_i - \sum_{j=1;j\neq i}^{N} \left(\frac{r_{jj}^2}{r_{ij}} S_j B_j B_j^T C_j - 2{r_{jj}}^2 S_i B_j B_j^T C_j\right) \\
\dot{P}_i &= -P_i \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix} + \sum_{j=1;j\neq i}^{N} r_{jj} P_i B_j B_j^T S_j \\
\dot{V}_i &= \sum_{j=1}^{N} r_{jj} P_i B_j B_j^T C_j
\end{aligned}
\tag{5.33}
$$

and the boundary conditions are

i. $S_i(T) = \sigma_i^2 \boldsymbol{I}$

ii. $C_i(T) = \boldsymbol{I}$

iii. $P_i(T) = \boldsymbol{I}$

iv. $V_i(T) = \mathbf{0}$

The differential equations in (5.33) may be easily integrated numerically and the control signal may be found uniquely. Notice that we assume that matrices V_i are non-singular. In the case above they are, since the matrices B_i are such that the reachability condition for $\dot{V}_i$ is satisfied. Stating it differently, we could say that

matrices B_i, $i \in N$, are such that the weighted reachability Gramian [66] defined by $V_i(t) = \int_{t_0}^{T} P_i B_j B_j^T C_j$ has rank equal to n. We could also argue that the transformation matrix A_z in (5.26), which is obviously related to the matrices B_i, must have full rank.

Let us now consider a practical example in order to illustrate the behaviour of the system. Consider that the three robots are located at points $(-6.54, 9.59)$, $(-4.57, -4.95)$ and $(7.51, 4.75)$. The objective of the game is to have them to move towards each other until they get to the desired position (at time T)

$$r(T) = \begin{bmatrix} 0 & 0 & 1 & 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{5.34}$$

The first two zeros mean that the robots should huddle together around the point $(0, 0)$. The following two terms (two 1's) mean that the relative position of robots 1 and 2 should be a vector of magnitude $\sqrt{2}$ and an angle of 45^o. The next two terms (-1 and 1) mean that robots 1 and 3 should be positioned at a distance of $\sqrt{2}$ of each other at an angle of 135^o. Observe that the relative position of robots 2 and 3 is completely determined by the previous conditions. The remaining zeros mean the the relative velocity among the robots should be zero.

Finally, we set the initial conditions to

$$\sigma_i = 1 \ , \ R = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix} \tag{5.35}$$

We then solve equation (5.33) to find the control signal for each robot. Simulating the system with the control signals, we observe the behaviour depicted in figure 5.1.

One may notice that the robots move steadily towards the point $(0, 0)$.

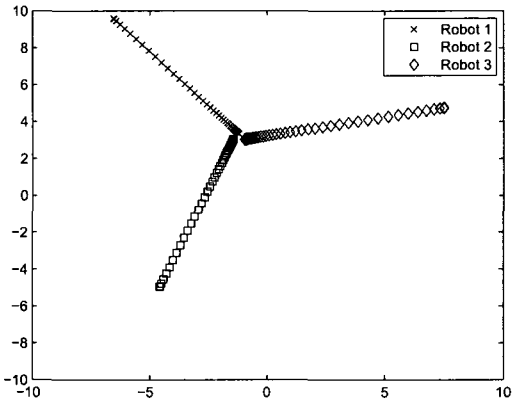

Figure 5.1: Robots swarming together.

The control signals for each one of the robots are shown in figures 5.2, 5.3 and 5.4. As it may be noticed, the control signals are smooth functions. Moreover, as it may be noticed by the control signals in figures 5.2-5.4 as well as the movement in figure 5.1, the robots accelerate and then decelerate until they come to a full stop (as depicted in figure 5.1) at the desired position. Observe also that the final position (or desired manifold) of the robots could be different than the one chosen. As a matter of fact, we could have defined more complex manifolds for the final position. However, for the sake of simplicity, we chose it to be a simple linear function. What is more important is the fact the robots do huddle together close to each other.

5.4 Conclusion

In this chapter, we showed that the swarm robotics problem may very well be seen as a multiplayer differential game analogous to several other games in the literature

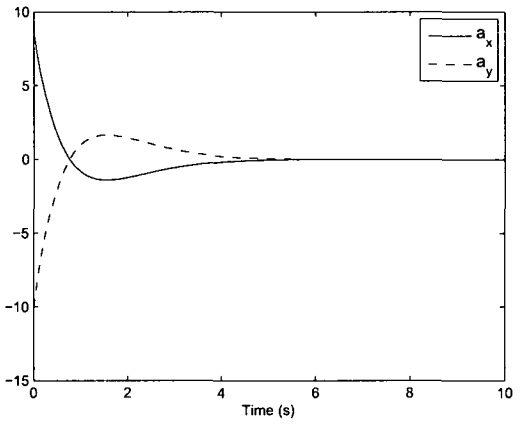

Figure 5.2: Control signal for robot 1.

[110, 14].

The kinematic equations for robots representing the change in their positions were introduced in section 5.2. It may be noticed that they may be simulated very easily. Then, we introduced the problem of swarming as a system of linear differential equations linked together by the objective of the game. It was also shown that by solving the equations by an analytical optimization method, we would be able to find a good solution for the control signals or strategies of players.

In section 5.3, we showed simulations of an instance of the problem introduced in section 5.2. It was shown how to calculate the optimal control signals of three players and the resulting behaviour of coming together as a swarm. These results point to the possibility of, in the future, modelling the problem of swarm formation as only one game.

In the next section we are going to model another problem as a differential game. However, this time, we are going to consider the dynamics of real robots. Also we are

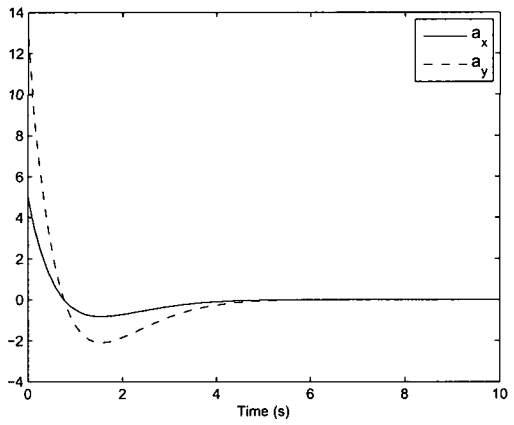

Figure 5.3: Control signal for robot 2.

going to consider how a robot could learn a strategy without explicitly solving the game equations.

In the future, we intend to take into account the dynamics of actual robots as well as running experiments in order to prove our concept.

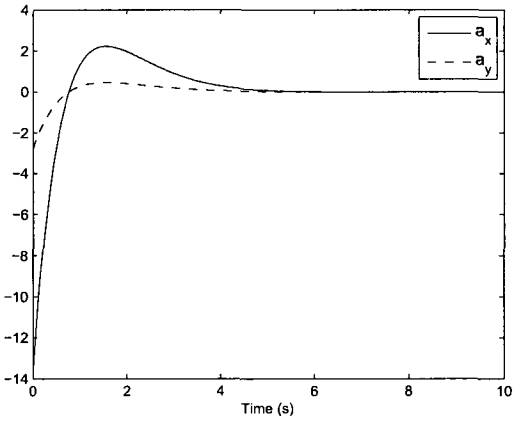

Figure 5.4: Control signal for robot 3.

Chapter 6

Learning in a Differential Game

6.1 Introduction

In this chapter, we are going to approach a multiple robot differential game from the point of view of learning and the question we want to answer is whether the game *converges* to an equilibrium point. As presented in chapter 2, in the particular case of two players, this point may be the *Nash* equilibrium or the *value* of the game. However, this is not guaranteed, since the game is not seen as only an optimization problem at this level.

In the particular case of differential games, one of the most popular learning approaches has been the use of reinforcement learning and Q-learning [46, 45]. However, there is a disadvantage in this technique when we deal with continuous processes such as the ones considered in differential games. Since Q-learning is based on the construction of tables, several different actions must be considered coupled with states in order to describe the possible behaviours of the players. This could lead to a proliferation of updates that would make the approach infeasible for implementation in

a microcontroller. Moreover, it is not easy to discretize the action space as well as the state space [24].

In order to avoid this problem, one could use a fuzzy controller. It is well known that a fuzzy system is a universal approximator [106]. Therefore, we propose in this work a fuzzy controller that is updated by a reinforcement learning algorithm. The advantages of such approach are:

- A fuzzy controller can deal with noisy data [106] and uncertainties [1].
- Reinforcement learning updates is an adequate way of updating the fuzzy rules on line [1].
- We could use a technique based on neural networks or genetic algorithms to extract rules from data and then apply the approach proposed in order to speed up the convergence of the controller.
- A fuzzy controller such as presented here could be easily implemented in a microcontroller.

We may think of the pursuit-evasion differential game in two different scenarios. The first is the typical one-on-one game, i.e., there is only one pursuer and one evader. This game has been extensively studied in the literature [55, 73, 96]. However, little attention has been given to how a player may learn how to play the game. The other more complicated scenario is the multiplayer pursuit-evasion differential game, wherein there may be several pursuers and/or evaders. This is a less studied case, but interesting results have also been published [38, 39]. In these papers, a hierarchical structure is proposed which is based on numerical or analytical solution of the pursuit-evasion game. Our learning approach, however, does not dwell on the mathematical

solution of the games. We are interested in how the players "learn" to play a game as it has been done in traditional game theory [2, 70]. Therefore, our fuzzy controller obtained after training (or even during the execution of the task) would determine the behaviour of the robot. Moreover, it must be noted that the approach presented in this chapter is not only applicable to pursuit-evasion games. Other dynamic games would have the same type of learning algorithm presented in here.

The chapter is divided as follows. In section 6.2 we present the control structure for the system. Section 6.3 reviews the learning techniques used in games in general and introduces a fuzzy algorithm for learning in differential games. In section 6.4, we describe the system that will be used in the simulations and experiments as well as how it relates to the notation in section 2.4. The hardware and software framework will also be introduced in this section. Section 6.5 reports the identification of the robots' models. In section 6.6, simulations of the system identified in the previous sections are presented. Section 6.7 presents the experimental results reached with the controller derived in the simulation section. And, finally, section 6.8 presents our conclusions from the simulations and experiments and points to future work that will be done in the field.

6.2 Controller Structure

In figure 6.1 we show the proposed structure for the controller [18].

In this section, we are going to focus on the structure of each block, more specifically the *controller* and the *critic*. The *reinforcement* block is particular to applications and will be dealt with in section 6.6. Also, the learning procedures will be postponed until the next section.

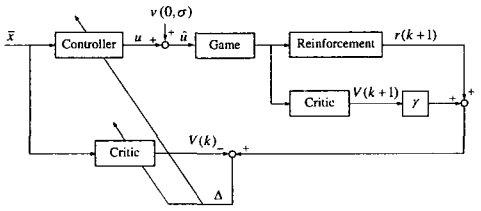

Figure 6.1: Architecture of the control system

We assume that the *controller* in figure 6.1 is a fuzzy controller. More specifically, it is a fuzzy controller implemented by Takagi-Sugeno (TS) rules with constant consequents [101]. It consists of M rules with n fuzzy variables as inputs and one constant number as consequent. Therefore, each rule is of the form [24]

$$R_l : IF\, x_1\, is\, F_1^l, \cdots, and\, x_n\, is F_n^l \tag{6.1}$$

$$THEN\, u = c_l \tag{6.2}$$

where x_i are the values passed to the controller, F_i^l is the fuzzy set related to the corresponding fuzzy variable, u is the rule's output, and c_l is a constant that describes the center of a fuzzy set.

Therefore, if we use the product inference for fuzzy implication [106], t norm, singleton fuzzifier and center-average defuzzifier, the output of the system is [24]

$$u(\bar{x}) = \frac{\sum_{l=1}^{M}((\prod_{i=1}^{n} \mu^{F_i^l}(x_i)) \cdot \chi_l)}{\sum_{l=1}^{M}(\prod_{i=1}^{n} \mu^{F_i^l}(x_i))} \tag{6.3}$$

where c_l in (6.2) is represented by χ_l for the controller.

Throughout the chapter, the membership functions used are triangular ones such

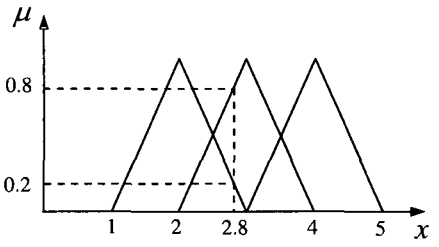

Figure 6.2: Triangular membership functions used

as the ones depicted in figure 6.2.

For the *critic*, we also assume a TS system with constant consequents [18]. However, it must be noted that this is not the only possible choice. A time-delay neural network (TDNN) could be used instead [24]. There are advantages and disadvantages in this choice for both cases. We chose the fuzzy system just for its simplicity. Therefore, just as in the case of (6.3), the output of the critic is [18] an approximation to the value of the state $V(\cdot)$

$$\hat{V}(\bar{X}) = \frac{\sum_{l=1}^{M}((\prod_{i=1}^{n} \mu^{F_i^l}(x_i)) \cdot \zeta_l)}{\sum_{l=1}^{M}(\prod_{i=1}^{n} \mu^{F_i^l}(x_i))} \tag{6.4}$$

where c_l in (6.2) is represented by ζ_l.

6.3 Learning

In the context of Game Theory, learning may be understood as strategy adjustments [2]. These adjustments may be functions of several different sets of variables.

Different types of continuous-time models of learning have been proposed for games. In the case of games in extensive form, [2] describes some models discussed in the literature, including the well known *replicator dynamics*, where one player tries to approximate the way the other (or others) will play in order to decide which strategy is more profitable for it to play. However, in the special case of differential games, the learning techniques presented in the literature are very different. Most of them are based on reinforcement learning [94].

Let us now consider a differential game between two players. Let us also assume that all the robots have the same dynamics. And lastly, one must notice that the game may have different solutions and the achievement of the optimal solution is not guaranteed for the general case.

Since we have only two players, equations (2.16) may be simplified to [56]

$$\dot{\bar{x}}(s) = f(\bar{x}(s), \bar{\phi}(s), \bar{\psi}(s), s), \quad \bar{x}(t_0) = \bar{x}_0 \tag{6.5}$$

where $\bar{\phi}$ and $\bar{\psi}$ are the strategies played by each player. The payoff, now represented as $P(\bar{\phi}, \bar{\psi})$, is given in the form

$$P(\bar{\phi}, \bar{\psi}) = q(t^*, \bar{x}(t^*)) + \int_{t_0}^{t^*} g(\bar{x}(s), \bar{\phi}, \bar{\psi}, s)ds \tag{6.6}$$

where t^* is the first time the states $\bar{x}(t)$ intersect a given final condition. In this case it is also assumed that the player who uses strategy $\bar{\phi}$ wants to maximize the payoff $P(\cdot)$, whereas the player using strategy $\bar{\psi}$ wants to minimize it. Therefore, the

objective of the game is to find control signals $\bar{\phi}^*$ and $\bar{\psi}^*$ such that [17]

$$P(\bar{\phi}^*, \bar{\psi}) \leq P(\bar{\phi}^*, \bar{\psi}^*) \leq P(\bar{\phi}, \bar{\psi}^*), \quad \forall \bar{\phi}, \bar{\psi} \tag{6.7}$$

Notice that equation (6.6) is equation (2.15) considering that we have only two players. Notice also that the approach presented here may be used for any number of players, but, for simplicity sake, we present the technique for just two-player games. In this case, the learning dynamics may be described by

$$\dot{\bar{\delta}}_1 = f_{\bar{\delta}_1}(\bar{x}, \bar{\delta}_1, \bar{\delta}_2) \tag{6.8}$$

$$\dot{\bar{\delta}}_2 = f_{\bar{\delta}_2}(\bar{x}, \bar{\delta}_1, \bar{\delta}_2) \tag{6.9}$$

Observe that the functions $f_{\bar{\delta}_1}(\cdot)$ and $f_{\bar{\delta}_2}(\cdot)$ must take into consideration the cost (6.6) in such a way that the *saddle point* (for the case of a two-player game) in (6.7) is reached. Also, the strategy adjustments in (6.8) and (6.9) will be functions of the strategies played by *both* players at a given time as well as their states. Therefore, we assume that the players play their strategies synchronously. Also, we assume that they know the strategies (the control signal) played by all the other players, although they do not know how they chose those strategies.

The problem of (6.8) and (6.9) is that the strategies played by each player are continuous. This means that the strategy vectors $\bar{\delta}_1$ and $\bar{\delta}_2$ should have dimensions of infinity. Another approach would be to try to discretize the action space and then use a Q-learning algorithm to calculate the strategy vectors. Although, there is another problem; we would have to discretize the state space as well and this is not easily done for most differential games. In order to avoid these potential problems, we use

the architecture shown in figure 6.1.

In this figure we see the addition of two blocks called *critic*. This block is used to approximate the value function for reinforcement learning [24]. It could be implemented by a time-delay neural network [24] or a fuzzy system [18]. The learning is practically the same for both cases. The value function for approximation of the reinforcement rewards has the format

$$V(k) = E\{\sum_{i=k}^{\infty} \gamma^{i-k} r(i+1)\} \tag{6.10}$$

where $\gamma \in [0, 1)$ is known as the forgetting factor and $r(\cdot)$ is the immediate external reward from the environment. Notice that we can rewrite (6.10) in a recursive fashion as

$$V(k) = r(k+1) + \gamma V(k+1) \tag{6.11}$$

With this approximation, we may compare it with the expected reward such that we generate a prediction error of the prediction $\hat{V}(k)$ [18] that is the output of the critic so that

$$\Delta = [r(k+1) + \gamma \hat{V}(k+1)] - \hat{V}(k) \tag{6.12}$$

as shown in figure 6.1. This difference error is then used to train the critic. Supposing it has parameters ζ to be adapted, the adaptation law would then be [24]

$$\zeta_j(k+1) = \zeta_j(k) + \alpha \Delta \frac{\partial \hat{V}(k)}{\partial \zeta_j} \tag{6.13}$$

where $\alpha \in (0, 1)$ is the learning rate for the adaptation. Observe that we do not want α to be too big in order to avoid instability in the generated system. Also the partial

derivative in (6.13) is easily calculated from (6.4) to be

$$\frac{\partial \hat{V}(k)}{\partial \zeta_j} = \frac{\prod_{i=1}^{n} \mu^{F_i^j}(x_i)}{\sum_{l=1}^{M}(\prod_{i=1}^{n} \mu^{F_i^l}(x_i))} \tag{6.14}$$

The controller in figure 6.1 is a fuzzy controller implemented by Takagi-Sugeno rules with constant consequents [101]. Observe that to its generated control signal $u(k)$ is added a random white noise $v(0, \sigma)$. This is done in order to promote exploration of the action space [24]. With the noisy signal $u'(t)$, taking χ as the parameters to be adapted, we may establish the controller update law [18]

$$\chi_j(k+1) = \chi_j(k) + \beta\Delta[\frac{u'(k) - u(k)}{\sigma}]\frac{\partial u(k)}{\partial \chi_j} \tag{6.15}$$

where $\beta \in (0, 1)$ is the learning rate for the controller adaptation. Note that we want $\beta < \alpha$, meaning that we want the controller to converge slower than the critic. Also notice that the initial fuzzy controller can give a bad performance for the player. The partial derivative in (6.15) is easily calculated to be from (6.3)

$$\frac{\partial u(k)}{\partial \chi_j} = \frac{\prod_{i=1}^{n} \mu^{F_i^j}(x_i)}{\sum_{l=1}^{M}(\prod_{i=1}^{n} \mu^{F_i^l}(x_i))} \tag{6.16}$$

Notice that the partial derivatives in (6.14) and (6.16) are rigorously the same and need to be calculated just once. Also notice that at no time we have a "desired" trajectory and, therefore, we have no error signal. The update laws in (6.13) and (6.15) are based on a "forecasted" improvement of the cost function (equation (6.6)).

In the next section, we describe the game we will solve with the technique described so far. In section 6.6 we present the particular case of the equations presented in this section to the problem under study.

6.4 Pursuer Evader Model

In this section we will introduce the model used in the coming simulations and experiments. The section will be divided in two subsections. In the first one, we will describe the general mathematical model for the robots and the game itself. The second subsection will describe the hardware involved in the experiments, consisting of the communication framework and how the robots were constructed.

6.4.1 Mathematical Model

Let us assume that we have a game where two robots are present. One of them is called the *pursuer* and tries to catch another one, called the *evader*. The model (kinematic equations) for the pursuer may be represented by

$$\begin{aligned} \dot{x}_p &= V_p \cos(\theta_p) \\ \dot{y}_p &= V_p \sin(\theta_p) \\ \dot{\theta}_p &= \frac{V_p}{R_p}\delta_p \end{aligned} \tag{6.17}$$

In the same way, the model for the evader may be described as

$$\begin{aligned} \dot{x}_e &= V_e \cos(\theta_e) \\ \dot{y}_e &= V_e \sin(\theta_e) \\ \dot{\theta}_e &= \frac{V_e}{R_e}\delta_e \end{aligned} \tag{6.18}$$

In (6.17) and (6.18), x_i and y_i, $i = \{e, p\}$ are the positions of the robots; V_p is the speed of the pursuer, V_e is the speed of the evader, and $V_p > V_e$; θ_p is the orientation of the pursuer and $|\theta_p| < \pi$; θ_e is the orientation of the evader and $|\theta_e| < \pi$; R_p is the turning radius of the pursuer, R_e is the turning radius of the evader, and $R_p > R_e$ and they are such that $\frac{V_p}{R_p} < \frac{V_e}{R_e}$. Finally, $|\delta_p| \leq 1$ is the control signal for the pursuer and $|\delta_e| \leq 1$ is the control signal for the evader. In words, the pursuer is faster, but the evader can make sharper turns. This game is known as the "game of two cars" [55].

Let us now assume a coordinate frame centered in the pursuer with its y'-axis in the direction of the pursuer's velocity vector [17] as shown in figure 6.3. The pair (x', y') is the relative position of the evader in this coordinate frame [17].

If we define $\psi = \theta_e - \theta_p$, then the differential equations for this game are of the form

$$\dot{x'} = V_e \sin\psi - \frac{V_p}{R_p} y' \delta_p \tag{6.19}$$

$$\dot{y'} = V_e \cos\psi - V_p + \frac{V_p}{R_p} x' \delta_p \tag{6.20}$$

$$\dot{\psi} = -\frac{V_p}{R_p}\delta_p + \frac{V_e}{R_e}\delta_e \tag{6.21}$$

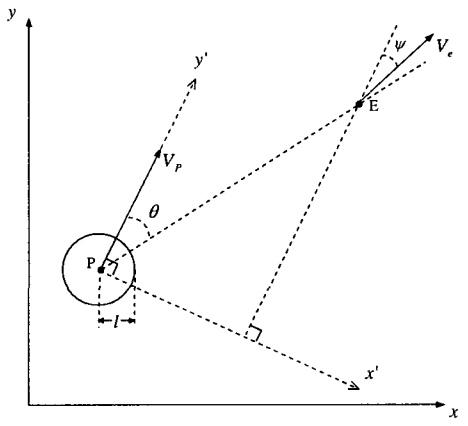

Figure 6.3: Relative position for the evader with respect to the pursuer

Equations (6.19) to (6.21) describe how the vector from the point P in figure 6.3 to point E in the same figure behaves over time. Let us now define that interception happens when

$$x'^2 + y'^2 \leq l^2 \tag{6.22}$$

where l is an arbitrarily defined distance from the pursuer to the evader.

The cost equation (6.6) in this particular case is such that

$$P(\delta_p, \delta_e) = (x'^2 + y'^2)|_{t=t_f} + \int_{t_0}^{t_f} 1 ds \tag{6.23}$$

Therefore, the functions $q(\cdot)$ and $g(\cdot)$ in (6.6) are

$$q(\bar{x'}(T)) = x'^2 + y'^2 \tag{6.24}$$

$$g(\bar{x'}(s), \delta_p(s), \delta_e(s), s) = 1 \tag{6.25}$$

Observe that the term $\int_{t_0}^{t_f} 1ds$ represents only a cost on the time to capture. The Hamiltonian is then found as [17]

$$H = \lambda_{x'}\dot{x'} + \lambda_{y'}\dot{y'} + \lambda_{\psi}\dot{\psi} + 1 \tag{6.26}$$

where $\lambda_{x'}$, $\lambda_{y'}$ and λ_{ψ} are lagrange multipliers.

By solving the hamiltonian, we may find the optimal play. Another approach to find the best capture time is by solving the similar equation [73]

$$\begin{aligned}\min_{\phi}\max_{\psi}(\frac{dP}{dt}) = \min_{\phi}\max_{\psi}\ [\ & P_{x'}(V_e\sin\psi - \frac{V_p}{R}y'\phi) \\ & P_{y'}(V_e\cos\psi - V_p + \frac{V_p}{R}x'\phi) \\ & P_{\psi}(-\frac{V_p}{R_p}\delta_p + \frac{V_e}{R_e}\delta_e)] \\ =& -1\end{aligned} \tag{6.27}$$

where P is the cost function of the game and $P_{x'}$, $P_{y'}$ and P_{ψ} are, respectively, its partial derivatives with respect to x', y' and ψ.

Solving either way, the optimal control for the pursuer and evader may be found (given some constraints) such that [17]

$$\delta_p = \mathrm{sgn}(\lambda_{x'}y' - \lambda_{y'}x' + \lambda_{\psi}) \tag{6.28}$$

$$\delta_e = \mathrm{sgn}(\lambda_{\psi}) \tag{6.29}$$

If we define some constraints and values for the parameters in (6.17) and (6.18), the solution may be further simplified, but this is not our objective here. The interested

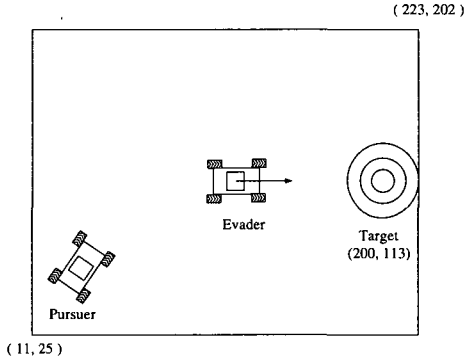

Figure 6.4: Conceptualization frame where the game is played

reader is referred to [55] and [17] for more details.

However, for the experiment we change the objectives of the game. The reason for that is due to the limitations of the experimental setup to be discussed in the next section. A game of two cars would tend to last longer and run farther than the range that our sensors would allow. Therefore, we use a variation where there is a target that the evader tries to reach while the pursuer tries to intercept the evader. This is depicted in figure 6.4. The pursuer is also supposed to be farther to the target than the evader. The strategy that the evader follows is to get the closest to the target as possible. In our simulations, the pursuer is assumed to not know the dynamics of the evader and its strategy.

Some remarks may be made with regard to equations (6.17) and (6.18). First, notice that the transfer functions for θ_p and θ_e describe an integrator. This is not quite the case, but, as it will be discussed in section 6.5, this is a good assumption. Moreover, in an actual system, V_p and V_e are not constant. There is some transient response that must be taken into account. Therefore, in section 6.5 we will also

identify equations for this transient response, such that we will have

$$\frac{V_p(s)}{u_p(s)} = \frac{b_p}{s + a_p} \tag{6.30}$$

$$\frac{V_e(s)}{u_e(s)} = \frac{b_e}{s + a_e} \tag{6.31}$$

Furthermore, we are actually going to identify the transfer function for the angular velocity instead of the one for the angle. This means that, for the pursuer, we will identify the transfer function

$$\frac{\omega_p(s)}{\delta(s)} = \frac{\frac{V_p}{R_p}}{s + a_{\omega_p}} \tag{6.32}$$

The same identification procedure will be performed for the evader, resulting in

$$\frac{\omega_e(s)}{\delta(s)} = \frac{\frac{V_e}{R_e}}{s + a_{\omega_e}} \tag{6.33}$$

However, as it will be seen in section 6.5 the poles of (6.32) and (6.33) will be very "fast" and the transfer functions will be similar to an integrator, which makes the approximations in (6.17) and (6.18) valid.

6.4.2 Experimental System

We want to implement the mathematical model of the previous subsection in an experimental set up. In order to do that, we built some robots in our lab. Fig 6.5 depicts one individual robot that may be used as pursuer or evader.

This robot is equipped with a Motorola 68HC11 microprocessor and two motors that may drive the robot forward or backward. By changing the PWM duty cycle in each motor, it is also possible to turn the robot to the right and to the left.

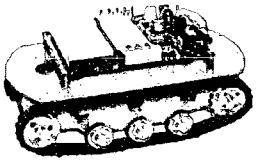

Figure 6.5: An individual robot

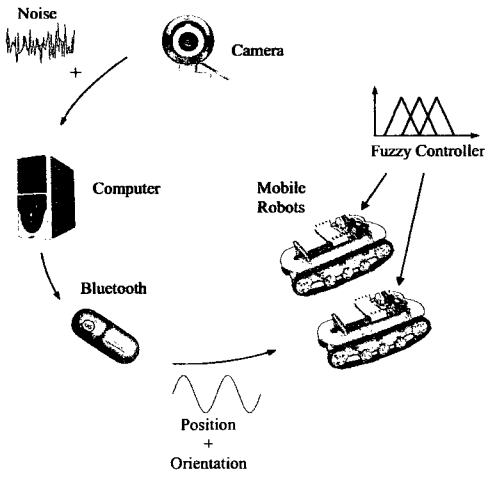

Figure 6.6: Experimental system used for implementing the differential game

The whole system for the experiment is shown in figure 6.6 and it may be divided in the following modules:

- Sensory - a camera and the filtering system coupled to it;
- Communication - after receiving the information from the sensors, it has to be sent to the robots;
- Controllers - implemented directly in the robots.

The first thing that must be decided is the information that is required to be

measured. Equations (6.17) and (6.18) require that we know the positions and orientations of each one of the robots. Therefore, the camera has to measure the states of all the robots. So, a webcamera is used to read the positions (in pixels) and the orientations (in radians) of each robot. The positions and orientations are measured by the use of a colour code. Each robot has two rectangles over it as shown in figure 6.7. A system implemented in the computer station runs filters that locate the center of such rectangles. By making use of this data, the position and the orientation may be acquired. Obviously, the data is intrinsically noisy. However, it is supposed that the controllers may deal with such noise. Also, we assume that such noise is Gaussian and white. Moreover, the camera is not synchronous and the time elapsed between measurements is not constant. These are supposed to be nonlinear effects (or noise) and we suppose the controllers have to deal with it as well.

The message is then sent from the camera to a computer. The computer, on its turn, assembles a packet with the positions and orientations of each robot. This message is finally sent to the mobile robots through a bluetooth link. Notice that both robots receive the same set of data. However, each one of them will then handle the information differently. There is no protocol that guarantees the delivery of the messages and some packets may not be received correctly. Indeed, this is the case in our experiments.

Both robots have embedded controllers that will determine the strategy that each one of them will execute. The pursuer is supposed to run the controller structure defined in section 6.2. The evader, however, runs a simpler control law that only guides it towards a fixed target as shown in figure 6.7. Notice also that the measurements of the angles depicted in figure 6.7 are also noisy.

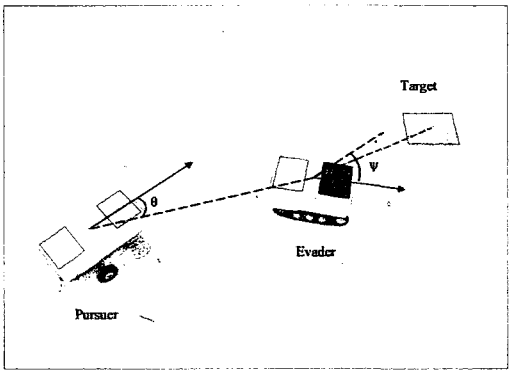

Figure 6.7: An evader and a pursuer robots

In order to simulate the equations (6.17) and (6.18), we have to first identify the transfer functions for the angle variation and the speed of each robot. This will be done in the next section.

6.5 Identification

In order to simulate the system of equations presented in section 6.4.1, we need to identify the parameters of the equations. Namely, we need to know what are the velocities V_p in (6.17) and V_e in (6.18). Moreover, we need to determine the differential equations for θ_p in (6.17) and θ_e in (6.18).

The necessity of having a model is that we are going to pre-train the controller offline in order to have some useful control signal for our experiments. Clearly, the modeling is only approximate. This is not a big problem, for a fuzzy controller should be able to deal with such uncertanties [106]. Moreover, we assume that the dynamics of the robots change slowly and smoothly over time. Therefore, the online adaptation would take care of these small variations.

For each robot, we are going to identify two difference equations that determine how the robot behaves. The first one is how the forward velocity of the robot changes when a nominal signal $u(t)$ is applied to both motors. The second one is related to the angular velocity when a specific differential signal δ is applied to the motors.

The relationship between the signals $u(t)$ and $\delta(t)$ is as follows. Let us suppose that we decide to drive the robot around a nominal input of $u(t) = 50$, where 50 is related to the duty cycle of the PWM. The forward speed of the robot increases until it reaches a steady state. Then we decide to turn the robot to the right. In order to do this, we increase the duty cycle of the left motor by $\delta(t) = 25$, where 25 is an experimentally determined value, and decrease the duty cycle in the right motor by the same value, such that, if $u_R(\cdot)$ is the signal for the right motor and $u_L(\cdot)$ is the signal for the left one. Mathematically, the signals to each motor are represented by

$$u_R(t) = u(t) - \delta(t) \tag{6.34}$$

$$u_L(t) = u(t) + \delta(t) \tag{6.35}$$

Observe that the forward speed is unchanged by the changes of the signals of the motors, since the average input is still the same.

After some tests, we determined that a first order transfer function was enough to represent the behaviours listed above. For the forward velocity, we want to identify the parameters a_v and b_v of the difference equation

$$v[k+1] = a_v v[k] + b_v u[k] \tag{6.36}$$

Observe that the velocities V_p and V_e in (6.17) and (6.18) are the steady state

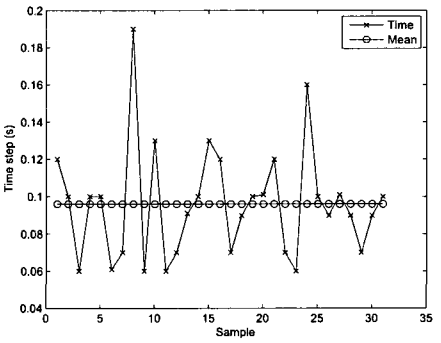

Figure 6.8: Time in one arbitrary run of a robot

values of (6.36) for each one of the robots.

For the angular velocity, we want to identify the parameters a_ω and b_ω of the difference equation

$$\omega[k+1] = a_\omega \omega[k] + b_\omega \delta[k] \tag{6.37}$$

The data collection is done through successive runs of the robots for different values of the signal $\delta(t)$. The nominal value for the motors $u(t)$ is the same for all the runs. Since, as discussed in the previous section, the time step for the measurements is not constant, the first step is to determine the mean value for this time. This is shown in figure 6.8. Notice that the time steps vary a lot. Any difference around the mean value depicted in this figure is considered as nonlinear noise. We assume that this noise will be small if compared to the linear behaviour of the system.

The data representing the forward displacement and the numerically calculated forward speed for one arbitrary run of a robot is shown in figure 6.9.

Then we use a least squares algorithm in order to identify the system. A figure with the predicted (identified) system and actual data is shown in figure 6.10.

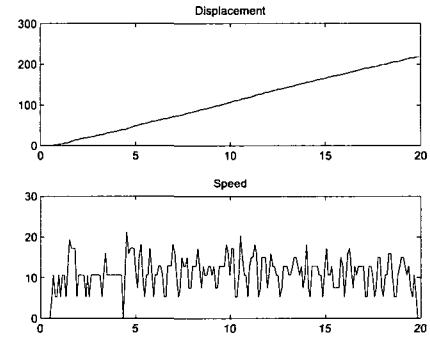

Figure 6.9: Displacement for one arbitrary run of a robot

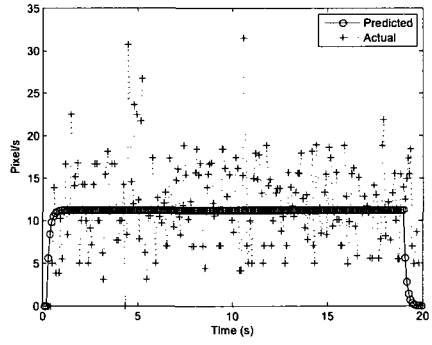

Figure 6.10: Identified system and actual collected data

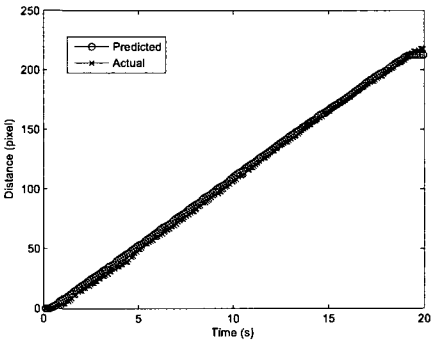

Figure 6.11: Simulation of the identified system

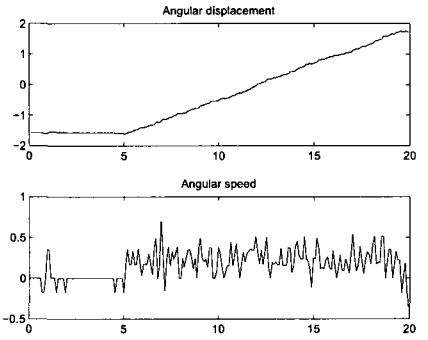

Figure 6.12: Angular displacement for one arbitrary run of a robot

We then simulate the identified system in order to compare its result with the actual data collected. Numerically calculating (6.36), and comparing it to the actual values read in the experiment, we have the results shown in figure 6.11.

The same steps have to be taken for the identification of (6.37). Let us take as an example a robot turning to the left. The data representing the angle over time is shown in figure 6.12.

After taking the derivative of the data and identifying the system with a least squares algorithm, we numerically integrate (6.37) (equations (6.17) and (6.18) use

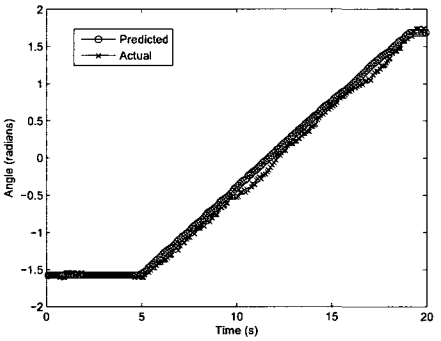

Figure 6.13: Simulation of the identified system

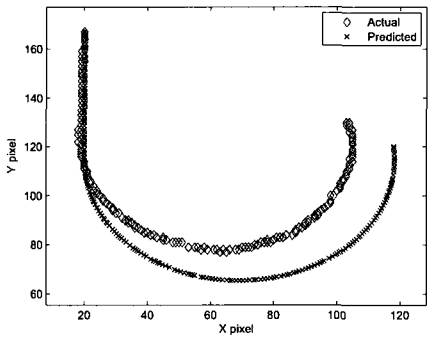

Figure 6.14: Simulation of the identified system

the turning rate and not the angular speed) and get the data shown in figure 6.13. One may notice that the agreement is very good.

After doing the identification, one may numerically integrate equations (6.17) and (6.18) and end up with the simulation shown in figure 6.14.

If we apply a signal for the robot to turn to the other side, we can predict its position as shown in figure 6.15.

Observe that in all the graphs, we use only the predicted value and do not use actual data to update the estimation. This means that if $y[k]$ is the actual data and

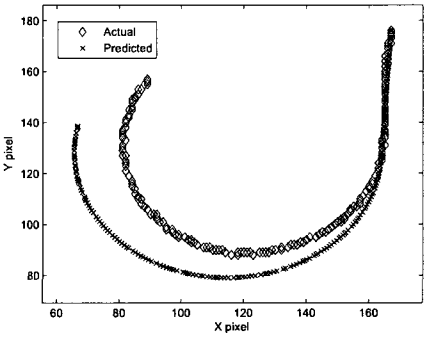

Figure 6.15: Prediction of the position of the robot when turning right

Table 6.1: Difference equations for the pursuer

Variable	**Difference equation**
Angular speed	$\omega_e[k+1] = 0.3624\omega_e[k] + 0.1569\delta_p[k]$
Forward speed	$v_e[k+1] = 0.5380v_e[k] + 9.1477u[k]$

$\hat{y}[k]$ its prediction, figures 6.11, 6.13 and 6.14 use the following relationship

$$\hat{y}[k+1] = a\hat{y}[k] + bu[k] \tag{6.38}$$

and not

$$\hat{y}[k+1] = ay[k] + bu[k] \tag{6.39}$$

As such, the difference seen in figures 6.14 and 6.15 represent the integration of a very small modeling error.

Therefore, the agreement is indeed very good and the equations found may be used to simulate the system to a very good degree of accuracy.

Table 6.1 summarizes the difference equations found for the pursuer robot using the procedure discussed in this section. Table 6.2 summarizes the same information

Table 6.2: Difference equations for the evader

Variable	Difference equation
Angular speed	$\omega_e[k+1] = 0.2243\omega_e[k] + 0.1775\delta_e[k]$
Forward speed	$v_e[k+1] = 0.5156v_e[k] + 5.4471u[k]$

for the evader robot. In both cases the sampling period is 0.09 s. In the next section we are going to simulate the system found in order to train a fuzzy controller that will drive the pursuer robot to catch the evader robot.

6.6 Simulation

In this section we are going to assess how the structure presented in sections 6.2 and 6.3 may be used to solve the problem described in section 6.4. For this evaluation, we suppose that the evader plays the optimal control to get to the target disregarding the position of the pursuer. In fact, this may be shown to be the optimal solution of the game given the constraints on the initial conditions of the game to be described. Since just one player adapts its strategy while the other plays its optimal strategy, it is expected that eventually the one who is learning will improve its response for capture time in (6.27).

Let us start by assigning values to the parameters of the model presented in section 6.4 according to the identification done in section 6.5. Calculating the steady state value for the speed of the difference equations presented in Table 6.1 and Table 6.2 for a unit step input (i.e., $u[k] = 1$), we find that V_e and V_p in (6.18) and (6.17) are $V_e = 11.235$ and $V_p = 19.800$. In the same way, finding the steady state values for ω_e and ω_p for unit step inputs ($\delta_p[k] = \delta_e[k] = 1$), we find that $\frac{V_e}{R_e} = 0.2285$ and

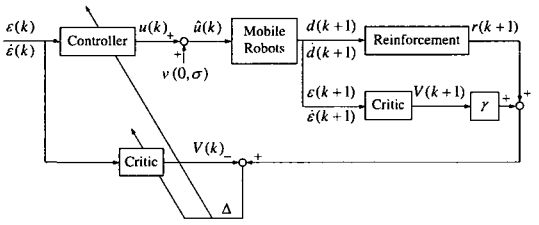

Figure 6.16: Architecture of the pursuer-evader model system

$\frac{V_p}{R_p} = 0.1969$. One may calculate then that $R_e = 49.17$ and $R_p = 100.56$. Therefore, the pursuer is almost twice as fast as the evader; and the radius of turning of the pursuer is twice that of the evader. Moreover, we remind the reader that the control signal for the pursuer and the evader are bounded and dependent on some internal dynamics; however, the pursuer is faster than the evader, who, on the other hand, can turn sharper than the pursuer. Also, the pursuer knows the orientation of the evader by checking its evolution over time. In other words, it may very well be noisy. The evader, on the other hand, has perfect information on the states of the pursuer. Of course, this last statement is only true for the simulation and not for the experiments.

Let us now redefine the architecture defined in figure 6.1 in order to make it particular to the problem of section 6.4. The new structure is shown in figure 6.16. First let us consider the reinforcement function $r(\cdot)$. This is the function that returns how well the control actuated so that the game came closer (or farther) to a solution. Since the objective of the game for the pursuer is to minimize the time to capture and capture is related to distance, it is clear that the reinforcement should be related to the variation of distance. Let us then define a function distance

$$D(\bar{x}') = \sqrt{x'^2 + y'^2} \tag{6.40}$$

and based on this function, let us define a variation of the distance

$$\Delta D(\bar{x}') \approx D(\bar{x}'(k+1)) - D(\bar{x}'(k)) \tag{6.41}$$

Observe that we can define, based on (6.41), an approximation of the derivative of the distance when the step size Δt is small (which is our case) such that

$$\dot{D}(\bar{x}'(k+1)) = \frac{\Delta D(\bar{x}'(k+1))}{\Delta t} \tag{6.42}$$

Therefore, $\Delta D(\bar{x}')$ and $\dot{D}(\bar{x}')$ are used in order to find the solution of equation (6.23). $\Delta D(\bar{x}')$ is related to function $q(\cdot)$ in (6.24), while $\dot{D}(\bar{x}')$ relates to function $g(\cdot)$ in (6.25). Therefore, if the approach speed is big, then the capture time tends to decrease.

The *reinforcement* is then the output of a fuzzy system with only one rule [18]

$$r(k+1) = \min[\mu_{small}(\Delta D(\bar{x}'(k+1))), \mu_{big}(\dot{D}(\bar{x}'(k+1)))] \tag{6.43}$$

As previously, the membership functions for these two cases are triangular. An example of such functions is depicted in figure 6.17. The center of the derivative of the distance is the maximum approach speed, calculated as the difference $V_e - V_p$ (introduced in section 6.4).

The reinforcement signal in (6.43) means that if the pursuer is getting closer to the evader at the maximum possible speed, the reinforcement is improved. Notice that if the pursuer is approaching the evader at V_{max} in figure 6.17, the reinforcement signal is determined only by the distance. Also, notice that the reinforcement is

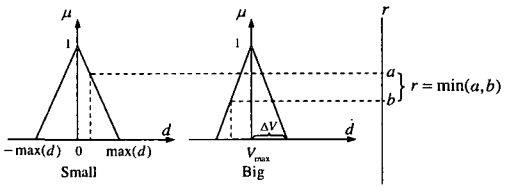

Figure 6.17: Sets for calculation of the reinforcement signal

not simply an error signal as in [24] and [18]. The distance and its derivative alone cannot represent error. Also, usually one requires errors to go to zero and in the case presented in (6.43), this is not what we want, since we require the approach speed to be the maximum. Furthermore, the pursuer does not have a "desired" path to follow, just a "desired" behaviour (catching the evader) and it is very difficult to define error based on behaviours [42].

The input variables for the controller are the angle difference $\epsilon = \theta - \psi$ (both angles defined in figure 6.3) and its derivative, i.e., $\dot{\epsilon}$. In order to define the fuzzy controller, ϵ and $\dot{\epsilon}$ are set to be our *fuzzy variables.* Notice that this defines a "PD-like" type of control. Each one of the fuzzy variables has five fuzzy sets labeled: negative big (NB), negative small (NS), zero (ZE), positive small (PS) and positive big (PB). The fuzzy sets for each of these fuzzy variables are depicted in figure 6.18.

The game takes place in a rectangle with the same dimensions as the camera range. The rectangle has its bottom-left point at $(11, 25)$ and its top-right point at $(193, 202)$. The target is assumed to be at the position $(0, 0)$. This set up information is represented in figure 6.4.

The critic has the same inputs as the controller. The output is the predicted reinforcement for the game that the critic seeks to approximate. In order to avoid

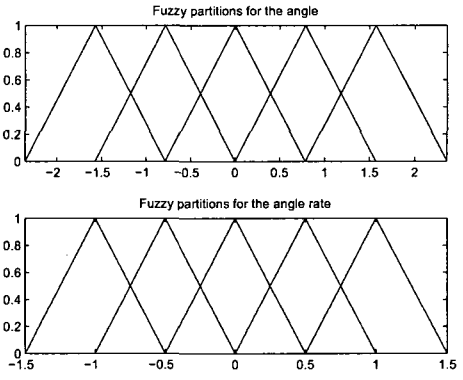

Figure 6.18: Membership functions for the antecedent of the fuzzy rules

Table 6.3: Initial values for the control signal

	angle rate				
angle	NB	NS	ZE	PS	PB
NB	1.5000	1.0000	0.5000	0.2500	0.2500
NS	1.0000	0.5000	0.2500	0.2500	0.1250
ZE	1.5000	1.0000	0	-1.0000	-1.5000
PS	-0.1250	-0.2500	-0.2500	-0.5000	-1.0000
PB	-0.2500	-0.2500	-0.5000	-1.0000	-1.5000

too much time for the controller to converge due to errors in the critic, it is advisable to perform an off line training for a previous convergence of the critic [18]. This is just an introductory learning phase and we play only 100 instances of the game for this to take place. During this phase, the learning rate α in (6.13) is set to 0.1 for a fast adaptation. Also, throughout the simulations we use γ in (6.12) as 0.5 that is a small forgetting factor, meaning the critic uses only around 20 past signals for adaptation.

The initial control surface is shown in figure 6.19. This figure represents the fuzzy

Table 6.4: Final values for the control signal

	angle rate				
angle	NB	NS	ZE	PS	PB
NB	1.3592	0.9442	0.6582	0.7440	0.2500
NS	0.9783	0.4072	2.2847	2.2105	0.1250
ZE	1.4988	0.4933	-0.3137	-0.4743	-1.4989
PS	-0.1250	-1.7160	-1.9857	-0.4134	-0.9737
PB	-0.2500	-0.7555	-0.6608	-0.9753	-1.3260

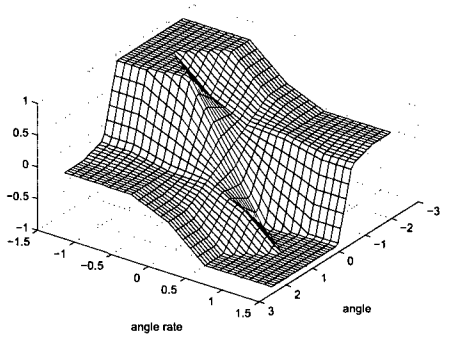

Figure 6.19: Initial control surface of the fuzzy controller

table shown in table 6.3.

We then run the adaptation law in (6.15) for 1000 instances of the game with random initial positions that satisfy the constraints of the game. Namely,

- the distance from the evader to the target is smaller than the distance from the pursuer to the target;
- the evader is supposed to be in front of the pursuer, i.e., angle θ in figure 6.3 is supposed to be in the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$;
- in the initial positions, if the optimal solution is played, the pursuer is able to

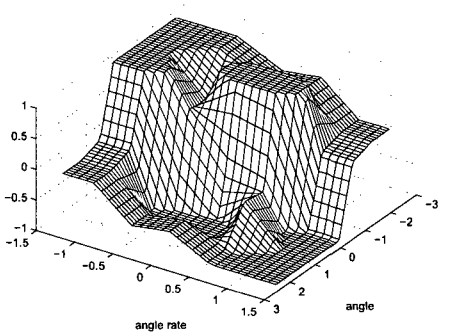

Figure 6.20: Control surface of the fuzzy controller after learning

catch the evader before it reaches the target.

The learning rate β in (6.15) is set to 0.01. At the same time, the critic continues to adapt at a learning rate of 0.1. After learning takes place, the control surface changes to the one shown in figure 6.20 that corresponds to the table 6.4.

The surface shown in figure 6.20 changes in a very important way if compared to the initial control surface in figure 6.19. First of all, the area where the control signal is on the bounds, i.e., $\delta_p = \{-1, 1\}$, is significantly increased. Since we know from (6.28) that the optimal solution is in the set $\{-1, 0, 1\}$, this is very suggestive. (Although, notice that we do not use this information in our training). Moreover, the inclination of the curve changes considerably. It is much sharper in figure 6.20. Also, the absolute value of the control signal increases or remains the same at every point of the surface, thus making the pursuer catch the evader more quickly.

Let us now see how effective the learning is. Let us define initial conditions for the evader ($x_e = 50$, $y_e = 110$ and $\theta_e = 0^o$) and for the pursuer ($x_p = 15$, $y_p = 30$ and $\theta_p = 72^o$) that were never presented during the training phase. Figure 6.21 shows that with the initial controller of table 6.3 the pursuer does not intercept the evader

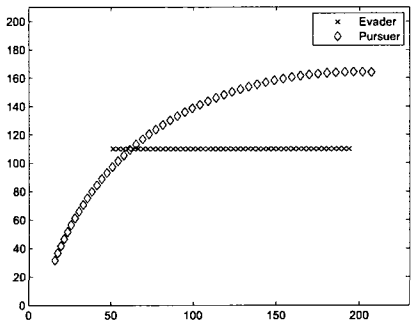

Figure 6.21: Pursuer not able to catch the evader before learning

before the evader reaches the target. In fact, the pursuer "overshoots" the evader due to its higher speed. As it is not able to turn sharply, it misses the evader. In figure 6.22, however, we show the pursuer catching the evader after learning takes place and it uses the learnt values of table 6.4. The point of interception (evader's position) is $(129, 110)$. For the matter of comparison, using the optimal controller, the interception occurs at $(127, 110)$.

In order to prove the resilience of the controller, noise was added to the measurements for the simulations depicted in figure 6.22. Another advantage of this method compared to genetic algorithms, for example, is that the convergence happens in a much faster way. Learning takes, in total, only 1100 epochs, which take only few minutes to run in a medium level computer.

The complete algorithm for the simulation discussed in this section is represented in algorithms 6.6.1 and 6.6.2. Algorithm 6.6.1 presents the initializations for the algorithm and algorithm 6.6.2 presents the adaptation of the fuzzy controller and the fuzzy critic. Notice also that algorithm 6.6.2 is executed several times until the controller and the critic converge. In the case presented in this chapter, this takes

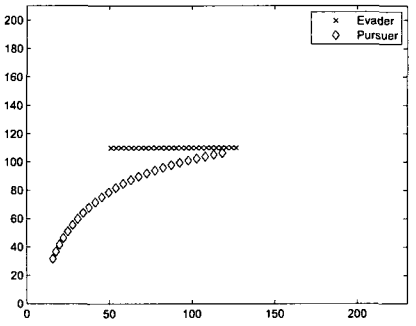

Figure 6.22: Pursuer catching the evader after learning

1000 iterations.

Algorithm 6.6.1 Initialization for the adaptation algorithm

1: Initialize the controller with the desired structure. The values for the antecedents of the rules are the ones shown in figure 6.18. The initial values for the consequent of the controller are the ones shown in table 6.3.
2: Initialize the critic with the same antecedents as the controller. As for the consequents, it is all zeros. The output of the critic is not a control signal, but the expected reinforcement.
3: Initialize the Reinforcement Block with a fuzzy system that will return the reward for some specific state. This block is initialized with the fuzzy sets in figure 6.17 and the output is as in (6.43).
4: $\gamma \leftarrow 0.5$ (equation (6.11))
5: $\alpha \leftarrow 0.1$ (equation (6.13))
6: $\beta \leftarrow 0.01$ (equation (6.15))

6.7 Experiments

Experiments are necessary in order to guarantee that the controller found in the previous section works in a real application.

As mentioned in section 6.4.2, the measurements are sent to the robots by a computer through a Bluetooth link. The packets are assembled with the positions of the robots (the evader and the pursuer) and their orientations, the position of the target and the time of the reading, as shown in figure 6.23.

In order to avoid too much noise in the calculation of the derivative $\dot{\epsilon}$, a low-pass

Algorithm 6.6.2 Adaptation algorithm

1: **while** The game does not finish **do**
2: Get the state of the system, $\bar{x}[k] = [\epsilon, \dot{\epsilon}]^T$
3: Calculate the output of the controller

$$u(k) \leftarrow \frac{\sum_{l=1}^{M}((\prod_{i=1}^{n}\mu^{F_i^l}(x_i[k]))\cdot\chi_l)}{\sum_{l=1}^{M}(\prod_{i=1}^{n}\mu^{F_i^l}(x_i[k]))}$$

4: Calculate the output of the critic

$$\hat{V}(k) \leftarrow \frac{\sum_{l=1}^{M}((\prod_{i=1}^{n}\mu^{F_i^l}(x_i[k]))\cdot\zeta_l)}{\sum_{l=1}^{M}(\prod_{i=1}^{n}\mu^{F_i^l}(x_i[k]))}$$

5: Run the game for the current time step
6: Get the states for the reinforcement, $\bar{y}[k+1] = [d, \dot{d}]^T$
7: Calculate the reinforcement signal, $r(k+1) \leftarrow \min[\mu_{small}(\Delta D(k+1)), \mu_{big}(\dot{D}(k+1))]$
8: Get the new states of the game, $\bar{x}[k+1] = [\epsilon, \dot{\epsilon}]^T$
9: Calculate the output of the critic with the new states

$$\hat{V}(k+1) \leftarrow \frac{\sum_{l=1}^{M}((\prod_{i=1}^{n}\mu^{F_i^l}(x_i[k+1]))\cdot\zeta_l)}{\sum_{l=1}^{M}(\prod_{i=1}^{n}\mu^{F_i^l}(x_i[k+1]))}$$

10: Calculate the delta signal

$$\Delta \leftarrow [r(k+1) + \gamma\hat{V}(k+1)] - \hat{V}(k)$$

11: Calculate the critic gradient

$$\frac{\partial\hat{V}(k)}{\partial\zeta_j} \leftarrow \frac{\prod_{i=1}^{n}\mu^{F_i^j}(x_i[k])}{\sum_{l=1}^{M}(\prod_{i=1}^{n}\mu^{F_i^l}(x_i[k]))}$$

12: Train the critic

$$\zeta_j(k+1) \leftarrow \zeta_j(k) + \alpha\Delta\frac{\partial\hat{V}(k)}{\partial\zeta_j}$$

13: Calculate the controller gradient

$$\frac{\partial u(k)}{\partial\chi_j} \leftarrow \frac{\prod_{i=1}^{n}\mu^{F_i^j}(x_i[k])}{\sum_{l=1}^{M}(\prod_{i=1}^{n}\mu^{F_i^l}(x_i[k]))}$$

14: Train the controller

$$\chi_j(k+1) \leftarrow \chi_j(k) + \beta\Delta[\frac{u'(k)-u(k)}{\sigma}]\frac{\partial u(k)}{\partial\chi_j}$$

15: **end while**

...	x_e	y_e	θ_e	x_p	y_p	θ_p	x_T	y_T	t	...

Figure 6.23: Packet sent to the robots

filter was implemented for the camera. Since the time step is not constant, the noise is amplified and outliers may become common, making it more difficult for the fuzzy controller to be efficient. The filter implemented is a simple discrete running average low-pass filter.

We also mentioned in section 6.4.2 that our robots are driven by motors controlled by a Motorola 68HC11 microprocessor built in a Handyboard. The message handling for both robots and the fuzzy controller for the pursuer were implemented in the microprocessor using C. Care had to be taken in order to guarantee that the calculations were made in an efficient way. The signals to the motors must be calculated quickly in order to avoid nonlinearities in the robots' behaviours.

The initial positions of the robots are the same as the initial positions for the simulations shown in Figs. 6.21 and 6.22. This is done for the comparison with the simulation to be meaningful. The real positions of the robots are shown in figure 6.24.

With the controller of table 6.4 implemented in the microprocessor of the pursuer and the same control strategy as the one presented in the simulation implemented for the evader robot, the experiment yielded the behaviour shown in figure 6.26. Figure 6.25 shows the simulation considering the real size of the robots. In other words, capture occurs when the pursuer and the evader robots physically touch.

As it may be seen from a comparison of figure 6.25 and figure 6.26, the experiment

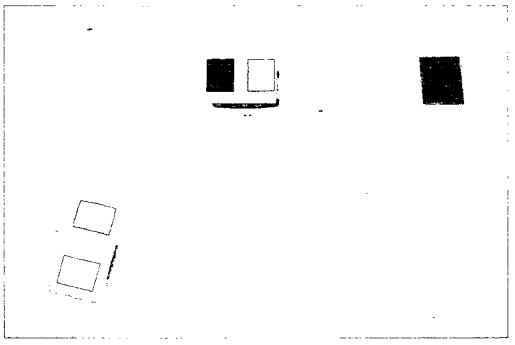

Figure 6.24: Initial conditions for the experiment

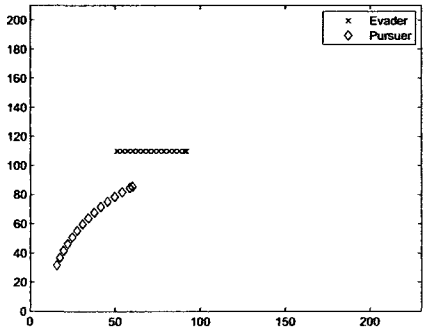

Figure 6.25: Simulation results

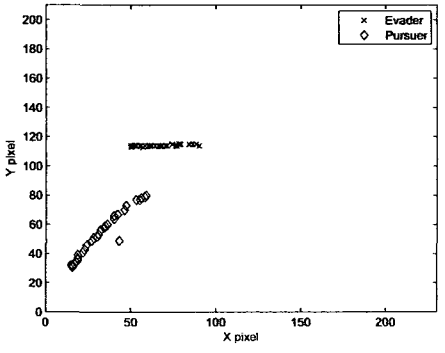

Figure 6.26: Experimental results

and the simulation agree very well. This shows that the identification and derivation of the controller were successful. Moreover, the learning was efficient in finding the best controller for an actual application.

Several other instances of the game were executed and the results invariably met the ones reached in the simulation. This shows that the controller found is suitable for the experimental set up.

One last remark must be stated regarding the experiment. The learning engine was turned off during the experiment. The reason for that is the limited range of our sensors. The games finish too soon for a reasonable learning to take place. In order to take advantage of the learning, we would have to have a larger range for the game. We are working on this and in the future we intend to have a fully experimental environment where no simulation is required for the learning to take place.

6.8 Conclusion

In this chapter we showed a method for learning in differential games. A fuzzy controller adapted by reinforcement learning is presented. An architecture derived from [24] and [18] is shown to be suitable for learning in a continuous environment.

A simulation of a modified version of the *game of two cars* is described. We suppose that only one of the players (the pursuer) adapts its behaviour. The evader is supposed to play its optimal strategy. Results show that the pursuer learns to catch the evader in an effective way. The controller adapted using the scheme described in section 6.3 is superior to the initial controller. Also, we show that the controller is resilient to noise.

We then implemented the derived control system in actual robots. Results show

that the quality of the response is similar to the simulation, illustrating that our approach is reasonable. Moreover, adaptation may continue in the robots in order for them to deal with changes in their dynamics.

In the next chapter, we will expand on the ideas introduced in this chapter and build a multiple robot pursuer game based on cooperation.

Chapter 7

Decentralized Decision Making in Games

7.1 Introduction

So far, we have been discussing how learning takes place in multiple player environments. In the previous chapter, we discussed how learning may take place in a pursuit-evasion game involving just one pursuer and one evader. We want now to extend our analysis to a game where multiple pursuers and evaders are possible.

In order to do this, we are going to switch our modeling from differential games to Markov games. The reason we are going to do this is that, when dealing with multiple pursuers and evaders, the modeling itself and the solution for the games in the differential game representation become very cumbersome [96].

In *Markov games* [103], the models of the environment and the interaction among players are considered to be Markov chains [32]. Players are able to influence the state transitions in a Markov chain by taking actions [49]. As before, we are interested to study if there is some equilibrium points related to the model, especially if there is any *Nash* equilibrium and the correspondent *value* of the game [78].

One of the most interesting questions in the field is on how the players would

adapt and learn to play strategies (or execute different actions). Again, this is the widely known problem of *learning* or adaptation in games.

In this chapter, we propose that the multiple pursuer-evader game (wherein several pursuers and/or evaders are present) is modeled by a Markov game. Moreover, each decision-making unit (or player) is adaptive. They are known as learning automata. It is going to be shown that using such an algorithm, the game converges to an equilibrium point.

The chapter is divided as follows. We start by defining the multiple-pursuer one evader game in terms of a Markov game in section 7.2. This section will introduce the notation used throughout the work (subsection 7.2.1), the representation of the states of the system (subsection 7.2.2), the transition probabilities (subsection 7.2.3), the policies available to each player (subsection 7.2.4), the rewards each player gets for each time step (subsection 7.2.5) and, finally, the general model for Markov pursuit-evasion games with the formalization of the problem to be addressed in the next sections (subsection 7.2.6). The main contribution of section 7.2 is in presenting a simple and powerful way to represent pursuit-evasion games. In section 7.3 we are going to detail the learning algorithm that will be used in the solution of the game. Section 7.4 states the objective of the learning process, namely, achieving an optimal solution for the problem described in section 7.2. Section 7.5 discusses the convergence of the approach used in three different levels: the convergence of the learning procedure to an Ordinary Differential Equation (section 7.5.2), the convergence of the algorithm at each state to a Nash equilibrium (section 7.5.3) and finally the convergence of the cost function of the game to the optimal solution (section 7.5.4). In section 7.6, simulations of the system are presented in order to show the feasibility of

the solution. Finally, section 7.7 presents our conclusions from theory and simulations and points to future work that will be done in the field.

7.2 Markov multiple-pursuers multiple-evaders games

In [50], the authors consider a Markov pursuit-evasion game between a team of n_p pursuers and a single evader. Since the team of pursuers is supposed to play as a single entity, the result is, in fact, a two-player game.

In this section, we are going to propose a model that is a variation of this game that allows multiple pursuers and multiple evaders. A somewhat similar model was proposed in [107] and [112]. However, in this section we are going to generalize the modelling in those works.

In this section we are going to follow to some extent the model presented in [50]. However, some noticeable differences must be emphasized. First of all, in their work, [50] consider a game between only two players, wherein one player, the pursuers, is in fact a *team of players*. Therefore, either some sort of sophisticated exchange of information among the pursuers or a central intelligence that controls all of them must be assumed. In our approach, we are going to assume that each player, pursuers and evaders, behave as separate entities. Moreover, we deal in here with a game with "rewards" (see subsection 7.2.5), whereas [50] deal with *probabilities*. Their cost function is in fact an expression of the probabilities of the pursuers to catch the evader at the next time step. However, this optimization is only achieved if we assume *nested information*, i.e., if one player, in this case the evader, has access to all the information available to the others plus information about its own state that is available just to itself.

Before we go any further, let us first define some needed notation. The notation follows closely the ones provided by [50] and [49].

7.2.1 Notation

Throughout this work we use low cap bold face letters (or greek symbols) for random variables, therefore, $\mathbf{x}(t)$ is going to denote a random variable. Also, given a random variable $\xi \in \mathbb{R}^n$ and some $c \in \mathbb{R}^n$, we denote the probability of the event happening as $\mathbb{P}(\xi = c)$. The same notation is used to represent conditional probabilities. The expected value of ξ conditioned to an event A is denoted by $\mathbb{E}[\xi|A]$.

The symbols $\wedge$ and $\vee$ signify the logical operators *and* and *or*, respectively. The symbols $\cap$ and $\cup$ signify the set intersection and union, respectively. If A is a set, $|A|$ represents its cardinality.

When $x \in \mathbb{R}^n$, we denote $\|x\| \in \mathbb{R}$ as the euclidian norm of x. Also, for any $x \in \mathbb{R}$, we define [103] $x^+ = \max\{x, 0\}$ and $x^- = \min\{x, 0\}$. Therefore, the absolute value is $|x| = x^+ - x^-$.

Finally, we are going to arbitrarily use the masculine pronoun "he" to refer to a pursuer, while using the feminine pronoun "she" for an evader.

7.2.2 State Representation

We start by considering a grid with a finite set of cells $\mathcal{X} = \{1, 2, \cdots, n_c\}$ and a quantization in time $\mathcal{T} = \{1, 2, \cdots\}$.

Also consider that there are two groups of players. One group is known as the *pursuers* and the other as the *evaders*. The set of pursuers is $U = \{U_1, \cdots, U_{n_p}\}$ and the set of evaders is $D = \{D_1, \cdots, D_{n_e}\}$. Therefore, there are n_p pursuers and n_e evaders. One may also define the set of all players as $N = U \cup D$ or $N =$

$\{U_1, \cdots, U_{n_p}, D_1, \cdots, D_{n_e}\}$. It is clear that $|N| = n_p + n_e = \iota$. Since the number of players is countable, in order to simplify our notation, we are simply going to denote a player by a positive integer number $k \in \{1, \cdots, \iota\}$.

Consider also that the game is quantized in time. The positions of the players are defined for all time steps $t \in \mathcal{T}$ for $\mathcal{T} = \{1, 2, \cdots\}$. If we denote by $\mathbf{x}(t) = (\mathbf{x}^1(t), \cdots, \mathbf{x}^\iota(t)) \in \mathcal{X}^\iota$ the positions at time t then the state of the game is only $\mathbf{s}(t) = (\mathbf{x}(t))$. Notice that we assume that the number of possible states $x \in \mathcal{X}^\iota$ of the system is finite, let us say $n_s \in \mathbb{Z}^+$. Hence, the cardinality of the set $\mathcal{X}^\iota$ of possible states of the game is $|\mathcal{X}^\iota| = n_s$.

7.2.3 Transition Probabilities

We now turn our attention towards the *transition probabilities* of the game. These probabilities govern the evolution of the game from an arbitrary state $x^t \in \mathcal{X}^\iota$ at time t to another arbitrary state $x^{t+1} \in \mathcal{X}^\iota$ at time $t + 1$. In order for the results to be described later to carry, the initial state x^0 is assumed to be independent of all the other random variables at time $t = 0$.

Each player k has a set of *allowable actions* [50]. We denote this set by $\mathcal{A}_k$ and the actual set $\mathcal{A}_k$ depends on the current state of the game.

The action is the desired next allowable cell that the player wants to reach. Therefore, we can define a vector of desired actions at time t for all the players involved in the game as

$$\alpha^t = [\alpha_1^t, \cdots, \alpha_\iota^t] \tag{7.1}$$

where each α_k^t is the action chosen by each player k at time t from his or her set of allowable actions $\mathcal{A}_k$. Observe then that $\alpha^t \in \mathcal{A}^\iota := \mathcal{A}_1 \times \cdots \times \mathcal{A}_\iota$.

The Markov game formalism [103] requires that the probability of transition from one arbitrary state to another is only a function of the action $\alpha \in \mathcal{A}^\iota$ chosen by each of the players at time t when the game is at state $x^t \in \mathcal{X}^\iota$. Hereafter we assume a stationary transition probability, i.e., $\mathbb{P}(\mathbf{x}(t+1) = x^{t+1} | \mathbf{x}(t) = x^t, \mathbf{a}(t) = \alpha) = p(x^t, \alpha, x^{t+1})$, $x^t, x^{t+1} \in \mathcal{X}^\iota$, $\alpha \in \mathcal{A}^\iota$, $t \in \mathcal{T}$. The function $p : \mathcal{X}^\iota \times \mathcal{A}^\iota \times \mathcal{X}^\iota \rightarrow [0,1]$ is the *transition probability function.*

Moreover, given the current state of the game $\mathbf{x}(t)$, we assume that the positions of all players k at the next time instant are determined independently by each action $\mathbf{a}_k(t)$. Hence, the probability the state is going to change from state x^t to state x^{t+1} due to action α may be written as

$$p(x^t, \alpha, x^{t+1}) = \prod_{k=1}^{\iota} p(x_k^t \xrightarrow{\alpha_k} x_k^{t+1}) \tag{7.2}$$

where $x_k^t \in \mathcal{X}$ is the position of the k^{th} agent *before* it moves and $x_k^{t+1} \in \mathcal{X}$ is the position of the k^{th} agent *after* it moves. Equation (7.2) means that the decision each player takes is independent of the decisions that all other players take at time $t \in \mathcal{T}$.

Furthermore, the probability that a player effectively reaches the chosen cell is ρ_k. This translates into:

$$\mathbb{P}((x_k^t) \xrightarrow{\alpha_k} x_k^{t+1}) = \begin{cases} \rho_k, & x_k^{t+1} = \alpha_k \in \mathcal{A}_k(x_k^t) \\ 1-\rho_k, & x_k^{t+1} = x_k^t \wedge \alpha_k \in \mathcal{A}_k(x_k^t) \\ 1, & x_k^{t+1} = x_k^t \wedge \alpha_k \notin \mathcal{A}_k(x_k^t) \\ 0, & otherwise \end{cases} \tag{7.3}$$

Observe that equation (7.3) applies to both pursuers and evaders. The probabilities ρ_k simulate the speed of the players. If $\rho_i = 1.0$ for player i and $\rho_j = 0.8$ for

player j, the speed $\dot{x}_i > \dot{x}_j$. Also, if two pursuers or two evaders try to move to the same cell, it is assumed that none of them can move.

7.2.4 Players Policies and Strategies

A "policy" for a player is a rule the player uses to select which action to take. The choice is based on past observations of how the game is played [34]. In this work, we consider policies that are *stochastic*. This simply means that at each time step $t \in \mathcal{T}$, each player selects an action according to some probability distribution [103].

Let us say that at time t the state of the game is $x^t \in \mathcal{X}^\iota$ and the position of player $k \in \{1, \cdots, \iota\}$ is $x_k^t \in \mathcal{X}$. A *policy* for player k is a function $\varphi_k : \mathcal{X}^\iota \times \mathcal{A}_k(x_k) \to [0, 1]$, such that [50]

$$\sum_{\alpha_k \in \mathcal{A}_k(x_k)} \varphi_k(x^t, \alpha_k) = 1, \ \forall x^t \in \mathcal{X}^\iota \tag{7.4}$$

A *stochastic strategy* π_k for the k^{th} player is a sequence $\pi_k^0, \pi_k^1, \cdots$. Since the game is *Markovian*, the strategy for the k^{th} player is going to be realized by a policy. Therefore, if we understand the superscript to be the time index for the state the game is, we can represent his or her strategy by the sequence $\pi_k = \{\varphi_k^0, \varphi_k^1, \cdots\}$. This is so because $\pi_k^t(\alpha)$ depends only on the present state. This means that $\pi_k^t(\alpha|(x^0, \cdots, x^t)) = \varphi_k(x^t, \alpha)$.

Now, let $\pi = \{\pi^0, \pi^1, \cdots\}$, where for each $t = 0, 1, \cdots$ each term π^t of the sequence π, represented by $\pi^t = \{\pi_1^t, \cdots, \pi_\iota^t\}$, is the set of strategies for all players at time t. We can then define probability measures dependent on π and state $x^t \in \mathcal{X}^\iota$ and denote it by $\mathbb{P}_{\pi,x^t}$. Therefore, $\mathbb{P}_{\pi,x^i}(E)$ denotes the probability of event E happening when strategy π is used [27] and the state is x^t. Also, if the probability measure is

dependent only on policy π_k, we are going to denote it by $\mathbb{P}_{\pi_k,x^t}$. In the same fashion, we may also define the expectation $\mathbb{E}_{\pi,x^t}[.]$. Notice also that the policies φ_k and φ_j, $j \neq k$ are conditionally independent of all random variables representing the states and actions at times smaller or equal to t.

Finally, observe that the quantity $\varphi_k(x^t, \alpha^t) \in [0, 1]$ is the probability of player $k \in \{1, \cdots, \iota\}$ choosing action α_k^t, where $\alpha_k^t \in \mathcal{A}_k$ represents the action chosen by player k at time $t \in \mathcal{T}$.

7.2.5 Rewards

Consider that at each time step t, as given above, the state of the game $\mathbf{x}(t)$ changes from the state $\mathbf{x}(t) = x^t \in \mathcal{X}^\iota$ to another state $\mathbf{x}(t+1) = x^{t+1} \in \mathcal{X}^\iota$ due to the action α^t [27]. Whenever the transition occurs, let us consider that each player k gets a "reward" associated with the transition from state x^t under the actuation of action α^t. We are going to denote this reward by $R_k(x^t, \alpha^t)$ [53], where $R_k : \mathcal{X}^\iota \times \mathcal{A}^\iota \to \mathbb{R}$.

One may observe that the reward $R_k(x^t, \alpha)$ is also a random variable [27]

$$R_k^t(\alpha) = R_k(x^t, \alpha) \tag{7.5}$$

Hence, one may also define the expected value of the random variable. If we consider the policies π, action α and state x^t, we may define

$$\mathbb{E}_{\pi,x^t}[R_k^t] = \sum_{\alpha^t \in \mathcal{A}^\iota} \mathbb{P}_{\pi,x^i}(x^t, \alpha^t) R_k(x^t, \alpha^t) \tag{7.6}$$

Observe that equation (7.6) is only the definition of the expected value for a discrete random variable. Also, the probability on the right hand side of (7.6) is

defined as

$$\mathbb{P}_{\pi,x^t}(x^t,\alpha^t) = (\prod_{k=1}^{\iota} \varphi_k(x^t,\alpha^t)) \tag{7.7}$$

and, therefore,

$$\mathbb{E}_{\pi,x^t}[R_k^t] = \sum_{\alpha\in\mathcal{A}^\iota} (\prod_{j=1}^{\iota} \varphi_j(x^t,\alpha^t)) R_k(x^t,\alpha^t) \tag{7.8}$$

We are going to investigate equations (7.6), (7.7) and (7.8) further in the next sections.

7.2.6 Pursuit-Evasion Games Model

Markov games [92] are extensions of the Markov decision process [103]. In a Markov game, actions are the joint result of multiple agents choosing an action individually [105].

Recall that the states of the game are represented by $\mathbf{x}(t) \in \mathcal{X}^\iota$. Also, recall that the actions available to each player are the allowable action sets $\mathcal{A}_k$ as defined in section 7.2.3. Considering that the game may be only in states that belong to the set $\mathcal{X}^\iota$, we may denote by $\mathcal{A}_k^t(x_k)$ the action set available to player $k \in \{1,\cdots,\iota\}$ when the game is found in state $x^t \in \mathcal{X}^\iota$. Furthermore, recall that the sets $\mathcal{A}_k^t(x_k)$ follow some rule that may be different for each k^{th} player.

Furthermore, the game is considered over when all the evaders are captured. This happens when a pursuer $i \in \{1,\cdots,n_p\}$ occupies the same cell as all the evaders $j \in \{1,\cdots,n_e\}$. Clearly, this implies that $n_p \geq n_e$. Consider $x_{p,i}$ as being the cell location of the i^{th} pursuer and $x_{e,j}$ as being the location of the j^{th} evader. Likewise, x_p is a vector with the position of all pursuers and x_e is the vector with the position of all evaders, as such if $x \in \mathcal{X}^\iota$, then $x = (x_p, x_e)$. Therefore, we formally define the set

$\mathcal{X}^{\iota}_{over} = \{(x_p, x_e) \in X : x_{e,j} = x_{p,i}, \forall j \in \{1, \cdots, n_e\}, \textit{ and for some } i \in \{1, \cdots, n_p\}\}$ as the *game-over set*. We assume that all players can detect when the game enters $\mathcal{X}_{over}$.

Hence, we may say that a pursuer's objective is to choose an action that would move him towards a faster end of the game (i.e., he will achieve a point in the game-over set $\mathcal{X}_{over}$) and an evader's objective is to choose an action that would make her more difficult for the pursuers to catch. These actions are dependent on the rewards that each player gets. Formally, let the set $N = U \cup D$, where U are the pursuers and D the evaders, be the set of players of a game. Also, let $\mathcal{A}^{\iota}$ be the set of actions available to all players, and $\mathcal{X}^{\iota}$ be the set of all states of the game. Finally, let π be the state strategy for all players, $\alpha \in \mathcal{A}^{\iota}$ be an arbitrary action of all ι players and $R(x^t, \alpha)$ a reward function of dimension ι for all players when the game is in state x^t. With this, we may define the pursuit-evasion game as follows.

Definition 7.2.1. *A Markov multiple pursuer-evader game is represented by the tuple* $\Gamma = \{N, \mathcal{X}^{\iota}, \mathcal{A}^{\iota}, \pi, R(.)\}$.

Moreover, we also assume the following.

Assumption 7.2.1. *Each player* $k \in \{1, \cdots, \iota\}$ *does not have full information about any other player* $j \in \{1, \cdots, \iota\}$ *such that* $k \neq j$. *In particular, player* k *does not know the reward function* $R_j(\cdot)$, *action space* $\mathcal{A}_j$ *and strategy* π_j.

Now that the definition of the game is formally completed, we want to focus on the existence of an algorithm that would lead the game to a solution. This is going to be done in the following sections.

7.3 The Learning approach

The problem of learning in a Markov process has had a lot of interest in the social sciences [99, 80] and in engineering [109].

The technique that we are going to use is known as "*Learning Automata*" [102]. Using these automata, we want to solve the problem described in definition (7.4.2). Actually, the problem may be seen as an online stochastic optimization [102, 83].

The automata we are going to focus on in this work is known as *Finite Action Learning Automata* (FALA). This is so because the actions available to each player at each time step, as defined by the game in section 7.2.6 is *finite*. We could use different classes of algorithms in order to solve the problem of pursuit evasion, but we are going to focus on the specific subclass of algorithms known as *linear reward-penalty* algorithms [102]. Moreover, it must be emphasized that no pre-adaptation is required and all the quantities necessary are computed online during the operation of the algorithm.

7.3.1 The Learning Algorithm

The algorithm we are going to use is a variation of the *linear reward-penalty automaton* [102], or L_{R-P}.

Let us consider an n-action automaton, i.e., an automaton that has n different actions from which it can choose. Also, assume that when an action is chosen, the automaton receives some *reward* $\beta(\alpha)$ from the environment [102].

Suppose that at time step t, the action chosen from the n possible actions $\{\alpha_{k1}, \cdots, \alpha_{kn}\}$ by player k is $\alpha_k^t = \alpha_{ki}$. Let us also consider the probabilities of choosing any one of the actions as a vector $\varphi_k^t = [\varphi_{k1}^t \cdots \varphi_{kn}^t]$ where $\sum_{j=1}^{n} \varphi_{kj}^t = 1$. Recall that this is what we previously called the *policy vector*. Then the vector φ_k^t is updated as follows [102, p. 17].

For φ_{ki}^t, the probability is updated as

$$\varphi_{ki}^{t+1}=\varphi_{ki}^t + \lambda_1\beta_k^t(\alpha^t)(1-\varphi_{ki}^t) - \lambda_2(1-\beta_k^t(\alpha^t))\varphi_{ki}^t \tag{7.9}$$

For φ_{kj}^t, $j \neq i$, the probabilities are updated as

$$\varphi_{kj}^{t+1}=\varphi_{kj}^t - \lambda_1\beta_k^t(\alpha)\varphi_{kj}^t + \lambda_2(1-\beta_k^t(\alpha))(\frac{1}{n-1} - \varphi_{kj}^t) \tag{7.10}$$

This is a type of reinforcement learning algorithm in which the action probability distributions (or policies) φ_k are updated based on stochastic methods of optimization.

One important note is that this algorithm encompasses another one known as *linear reward-inactivity* automaton, or L_{R-I}. In order to turn the L_{R-P} into the L_{R-I}, one only needs to set $\lambda_2 = 0$. Therefore, all the proceeding proofs are equally valid for the L_{R-I} algorithms. Moreover, the regular L_{R-P} algorithm has $\lambda_1 = \lambda_2$.

The payoffs $\beta_k^t(\alpha^t)$ are *stochastic.* Also, it is important to notice that they are, in general, different. Therefore, if we have two different agents k and j, the payoffs $\beta_k^t(\alpha^t)$ and $\beta_j^t(\alpha^t)$ are, in general, $\beta_k^t(\alpha^t) \neq \beta_j^t(\alpha^t)$

7.3.2 Learning Algorithm Applied to the Pursuer-Evader Game

Now, let us return to the game defined in section 7.2.6. Recall that the game is played on a grid with a finite set of cells $\mathcal{X} = \{1, 2, \cdots, n_c\}$. Moreover, recall that the game may be in any one of the finite *possible states* $x \in \mathcal{X}^\iota$.

Also recall that at each time $t \in \mathcal{T}$, each $\alpha_k^t \in \mathcal{A}_k$ is the desired position for player k (who may be a pursuer or an evader) at the next time instant. These are called the *actions* at each player's disposal. Therefore, we assume here the one-step motion for both players are constrained, since $\mathcal{A}_k \subset \mathcal{X}$. In the same way, recall that

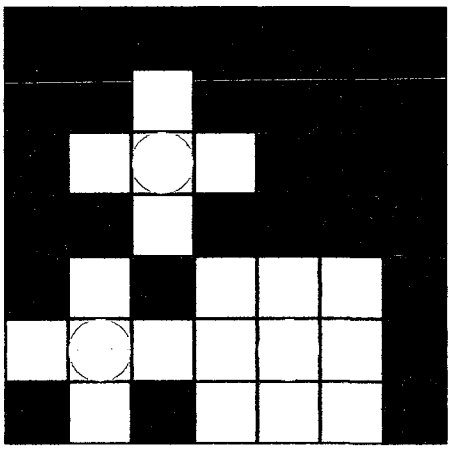

Figure 7.1: Grid of the Markov Game

$\mathcal{A}_k(x_k)$ is the set of *reachable* cells given its current position x_k and we say that these cells are *adjacent* to x_k for player k (however, notice that the cells are not necessarily contiguous to the position x_k and that, since there is a probability that the player does not move, the cell where the player is located is considered adjacent to itself). Let us discuss an example.

Let us consider that there are two pursuers ($n_p = 2$) and one evader ($n_e = 1$) in the game. Let us further assume that the cells reachable to each pursuer are those neighboring cells located at north, south, east and west. Therefore, unless the pursuer k is located at the sides of the grid, he has five possible actions at his disposal, the four neighboring cells plus to keep his own position, and $|\mathcal{A}_k(x_k(t))| = 5$. For the evaders, let us assume that they can move to north, south, east, west plus the diagonals. Therefore, unless the evader j is located at the sides of the grid, she has nine actions at her disposal, the eight neighboring cells plus to keep her own position, and $|\mathcal{A}_j(x_j(t))| = 9$. This is shown graphically in figure 7.1.

In the game described in figure 7.1, $n_s = 49 \times 49 \times 49 = 117649$ states. It is easy

to see that as we increase the number of cells or the number of players, the number of states grows very fast. This has an impact on the speed of the learning of the system, for each decision is based on the states of the game. Recall that the policies of each player (section 7.2.4) is a mapping from the state space and action space into a simplex. Therefore, in order to decide the action that is going to be taken, the players must check a table that stores the probability of each action being executed given the state in which the game is. In the beginning of the game, it is assumed that all probabilities are the same and the player behaves erratically and randomly.

When the player chooses an action, the probabilities for a given state are updated as described in equations (7.9) and (7.10). After the game is played enough times it is expected that the overall performance of the players will improve. The whole game procedure is given in algorithm 7.3.1.

In the next section, we are going to present some theorems that show that the algorithm given provides a framework for the convergence to the Nash equilibrium of the game.

7.4 Objective of Learning

7.4.1 Cost Functions

Let us consider that the pursuit game is played and that the current time step is $t \in \mathcal{T}$. The question that arises is what is the accumulated reward. The expected reward for player k is dependent on the initial state $x^0 \in \mathcal{X}^\iota$ and the strategies π that are played by each player [103]. Therefore, it may be defined as

$$v_k^t(x^0, \pi) = \mathbb{E}_{\pi, x^0}[\sum_{j=0}^{t-1} R_k^j(\alpha) + v(x^t)] \tag{7.11}$$

Algorithm 7.3.1 Adaptation algorithm

Initialize the states of the system and a table of actions to each player $k \in \{1, \cdots, \iota\}$ according to the rule $\mathcal{A}_k(x_k)$.
Initialize the learning parameters λ_1 and λ_2 to the desired values.
Place all players $k \in \{1, \cdots, \iota\}$ randomly in the play grid.
while All the evaders are not captured **do**
  Get the state of the system $\mathbf{x}^t = [x_1^t, \cdots, x_\iota^t]$
  for All evaders j **do**
    Calculate her next action $\alpha_j^t \in \mathcal{A}_j(x_j^t)$.
    if Probability of moving according to equation (7.3) is satisfied **then**
      $x_j^{t+1} = \alpha_j^t$
    end if
  end for
  for All pursuers i **do**
    Calculate his next action $\alpha_i^t \in \mathcal{A}_i(x_i^t)$.
    if Probability of moving according to equation (7.3) is satisfied **then**
      $x_i^{t+1} = \alpha_i^t$
    end if
  end for
  for All players $k \in \{1, \cdots, \iota\}$ **do**
    Calculate payoff $R_k^t(\alpha^t)$
    Learn according to equations (7.9) and (7.10).
  end for
end while

The term $v(x^t)$ on the right hand side of equation (7.11) is the *terminal payoff* of the game and it is defined as

$$v(x^t) = \begin{cases} v_{over}, if\ x^t \in \mathcal{X}_{over} \\ 0, \quad otherwise \end{cases} \tag{7.12}$$

Moreover, in order for the right hand side of equation (7.11) to be properly defined, we must have [103]

$$\mathbb{E}_{\pi,x^0}[\sum_{j=0}^{t-1} \|R_k^j(\alpha)\|] < \infty\ ,\ \forall x^0 \in \mathcal{X}^\iota \tag{7.13}$$

and

$$\mathbb{E}_{\pi,x^0}[\|v(x^t)\|] < \infty\ ,\ \forall x^0 \in \mathcal{X}^\iota \tag{7.14}$$

Since there may be different reward functions for different players, equation (7.11) implies that, in general, it is impossible to find an optimal policy for all agents playing as a group [105]. In other words, individual optimization does not imply group optimization [109]. The problem of finding some kind of individual optimization that would lead to group optimization is a design problem and we will discuss it further when we present our simulations.

Let us define a function $f : \mathcal{X}^\iota \times \mathcal{A}^\iota \rightarrow \mathcal{X}^\iota$ as a *Markov mapping* from a state to another depending on an action α. Therefore, equation (7.11) could be written iteratively as [8]

$$v_k^0 = v_0 \tag{7.15}$$

$$v_k^{t+1}(x^t, \pi) = \mathbb{E}_{\pi,x^i}[R_k^t(\alpha) + v_k^t(f(x^t, \alpha), \pi)] \tag{7.16}$$

where $x^{t+1} = f(x^t, \alpha)$ is a mapping from the current state x^t to x^{t+1}.

Since equations (7.11) and (7.16) represent the same quantity, in the next subsections we are going to state the objective of the game interchangeably using any one of these equations.

7.4.2 Optimal Solution

Before we go any further, we must define what an optimal (or nearly optimal) solution for the problem means. This is referred as the *Nash equilibrium* of a multi player game. Recall that $\iota = n_p + n_e$, then the following definition may be established [89].

Definition 7.4.1. *The array of strategies* $\pi^* = (\pi_1^*, \cdots, \pi_\iota^*)$ *is called a* **Nash equilibrium** *if for each* k, $1 \leq k \leq \iota$, $\forall \pi \in [0,1]^{\mathcal{A}}$, *we have*

$$v_k^t(x^0, (\pi_1^*, \cdots, \pi_\iota^*)) \geq v_k^t(x^0, (\pi_1^*, \cdots, \pi_{k-1}^*, \pi_k, \pi_{k+1}^*, \cdots, \pi_\iota^*)) \tag{7.17}$$

Definition 7.4.1 means that the optimal solution for the game is the maximization of the cost functions of each individual player. Furthermore, this definition refers to the *strategies* used by each player. However, recall that since the game is Markovian, the strategies are realized, at each time instant $t \in \mathcal{T}$, by a *policy* as described in section 7.2.4. We are going to discuss the relationship between strategies and policies further when we discuss the convergence of the learning algorithm.

7.4.3 Statement of Objective

The objective of the game is to find the optimal (or suboptimal) solution for each player in a decentralized fashion. The format of the reward may vary from player to player, but they are written in such a way that all the players try to maximize their gains.

Furthermore, it must also be emphasized that no prior computation must be performed in order to pre-adapt some parameters [109] and that only minimal (or no) communication is allowed among the players. Therefore, the problem may be formalized as follows.

Definition 7.4.2. *Find the policy π_k for each player $k \in \{1, \cdots, \iota\}$ such that the cost functions described in equation (7.11) are maximized (or nearly maximized) for all players k.*

The expected value in equation (7.11) is justified by the fact that the learning algorithm does not operate in the space of actions [102, p. 58], but it updates the probability distributions φ_k for the actions of all automata $k \in \{1, \cdots, \iota\}$.

7.5 Convergence

As stated before, we are interested to know if the game defined in section 7.2.6 with the learning algorithm as defined in section 7.3.2 will converge to the Nash equilibrium as discussed in section 7.4.2.

In order to show that the optimal (or suboptimal) play may be found, it is sufficient to show that the learning process takes the system to a mapping that maximizes the cost of equation (7.11). The problem could be solved by using *dynamic programming*; however, the problem is that in order to do so it is required that the players have access to all possible payoffs ahead of time and the number of combinations grows very fast [109].

Therefore, this section is divided as follows. We will start, in section 7.5.1, by presenting a sketch of the proof that will be presented in the next subsections. In this section, we will also link the notation of learning in automata systems to the notation we are using so far. Section 7.5.2 shows that an Ordinary Differential Equation can

be used in place of the difference equation used to describe the algorithm. Then, in section 7.5.3 we will state the theorems that show that the learning automata system converges to the equilibrium point as required. Finally, section 7.5.4 shows that the game converges to the desired Nash equilibrium.

7.5.1 Sketch of the Proof

In order to prove that the learning algorithm of equations (7.9) and (7.10) converges to the strategy that achieves the Nash equilibrium of definition 7.4.1, we need to prove three separate theorems. Theorem 7.5.2 deals with the possibility to represent the algorithm by an Ordinary Differential Equation (ODE). Theorem 7.5.3 deals with the convergence of the algorithm to some stable optimal policies (a Nash equilibrium). Finally, theorem 7.5.4 states the convergence of the strategies to the Nash equilibrium (definition 7.4.1).

We are going to state all theorems in this section and will prove them separately in the subsequent sections.

Notice that there are two different time steps involved in the algorithm. The first is the change of states of the game and is referred to as t. This index is used as a superscript in the state $x^t \in \mathcal{X}^\iota$, for example. The other time instant will be referred to as n and it refers to the number of times a given state is visited and is used in referencing the evolution of the policies. Therefore, $\varphi_k^n(x^t, \alpha^t)$ refers to the n^{th} visit that player k makes to state x^t. For short, we are going to use $\varphi_k^n(\alpha^n) = \varphi_k^n(x^t, \alpha^t)$ in the remaining of the article, unless it is strictly necessary to explicitly include the state x^t.

Now, let us go back to equations (7.9) and (7.10). Consider that λ_2 is set to zero, i.e., $\lambda_2 = 0$. If we also set $\lambda_1 = \lambda$, this would result in the evolution of the policies

φ_k to [102]

$$\varphi_k^{n+1} = \varphi_k^n + \lambda G(\varphi_k^n, \xi_k^n) \tag{7.18}$$

where $\xi_k^n = [\alpha_k^n, \beta_k^n]$ and the function $G(\cdot)$ is the updating given by equations (7.9) and (7.10). Notice that we used the index n even for the actions. This was done in order to simplify the notation and it must be taken as the n^{th} visit to the state x^t in which the policies $\varphi_k^n(x^t)$ are used.

For the case described above, consider that $e_{\alpha_k^n}$ as a unit vector with the same dimension of φ_k^n with the unit at the position that represents the action α_k^n. Also, let us represent $\xi_k^n = [\alpha_k^n, \beta_k^n] = [\xi_{k1}^n, \xi_{k2}^n]$, then function $G(\cdot)$ is

$$\begin{aligned} G(\varphi_k^n, \xi_k^n) &= \beta_k^n (e_{\alpha_k^n} - \varphi_k^n) \\ G(\varphi_k^n, \xi_k^n) &= \xi_{k2}^n (e_{\xi_{k1}^n} - \varphi_k^n) \end{aligned} \tag{7.19}$$

One could observe that even the equations (7.9) and (7.10) could also be described with equation (7.18). In order to do so, the only thing necessary is to set λ as a vector encompassing λ_1 and λ_2 and change the definition of $G(\cdot)$. This would not change our results very much. However, some details would become more cumbersome and we chose to simplify the approach taken.

Moreover, let us assume that the reward $\xi_{k2}^n = \beta_k^n \in [0, 1]$, i.e., it is normalized. Also, let us represent $G(\cdot) = [G_1(\cdot), \cdots, G_{r_k}(\cdot)]$ where r_k is the number of actions at player k's disposal, therefore

$$G_i(\varphi_k^n, \xi_k^n) \in [0, 1], \ \forall i = 1, \cdots, r_k \tag{7.20}$$

With these considerations, we turn our attention to the sketch of the proof that

will be implemented in the next subsections.

The first theorem (theorem 7.5.2) states that the learning algorithm in its discrete form (equation (7.24)) can be represented by the ODE (7.26). This is important, for, after this is established, we can state theorem 7.5.3 that guarantees that in any given state, the learning algorithm results in the convergence to the Nash equilibrium [102] in the policy space. Observe also that the probability $\mathbb{P}_{\pi,x^t}$ explicitly depends on the state $x^t \in \mathcal{X}^\iota$ and all the strategies π for all players. This is also true for the expected value $\mathbb{E}_{\pi,x^t}$. This is explicitly done just to remind the reader that the development done here is directly linked to the development of section 7.4. However, it must be noticed that each automata does not know the strategies followed by the other robots. But, mathematically the dependency is necessary and signals that the overall optimal value indeed depends on the actions of all players.

And finally, with the statement of theorems 7.5.2 and 7.5.3, we can prove the convergence of the game defined in section 7.2.6 to the Nash equilibrium defined in 7.4.2. In other words, the learning algorithm converges to the optimal policies independent of the initial state $x^0 \in \mathcal{X}^\iota$.

7.5.2 Analysis of the ODE

In this subsection, since it is clear that we are discussing a feature that is common to all players $k \in \{1, \cdots, \iota\}$, we are going to drop the subscript k from the equations that describe the evolution of the learning algorithm. We also introduce the notation $\varphi^{n,\lambda}$, $\xi^{n,\lambda}$, $G^\lambda(\cdot)$ etc. to emphasize that the evolution of the processes is dependent on a parameter λ that is supposed to be small.

Let us again consider the process $\{G(\varphi^n, \xi^n)\}$ in equation (7.19). Recall that $\xi^n = [\alpha^n, \beta^n]$. Now, let us assume that the process $\{\xi^n\}$ depends on the states of

the game in a Markovian way. We are going to deal with the actual form of this dependency later. But, for now, it suffices to notice that the reward β_k^n is dependent on the strategies π used and, therefore, must depend on the past actions taken so far.

Let us next define the expected value of $G(\cdot)$ as [61, p. 37]

$$\mathbb{E}[G(\varphi^n, \xi^n)|\varphi^i, \xi^i, i \leq n] = \mathcal{G}(\varphi^n, \xi^n) \tag{7.21}$$

From equation (7.20) it may also be concluded that

$$\{\mathcal{G}(\varphi^{n,\lambda}, \xi^{n,\lambda}); \lambda, n\} \ \textit{is uniformly integrable} \tag{7.22}$$

And also define

$$\begin{aligned} \theta^n &= G(\varphi^n, \xi^n) - \mathbb{E}[G(\varphi^n, \xi^n)] \\ \theta^n &= G(\varphi^n, \xi^n) - \mathcal{G}(\varphi^n, \xi^n) \end{aligned} \tag{7.23}$$

Then, we can rewrite equation (7.18) as

$$\varphi^{n+1} = \varphi^n + \lambda \mathcal{G}(\varphi^n, \xi^n) + \lambda \theta^n \tag{7.24}$$

where θ^n are martingale differences. However, the conditional mean term $\mathcal{G}(\varphi^n, \xi^n)$ is not simple [61] due to the fact that the evolution of $\{\xi^n\}$ depends on the evolution of the sequence $\{\varphi^n\}$.

Since we are interested in studying the asymptotic behaviour of $\varphi^{n,\lambda}$, it is useful to introduce a process wherein the parameter φ is kept constant. Let $\{\xi^n(\varphi)\}$ denote a Markov chain that results from $\{\xi^n\}$ when the parameter $\varphi^{n,\lambda}$ is held constant at value φ [61, p. 37]. If λ is small, then $\varphi^{n,\lambda}$ varies slowly and it may be argued that

the law of large number suggests that

$$\bar{\mathcal{G}}(\varphi) = \mathbb{E}_{\varphi}[\mathcal{G}(\varphi, \xi^n(\varphi)|\varphi] \tag{7.25}$$

where $\bar{\mathcal{G}}(\varphi)$ is a continuous function of φ. Henceforth, we want to show that the algorithm in equation (7.24) may be represented by

$$\dot{\varphi} = \bar{\mathcal{G}}(\varphi) \tag{7.26}$$

Notice that since the process $\{\varphi^{n,\lambda}\}$ is bounded, the limit set represented by equation (7.26) is also bounded. Furthermore, one should notice that we are looking for the zeros of $\bar{\mathcal{G}}$.

Finally, before we enunciate the theorem, let us state without proof another theorem [61, p. 247]. It can be found in [61, p. 247].

Theorem 7.5.1. *Assume algorithm (7.18) and the following assumptions:*

i. $\{\mathcal{G}(\varphi^{n,\lambda}, \xi^{n,\lambda}); \lambda, n\}$ *is uniformly integrable.*

ii. If we write equation (7.18) as

$$\varphi_k^{n+1} = \varphi_k^n + \lambda G(\varphi_k^n, \xi_k^n) + \tau_n^{\lambda}$$

where τ_n^{λ} *is an error term, then*

$$\lim_{m,n,\lambda} \frac{1}{m} \sum_{i=n}^{n+m-1} \mathbb{E}_n^{\lambda}[\tau_i^{\lambda}] = 0$$

where in the limit, $m \to \infty$, $n \to \infty$ *and* $\lambda \to 0$.

iii. There are measurable functions $g_n^{\lambda}(\cdot)$ *and random variables* τ_n^{λ} *such that*

$$\mathbb{E}[\mathcal{G}(\cdot)] = g_n^{\lambda}(\varphi_n^{\lambda}, \xi_n^{\lambda}) + \tau_n^{\lambda}$$

iv. For each $\delta > 0$ *there is a compact* A_δ *such that*

$$\inf_{n,\lambda} \mathbb{P}(\xi_n^{\lambda} \in A_\delta) \geq 1 - \delta$$

v. For each λ, φ, and n, there is a transition function $\mathbb{P}_n^\lambda(\cdot,\cdot|\varphi)$ such that $\mathbb{P}_n^\lambda(\cdot,A|\cdot)$ is measurable for each set A in the range space Ξ of ξ_n^λ and

$$\mathbb{P}(\xi_{n+1}^\lambda \in \cdot|\mathcal{F}_n^\lambda) = \mathbb{P}_n^\lambda(\xi_n^\lambda,\cdot|\varphi_n^\lambda)$$

vi. $\mathbb{E}|\mathbb{E}_n^\lambda[g_i^\lambda(\varphi_n^\lambda,\xi_i(\varphi_n^\lambda))] - \mathbb{E}_n^\lambda[g_i^\lambda(\varphi_i^\lambda,\xi_i^\lambda)]| \leq \rho$, for i and n such that $\mu_{\rho,\lambda} \leq i-n \leq m_{\rho,\lambda}$ and $\lambda \leq \lambda(\rho)$

vii. Given $\rho > 0$, there are $\nu > 0$ and $\lambda_0(\rho) > 0$ such that for any random variables $(\tilde{\varphi},\hat{\varphi})$ satisfying $\mathbb{P}(|\hat{\varphi}-\tilde{\varphi}| > \nu$ and $\lambda \leq \lambda_0(\rho)$, we have

$$\sup_{i,n:m_{\rho,\lambda}\geq i-n\geq\mu_{\rho,\lambda}} \mathbb{E}|\mathbb{E}_n^\lambda[g_i^\lambda(\hat{\varphi},\xi_i(\hat{\varphi}))] - \mathbb{E}_n^\lambda[g_i^\lambda(\tilde{\varphi},\xi_i(\tilde{\varphi}))]| \leq \rho$$

viii. There is a continuous function $\bar{\mathcal{G}}(\cdot)$ such that for any sequence of integers $N \to \infty$ satisfying $\lambda N \to 0$ as $\lambda \to 0$ and compact set A

$$\frac{1}{N}\sum_{i=jN}^{jN+N-1} \mathbb{E}[\mathcal{G}(\varphi,\xi^i(\varphi)) - \bar{\mathcal{G}}(\varphi)]$$

Then for almost all ω, the path $\varphi(\omega,\cdot)$ lies in an invariant set of equation (7.26).

We can now enunciate the theorem.

Theorem 7.5.2. *For any initial condition φ^0 and the algorithm given in equation (7.18), the sequences $\{\varphi^n\}$ are trajectories of equation (7.26)*

Proof. This theorem is satisfied if the conditions of theorem 8.4.4 of [61, p. 247] stated above as theorem 7.5.1 are satisfied.

The condition on the uniform integrability of $\mathcal{G}(\varphi^{n,\lambda},\xi^{n,\lambda})$ (condition (i)) is satisfied according to the discussions that led to equation (7.22). Observe also that according to equation (7.21), function $\mathcal{G}(\cdot)$ does not depend explicitly on n and λ and the noise term assumed in theorem 7.5.1 can be taken to be zero. Therefore condition (ii) is also satisfied. Notice that condition (iii) is a restatement of equation (7.21) and is also satisfied.

Condition (iv) is satisfied by the fact that the tightness of $\{\xi^{n,\lambda}\}$ is guaranteed by the definition of the rewards β^n and the boundedness of $\{G(\cdot)\}$. Also, the transition function

$$\mathbb{P}(\xi,A|\varphi) = P(\xi^{n+1,\lambda} \in A|\varphi^{n,\lambda} = \varphi, \xi^{n,\lambda} = \xi)$$

is continuous and measurable for all the range space of ξ and condition (v) is also satisfied.

Now, observe that $\mathcal{G}(\cdot)$ is bounded and continuous. Therefore, small variations on the parameters φ will not lead to large variations in the update of the algorithm as described by equation (7.24). Therefore, conditions (vi) and (vii) are readily satisfied.

Let us now, focus on the final condition and let us write the equation according to the conditions of the present problem

$$\lim_{N\to\infty} \frac{1}{N} \sum_{i=jN}^{jN+N-1} \mathbb{E}[\mathcal{G}(\varphi, \xi^i(\varphi)) - \bar{\mathcal{G}}(\varphi)]$$
$$= \lim_{N\to\infty} \frac{1}{N} \sum_{i=jN}^{jN+N-1} \mathbb{E}[\mathcal{G}(\varphi, \xi^i(\varphi)) - \mathbb{E}_{\varphi}[\mathcal{G}(\varphi, \xi^i(\varphi))]]$$

Observe that the term $\mathcal{G}(\varphi, \xi^i(\varphi)) - \mathbb{E}_{\varphi}[\mathcal{G}(\varphi, \xi^i(\varphi))]$ is only white noise with an unknown distribution. Therefore, it may be concluded that the limit goes to zero and condition $(viii)$ holds.

Therefore, the evolution of policies may be represented by equation (7.26). □

Now that we have shown that the game may be approximated by the ODE, we are ready to prove the convergence of the policies as updated by the learning automata algorithm to the Nash equilibrium.

7.5.3 Analysis of the Learning Algorithm

We now turn our attention to the problem of the convergence of the algorithm in each state to the Nash equilibrium, i.e., given that the game is at state $x^t \in \mathcal{X}^\iota$, the policies of the k^{th} player converge to a set of policies that maximize the gains that the player will have.

Recall that the Nash equilibrium in the strategy space is stated as definition 7.4.1. Also, recall that the strategies are actually realized by the use of a policy φ_{k,x^t} at each state $x^t \in \mathcal{X}^\iota$ of the game. At last, recall that we defined the reward at time

$t \in \mathcal{T}$ for a game starting at state x^0 and player k to be

$$v_k^t(x^0, \pi) = \mathbb{E}_{\pi, x^0}[\sum_{j=0}^{t-1} R_k^j(\alpha) + v(x^t)] \tag{7.27}$$

In order to simplify our discussions, let us define the *payoff function* $\beta_k(x^t, \alpha)$ as

$$\beta_k(x^t, \alpha) = R_k(x^t, \alpha) + v_k(f(x^t, \alpha), \pi) \tag{7.28}$$

From now on, when we use the set of actions $\mathcal{A}_k$ for player k we actually mean $\mathcal{A}_k^{x^t}$, i.e., the set of actions when player k is at state x^t. Again, we do this in order to simplify the notation. With this consideration, we can now define the expected payoff due to an action.

Definition 7.5.1. *Functions* $\mathcal{F}_k^{x^i} : \prod_{j=1}^{\iota} \mathcal{A}_j \to [0, 1]$ *for* $k \in \{1, \cdots, \iota\}$ *and* $x^t \in \mathcal{X}^\iota$ *are defined by*

$$\mathcal{F}_k^{x^t}(\alpha) = \mathbb{E}_{\pi_k, x^t}[\beta_k(x^t, \alpha) | \alpha \in \mathcal{A}^\iota] \tag{7.29}$$

Furthermore, the players have no knowledge of the payoff functions given by the environment, they only have access to the payoff signals (the $\beta_k(x^t, \alpha)$*).*

This definition actually encompasses some assumptions as well. The first is that the players have incomplete information on the environment and on the behaviour of the other players. This is done in order to approximate the algorithm with real life situations. Also, it may be noticed that

$$v_k^{t+1}(x^t, \pi) = \sum_{\alpha \in \mathcal{A}} (\prod_{i=1}^{\iota} \varphi_i(\alpha)) \mathcal{F}_k^{x^t}(\alpha) \tag{7.30}$$

Let us now define the vector of policies $\Phi_{x^t} \in \Psi_{x^t}$ as being the policies when the game is in state $x^t \in \mathcal{X}^\iota$

$$\Phi_{x^t} = [\varphi_1^{x^t}, \cdots, \varphi_\iota^{x^t}] \tag{7.31}$$

We now define yet another set of functions $\mathcal{V}_k^{x^t} : \Psi_{x^t} \rightarrow [0,1]$ as

$$\mathcal{V}_k^{x^t}(\Phi_{x^t}) = \mathbb{E}_{\pi_k, x^t}[\beta_k(x^t, \alpha)|\Phi_{x^t}] \tag{7.32}$$

i.e., the payoff for the k^{th} player under the set of policies Φ_{x^t}. Therefore, functions $\mathcal{F}_k^{x^t}$ are the expected payoffs due to the action chosen while $\mathcal{V}_k^{x^t}$ are the expected payoffs due to the policy profile (or probability distribution).

Now we are ready to determine what a maximal point for state $x^t \in \mathcal{X}^\iota$ is [102, p. 59]

Definition 7.5.2. *Consider the set of policies $\Phi_{x^t}^* = [\varphi_1^{x^t *}, \cdots, \varphi_\iota^{x^t *}]$. We say that Φ^* is a maximal point (and in this context a Nash equilibrium) of the game at stage $x^t \in \mathcal{X}^\iota$ if for each $k \in \{1, \cdots, \iota\}$*

$$\mathcal{V}_k^{x^t}(\Phi_{x^t}^*) \geq \mathcal{V}_k^{x^t}(\Phi_{x^t})$$

for all Φ such that $\Phi = [\varphi_1^{x^t *}, \cdots, \varphi_{k-1}^{x^t *}, \varphi_k^{x^t}, \varphi_{k+1}^{x^t *}, \cdots, \varphi_\iota^{x^t *}]$, with $\varphi_k^{x^t} \in |\mathcal{A}_k|$ is a probability vector different from $\varphi_k^{x^t *}$, i.e., $\varphi_k^{x^t} \neq \varphi_k^{x^t *}$.

In the same fashion, we may also define the maximal point for the set of actions (called, in this case, a *modal point*) as follows [102, p. 61]

Definition 7.5.3. *The the set of actions $\alpha^* = [\alpha_1^*, \cdots, \alpha_\iota^*]$ is a modal point (and in this context also a Nash equilibrium) of the game at stage $x^t \in \mathcal{X}^\iota$ if for each $k \in \{1, \cdots, \iota\}$*

$$\mathcal{F}_k^{x^t}(\alpha^*) \geq \mathcal{F}_k^{x^t}(\alpha)$$

for all $\alpha = [\alpha_1^*, \cdots, \alpha_{k-1}^*, \alpha_k, \alpha_{k+1}^*, \cdots, \alpha_\iota^*]$ such that $\alpha_k \neq \alpha_k^*$.

We also need an extra assumption before we proceed to prove the theorem.

Assumption 7.5.1. *For every x^t, k and α, $\mathcal{F}_k^{x^t}(\alpha)$ has a finite number of maxima, all in a compact set and has no maxima at infinity [102].*

This assumption is somewhat intuitive since the game should actually have a finite number of maxima due to the fact that it has a finite number of states. However, it

also assures that the payoff functions do not grow indefinitely. This assumption is in line with equation (7.13).

Using the array Φ_{x^t} defined in equation (7.31), the ODE in equation (7.26) may be rewritten as

$$\frac{d\Phi_{x^t}}{dt} = \bar{\mathcal{G}}^{x^t}(\Phi_{x^t}) \tag{7.33}$$

Equation (7.33) implies that each individual policy j for each player k is defined by

$$\frac{d\varphi_{kj}^{x^t}}{dt} = \bar{\mathcal{G}}_{kj}^{x^t}(\Phi_{x^t}) \tag{7.34}$$

for all players $k \in \{1, \cdots, \iota\}$ and individual policy $1 \leq j \leq |\mathcal{A}_k|$ at each state x^t.

Let us now define α_k^j as the unit vector with unity in position $1 \leq j \leq |\mathcal{A}_k|$, i.e., the k^{th} player chooses action j. Now, we can define the following function

$$f_{kj}^{x^t}(\Phi_{x^t}) = \mathbb{E}_{\pi,x^t}[\beta_k^t | \varphi_l^{x^t}, k \neq l, \alpha_k^j] \tag{7.35}$$

Function $f_{kj}^{x^t}(\Phi_{x^t})$ is the expected payoff function given that player k chooses action j and all other players $l \neq k$ play their expected mixed strategy defined by vectors φ_l. Observe that if we define the set $\Upsilon \subset \mathcal{A}^\iota$ as the set of all actions with the k^{th} player always choosing action j but all other players choosing every possible combination, we get

$$f_{kj}^{x^t}(\Phi_{x^t}) = \sum_{\alpha \in \Upsilon} \left(\prod_{i \neq k} \varphi_i(\alpha) \right) \mathcal{F}_k^{x^t}(\alpha) \tag{7.36}$$

One could also write the function $\mathcal{V}_k^{x^t}(\Phi_{x^t})$ in terms of functions $f_{kj}(\Phi_{x^t})$

$$\mathcal{V}_k^{x^t}(\Phi_{x^t}) = \sum_{l=1}^{|\mathcal{A}_k|} f_{kl}^{x^t}(\Phi_{x^t}) \varphi_{kl}^{x^t} \tag{7.37}$$

Now, recall that

$$\begin{aligned}\bar{\mathcal{G}}_k^{x^t}(\Phi_{x^t}) &= \mathbb{E}_{\pi,x^t}[\beta_k^t(e_{\alpha_k} - \varphi_k)|\varphi_k = \varphi_k^{x^t}] \\ &= \sum_{q=1}^{r} \mathbb{E}_{\pi,x^t}[\beta_k^t(e_q - \varphi_k)|\varphi_k = \varphi_k^{x^t}, \alpha_k^q]\varphi_{kq}^{x^t} \\ &= \sum_{q=1}^{r} \mathbb{E}_{\pi,x^t}[\beta_k^t|\varphi_k = \varphi_k^{x^t}, \alpha_k^q](e_q - \varphi_k^{x^t})\varphi_{kq}^{x^t} \end{aligned} \tag{7.38}$$

where r is the number of actions at the k^{th} player's disposal.

If we now substitute equation (7.35) into equation (7.38), we get

$$\bar{\mathcal{G}}_k^{x^t}(\Phi_{x^t}) = \sum_{q=1}^{r} f_{kq}^{x^t}(\Phi_{x^t})(e_q - \varphi_k^{x^t})\varphi_{kq}^{x^t} \tag{7.39}$$

However, we are interested in the terms $\bar{\mathcal{G}}_{kj}^{x^t}(\Phi_{x^t})$, i.e., we are interested in the j^{th} component of $\bar{\mathcal{G}}_k^{x^t}(\Phi_{x^t})$. Therefore, the term $\bar{\mathcal{G}}_{kj}^{x^t}(\Phi_{x^t})$ is a scalar and we can write the j^{th} component of equation (7.39) as

$$\begin{aligned}\bar{\mathcal{G}}_{kj}^{x^t}(\Phi_{x^t}) &= \varphi_{kj}^{x^t}(1 - \varphi_{kj}^{x^t})f_{kj}^{x^t}(\Phi_{x^t}) - \sum_{q\neq j} \varphi_{kq}^{x^t}\varphi_{kj}^{x^t}f_{kq}^{x^t}(\Phi_{x^t}) \\ &= \varphi_{kj}^{x^t}f_{kj}^{x^t}(\Phi_{x^t}) - (\varphi_{kj}^{x^t})^2 f_{kj}^{x^t}(\Phi_{x^t}) - \sum_{q\neq j} \varphi_{kq}^{x^t}\varphi_{kj}^{x^t}f_{kq}^{x^t}(\Phi_{x^t}) \end{aligned} \tag{7.40}$$

Now we can enunciate theorem 7.5.3.

Theorem 7.5.3. *With the updating rule of equation (7.24), the algorithm presents the following characteristics.*

i. All strict Nash equilibria in pure policies are asymptotically stable.

ii. Any equilibrium point which is not a Nash equilibrium in pure or mixed strategies is unstable.

Proof. We now consider the first part of the theorem.

i. Let $\Phi_{x^t}^* = [\mathbf{e}_{\alpha_1}^*, \cdots, \mathbf{e}_{\alpha_\iota}^*]$ be a pure maximal point. Now, define a region $\Phi_{x^t}^\rho = \{\Phi_{x^t} \in \mathcal{A}^\iota | \Phi_{x^t} = [\varphi_1^{x^t}, \cdots, \varphi_\iota^{x^t}]\}$ such that each $\varphi_k^{x^t}$, $k \in \{1, \cdots, \iota\}$, is a probability vector of dimension $|\mathcal{A}_k|$ close to the vector $\mathbf{e}_{\alpha_k}^*$ by a distance of $\epsilon > 0$.

Now let $\Phi_{x^t} \in \Psi_{x^t}^{\rho}$ such that

$$\Phi_{x^t} = [\mathbf{e}^*_{\alpha_1}, \cdots, \mathbf{e}^*_{\alpha_{k-1}}, \varphi_k, \mathbf{e}^*_{\alpha_{k+1}}, \cdots, \mathbf{e}^*_{\alpha_\iota}]$$

Then, substituting equation (7.40) into equation (7.34), we would have

$$\frac{d\varphi_{kl}^{x^t}}{dt} = \varphi_{kj}^{x^t} f_{kj}^{x^t}(\Phi_{x^t}) - (\varphi_{kj}^{x^t})^2 f_{kj}^{x^t}(\Phi_{x^t}) - \sum_{q \neq j} \varphi_{kq}^{x^t} \varphi_{kj}^{x^t} f_{kq}^{x^t}(\Phi_{x^t}) \tag{7.41}$$

Now, recall that the terms $f_{ki}^{x^t}$ as well as all the policy components φ_{ki} are in the interval $[0, 1]$. Hence, the two last terms in the RHS of equation (7.41) are negative. Let us then analyze the term $\varphi_{kj}^{x^t} f_{kj}^{x^t}(\Phi_{x^t})$. Since it is supposed that Φ_{x^t} is close to $\Phi^*_{x^t}$, a Taylor expansion around $\Phi^*_{x^t}$ is valid. Then, using a Taylor expansion of (7.36), we end up with

$$\varphi_{kj}^{x^t} f_{kj}^{x^t}(\Phi_{x^t}) = \varphi_{kj}^{x^t}[\mathcal{F}_k^{x^t}(\alpha) - \mathcal{F}_k^{x^t}(\alpha^*)] + \ high - order \ terms$$

where $\alpha = [\alpha_1^*, \cdots, \alpha_{k-1}^*, \alpha_k, \alpha_{k+1}^*, \cdots, \alpha_\iota^*]$, i.e., $\alpha \neq \alpha^*$ just by its k^{th} element. By definition 7.5.3 it is clear that

$$\varphi_{kj}^{x^t} f_{kj}^{x^t}(\Phi_{x^t}) < 0, \ \ \forall \alpha \neq \alpha^*$$

and $\dot{\varphi}_{kl}^{x^t} < 0$.

We now consider a Lyapunov function defined over $\Phi_{x^t}^{\rho}$ as

$$V_k(\Phi_{x^t}) = \sum_k \sum_{l \neq \alpha_k} \varphi_{kl}^{x^t} \tag{7.42}$$

Note that we are dealing with pure policies as such $\sum_{l \neq \alpha_k} \varphi_{kl}^{x^t} = 0$ and then $V_k(\Phi^*_{x^t}) = 0$. Also, since $\Phi^*_{x^t}$ is supposed to be a strict maximal point, for every other $\Phi_{x^t} \neq \Phi^*_{x^t}$, $V_k(\Phi_{x^t}) > 0$. Let us now take the derivative of $V_k(\cdot)$ with respect to time

$$\begin{aligned} \dot{V}_k(\Phi_{x^t}) = & \sum_k \sum_{l \neq \alpha_k} \dot{\varphi}_{kl}^{x^t} \\ < & 0, \ \ \forall l \neq \alpha_k^* \end{aligned}$$

Therefore, $\Phi^*_{x^t}$ is asymptotically stable.

ii. Let Φ^* be an equilibrium point of the game Γ. Since it is supposed not to be a Nash equilibrium, it is not a maximal point. Therefore, by definition 7.5.2, we have

$$\mathcal{V}_k^{x^t}(\Phi_{x^t}^*) < \mathcal{V}_k^{x^t}(\Phi_{x^t})$$

for some Φ_{x^t} in the neighbourhood of $\Phi_{x^t}^*$. This implies that [6] there is a $f_{kj}^{x^t}(\Phi_{x^t}^*)$ such that

$$f_{kj}^{x^t}(\Phi_{x^t}^*) > \mathcal{V}_k^{x^t}(\Phi_{x^t}^*) \tag{7.43}$$

Now, recall that

$$\begin{aligned}
\bar{\mathcal{G}}_{kj}^{x^t}(\Phi_{x^t}) &= \varphi_{kj}^{x^t}(1-\varphi_{kj}^{x^t}) f_{kj}^{x^t}(\Phi_{x^t}) - \sum_{q\neq j} \varphi_{kq}^{x^t}\varphi_{kj}^{x^t} f_{kq}^{x^t}(\Phi_{x^t}) \\
&= \varphi_{kj}^{x^t} f_{kj}^{x^t}(\Phi_{x^t}) - (\varphi_{kj}^{x^t})^2 f_{kj}^{x^t}(\Phi_{x^t}) - \sum_{q\neq j} \varphi_{kq}^{x^t}\varphi_{kj}^{x^t} f_{kq}^{x^t}(\Phi_{x^t}) \\
&= \varphi_{kj}^{x^t} f_{kj}^{x^t}(\Phi_{x^t}) - \sum_{q\neq j} \varphi_{kq}^{x^t}\varphi_{kj}^{x^t} f_{kq}^{x^t}(\Phi_{x^t}) - \varphi_{kj}^{x^t}\varphi_{kj}^{x^t} f_{kj}^{x^t}(\Phi_{x^t}) \\
&= \varphi_{kj}^{x^t} f_{kj}^{x^t}(\Phi_{x^t}) - \sum_{q} \varphi_{kq}^{x^t}\varphi_{kj}^{x^t} f_{kq}^{x^t}(\Phi_{x^t}) \\
\bar{\mathcal{G}}_{kj}^{x^t}(\Phi_{x^t}) &= \varphi_{kj}^{x^t}[f_{kj}^{x^t}(\Phi_{x^t}) - \sum_{q} \varphi_{kq}^{x^t} f_{kq}^{x^t}(\Phi_{x^t})]
\end{aligned}$$

Using equation (7.37), we get

$$\bar{\mathcal{G}}_{kl}^{x^t}(\Phi_{x^t}) = \varphi_{kl}^{x^t}[f_{kl}^{x^t}(\Phi_{x^t}) - \mathcal{V}_k^{x^t}(\Phi_{x^t})]$$

By equation (7.43)we have that

$$\bar{\mathcal{G}}_{kl}^{x^t}(\Phi_{x^t}) > 0$$

This means that

$$\frac{d\varphi_{kl}^{x^t}}{dt}(\Phi_{x^t}) > 0$$

Therefore, in small neighbourhoods of $\Phi_{x^t}^*$ the policies will diverge and the policy set $\Phi_{x^t}^*$ is unstable.

This completes the proof of the theorem. □

Remark 1. *Notice that the application of the algorithm results in an optimization of*

the cost function of equation (7.11) for the present state, i.e., as if all previous states were omitted and there was only the transition to state t to $t+1$. Explicitly, it results, for each player $k \in \{1, \cdots, \iota\}$, in

$$v_k^{t+1}(x^t, \pi) = \max_{\Phi_{x^t} \in \Psi_{x^t}} \mathbb{E}_{\pi, x^i}[R_k^t(\alpha) + v_k^t(f(x^t, \alpha), \pi)] \tag{7.44}$$

This maximization is exactly the equation that is used in dynamic programming [8] (Bellman's equation). However, we do not go over the whole set of possible transitions in order to find the optimal policy. The maximization is achieved iteratively through the learning procedure of algorithm 7.3.1.

7.5.4 Analysis of the Game

Finally, in order to prove that the game with the learning algorithm described so far will converge to the Nash equilibrium in the strategy space, we need to prove that the maximization in the policy space as shown in theorem 7.5.3 will lead to the optimal solution for the Markov game. Before we proceed to theorem 7.5.4, let us enunciate the following lemma.

Lemma 7.5.1. *Consider that algorithm 7.3.1 converges and let $v_k^{t*}(x^0, \pi^*)$ be the optimal value for the cost function*

$$v_k^t(x^0, \pi) = \mathbb{E}_{\pi, x^0}[\sum_{j=0}^{t-1} R_k^j(\alpha) + v(x^t)] \tag{7.45}$$

at time $t \in \mathcal{T}$ for player k. Then, the maximization defined by equation (7.44) leads to the optimal value $v_k^{t*}(x^0, \pi^*)$ and, moreover, the strategy $\pi_k^* = \{\varphi_k^{x^0*}, \cdots, \varphi_k^{x^{t-1}*}\}$ is optimal.

Proof. Recall that the maximization derived in theorem 7.5.3 is explicitly written as equation (7.44). Also notice that since the game we are considering is Markovian, the

optimal value $v_k^{t*}(x^0, \pi^*)$ for equation (7.45) is

$$\begin{aligned}
v_k^{t*}(x^0,\pi^*) &= \max_{\varphi_k^{x^0},\cdots,\varphi_k^{x^{t-1}}} \{\mathbb{E}_{x^0}[R_k(x^0\alpha^0) + \mathbb{E}_{x^1}[R_k(x^1,\alpha^1) + \cdots \\
&\quad + \mathbb{E}_{x^{t-1}}[R_k(x^{t-1},\alpha^{t-1}) + v(x^t)]\cdots]]\} \\
v_k^{t*}(x^0,\pi^*) &= \max_{\varphi_k^{x^0}}\{\mathbb{E}_{x^0}[R_k(x^0,\alpha^0) + \max_{\varphi_k^{x^1}}\{\mathbb{E}_{x^1}[R_k(x^1,\alpha^1) + \cdots \\
&\quad + \max_{\varphi_k^{x^{t-1}}}\{\mathbb{E}_{x^{t-1}}[R_k(x^{t-1},\alpha^{t-1}) + v(x^t)]\}\cdots]\}]\}
\end{aligned} \tag{7.46}$$

By using equation (7.44) in equation (7.46) sequentially, we get

$$\begin{aligned}
v_k^{t*}(x^0,\pi^*) &= \max_{\varphi_k^{x^0}}\{\mathbb{E}_{x^0}[R_k(x^0,\alpha^0) + \max_{\varphi_k^{x^1}}\{\mathbb{E}_{x^1}[R_k(x^1,\alpha^1) + \cdots \\
&\quad + v_k^t(x^t,\varphi_k\pi)]\}\cdots]\} \\
v_k^{t*}(x^0,\pi^*) &= \max_{\varphi_k^{x^0},\cdots,\varphi_k^{x^{t-1}}} \{\mathbb{E}_{x^0}[R_k(x^0\alpha^0) + v_k^t(x^1,\pi)]\} \\
v_k^{t*}(x^0,\pi^*) &= v_k^t(x^0,\pi)
\end{aligned}$$

Since the right hand side and the left hand side of the equation above are equal, it shows that the use of equation (7.44) at each step of the optimization process indeed leads to the optimal solution for the problem. Also, the equation makes it explicit that the strategy $\pi_k^* = \{\varphi_k^{x^{0*}}, \cdots, \varphi_k^{x^{t-1*}}\}$ maximizes equation (7.44) at every time step related to each state $x^i \in \mathcal{X}^\iota$, $i = 0, \cdots, t$. □

With the proof of lemma 7.5.1, we can finally introduce theorem 7.5.4. Before we do so, notice that we assume that at time t the game reaches one of the final states, i.e., that $x^t \in \mathcal{X}_{over}$.

Theorem 7.5.4. *With the updating rule of equation (7.24) applied to all possible states $x^t \in \mathcal{X}^\iota$, the game defined in definition 7.2.1 converges to the optimal strategy profile as defined in definition 7.4.1.*

Proof. Lemma 7.5.1 shows that the maximization of equation (7.44) found in section 7.5.3 converges to the optimal solution. Now, we only need to prove that this optimal solution is the Nash equilibrium in the space of strategies.

Assume that all players $j \neq k$ play their optimal strategies, but player k plays something different, say $\tilde{\pi}_k$. Also, without loss of generality, assume that only the policy at time step $t-1$ is different from the optimal, i.e.,

$$\tilde{\pi}_k = \{\pi_k^{1*}, \cdots, \pi_k^{t-2*}, \tilde{\pi}_k^{t-1}\}$$

and we call this set of strategies $\tilde{\pi} = \{\pi_1^*, \cdots, \pi_{k-1}^*, \tilde{\pi}_k, \pi_{k+1}^*, \cdots, \pi_\iota^*\}$

Hence, we could write the cost function $v_k^t(\cdot)$ as

$$\begin{aligned} v_k^t(x^0, \tilde{\pi}) = & \{\mathbb{E}_{x^0}[R_k(x^0\alpha^0) + \mathbb{E}_{x^1}[R_k(x^1, \alpha^1) + \cdots \\ & + \mathbb{E}_{x^{t-1}}[R_k(x^{t-1}, \alpha^{t-1}) + v(x^t)|\tilde{\pi}_k^{t-1}] \cdots |\pi_k^{1^*}]|\pi_k^{0^*}]\} \\ v_k^t(x^0, \tilde{\pi}) = & \max_{\varphi_k^{x^{0^*}}} \{\mathbb{E}_{x^0}[R_k(x^0, \alpha^0) + \max_{\varphi_k^{x^{1^*}}} \{\mathbb{E}_{x^1}[R_k(x^1, \alpha^1) + \cdots \\ & + \mathbb{E}_{x^{t-1}}[R_k(x^{t-1}, \alpha^{t-1}) + v(x^t)|\tilde{\pi}_k^{t-1}]]\}] \end{aligned}$$

Since only the policy for state $x^{t-1}$is supposed to be different of the optimal, we may conclude that the reward received will not be larger than the one received if the optimal policy according to lemma 7.5.1 was used. Therefore,

$$v_k^t(x^0, \tilde{\pi}) \leq v_k^t(x^0, \pi^*)$$

□

Remark 2. *Notice that theorem 7.5.4 does not claim that the Nash equilibrium is unique.*

Remark 3. *The expected value* $\mathbb{E}_{x^t,\pi}[R_k(x^t, \alpha^t) + v_k^{t-1}(f(x^t, \alpha), \pi)]$ *is calculated by the environment and the players do not need to have access to it.*

7.5.5 Conclusions of the proofs

The proofs of the three theorems of this section (theorem 7.5.2, theorem 7.5.3 and theorem 7.5.4) allow us several conclusions.

The first conclusion is that even when the payoffs are not supposed to be independent, the algorithm presented will converge to a Nash equilibrium. This is discussed in theorem 7.5.2. In [102], all the cases discussed assume that the payoffs are independent of each other. In our problem, however, the payoffs depend on the past decisions and values of the policy parameters φ such that independence cannot be assumed.

The second conclusion is that the stable maximal points are those with pure strategies. This is in fact a characteristic of the Finite Action Learning Automaton (FALA)

[102] that is made explicit by theorem 7.5.3. One should notice that depending on the problem under study, mixed policy Nash equilibria could also arise. However, for the pursuit-evasion game studied in this chapter, intuitively these mixed policies do not arise [102]. The general proof of this characteristic is still an open question.

The third conclusion is that given enough time the algorithm will lead to Nash equilibria in the strategy space. This is a result similar to the one reached by Dynamic Programming. However, it must be emphasized again that the process to get to the solution is very different in the approach used in this chapter, since the maximization is achieved iteratively through the learning procedure of algorithm 7.3.1 and not through the investigation of all possible transitions.

We now show a simulation that illustrates the use of the algorithm discussed so far.

7.6 Simulation

In order to show that players learn when using the algorithm discussed in this chapter, we have only the pursuers learning using algorithm 7.3.1.

We are now going to discuss a simulation that shows the feasibility of the suggested approach. In it, we consider a game played in a 7×7 grid. It must be noticed that the number of states for this simulation is much larger and we will not provide a comparison to the optimal calculated directly from the equations, since this is prohibitive.

For the simulation, we place in the grid 2 pursuers and 1 evader. The set of pursuers is defined by $U = \{1, 2\}$ and the set of evaders by $D = \{3\}$. The pursuers have as the *reachable* cells the neighboring cells located at north, south, east and west

as depicted in figure 7.1. The evader can move to north, south, east, west plus the diagonals as also depicted in figure 7.1.

The evader plays some fixed strategy defined as getting as far from the pursuers as possible. In fact, the evader tries to occupy cells that cannot be reached by the pursuers at the next time step. In doing so, the evader actually postpones the capture time by at least one time step.

Furthermore, the probability that the pursuers will move, ρ_1 and ρ_2, is only 0.8, meaning that 20% of the time the players do not change position when commanded to do so. However, the evader moves all the time, i.e., $\rho_3 = 1.0$. These numbers simulate the speed of the players. Therefore, the pursuers are substantially slower than the evader. Also, notice that the evader is more maneuverable than the pursuers, for she can move to more cells than the pursuers. Hence, in order to capture the evader the pursuers must work together, otherwise the evader can always avoid capture.

The reward function $R_k(x^t, \alpha^t)$ for the pursuers is only a normalized version of the *Manhattan distance* from the pursuer to the evader. The Manhattan distance is the distance between the players measured along axes at right angles. Suppose that the pursuer is located at position $x^p = (x_p, y_p)$ and the evader is located at position $x_e = (x_e, y_e)$. The Manhattan distance between both is simply $|x_p - x_e| + |y_p - y_e|$.

The expected value $v_k^t(\cdot)$ for each state $x^t \in \mathcal{X}^\iota$ is calculated numerically. Since the policies are evolving and we want the expected value given to the use of the current policy, we need to use some type of *forgetting factor* in order to guarantee the convergence of the game. We use a forgetting factor of 0.95 meaning that the expected value is calculated based on the past 20 visits to that state.

In order to establish a baseline for comparison, we run the game 100 times and

collect the time required for the pursuers to capture the evader. The mean time for capture and the standard deviation are depicted in table 7.1.

Table 7.1: Mean and standard deviation of capture before training.

Mean	Standard Deviation
12,990.11	14,064.27

We then train the pursuers for 1000 games. We collect the times for capture and in intervals of 10 games calculate the average time for capture. The data is plot in figure 7.2

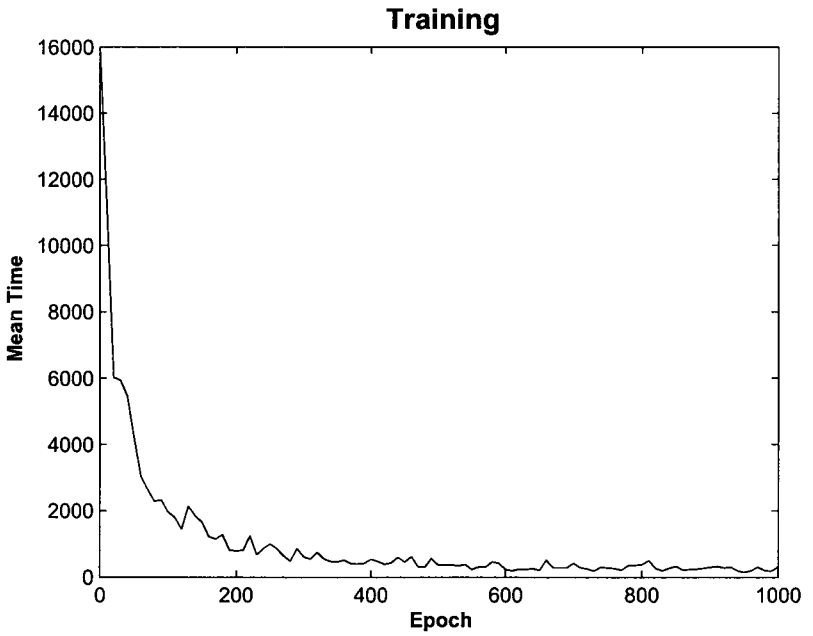

Figure 7.2: Performance of the training

In fact, in order to guarantee that the game converges to the optimal solution, the number of iterations (the number of different games played) should be even greater

due to the fact that we would need to visit all the states of the game a considerable number of times so the expected values could be evaluated and converge to its real value. Therefore, the game converges to the optimal solution given the current expected values, but it would take longer for the optimal solution to be found. Regardless, one may observe that the average time for capture decreases sharply with training.

In order to assess how well the training is, we run the game for more 100 time and recalculate the mean time and standard deviation for capture. The results are show in table 7.2. The decrease shows that the pursuers actually "learn" to capture the evader. Although the choice of rewards take into account only the benefit for the player himself this actually leads him to cooperate with the other pursuer. In summary, the pursuers learn to work together towards an objective because this is the most profitable action they could individually take.

Table 7.2: Mean and standard deviation of capture after training.

Mean	**Standard Deviation**
249.0000	214.0179

One should also observe how large the standard deviations in tables 7.1 and 7.2 are in comparison with the mean values. The reason for this is the stochastic nature of the game, especially the choices of $\rho_1 = \rho_2 = 0.8$. If we choose this number to be 1.0 the standard deviation decreases substantially. However, we think that leaving it at 0.8 brings some good features to the system, since it forces the pursuers to work together besides simulating delays and noise in their commands.

7.7 Conclusion

In this chapter we have shown how a pursuit-evasion game may be modeled with Markov chains. We have also shown that using a learning automata will lead to the optimal solution of the finite time/finite state game with incomplete information.

Simulations were provided in order to show the feasibility of the approach. It is shown that the learning phase leads to a considerable improvement in the response of the game when compared to the initial configuration of policies. Moreover, it is shown that players can learn to cooperate even when they optimize only their own cost function. However, it must be emphasized that the optimization achieved is not the optimal group optimization. This would only be accomplished if there is a global cost function to be optimized and if full information is assumed, which is not the case. The cooperation emerges only because this is also more profitable to the individual robots.

One drawback of the technique is that convergence to the optimal can only be proved if the players have full representation of all possible states of the system. As either the number of players or the size of the grid increases, the calculations would become too cumbersome. In the future, we intend to use a fuzzy representation of the environment in order to reduce the complexity of the game. However, this would make it impossible to prove a theoretical convergence to the optimal policy.

Chapter 8

Conclusion

In this thesis we have demonstrated that using different techniques coupled with different areas of Game Theory we can model, simulate and implement robots that are able to solve several problems.

We have introduced and extended the concept of *personality traits* and showed how it can be used to promote cooperation among robots. The three simulations discussed also present how powerful the concept is. It was shown that when the game is purely competitive the algorithm derived will converge to the Nash equilibrium. Also the problems of robots leaving a room and the search and rescue task demonstrate that the algorithm may be used to achieve cooperation in very difficult environments.

Then we have shown that the problem of reaching a stable swarm formation can be modeled as a differential game. Also, the solution of this game has been achieved and a algorithm based on classical optimization techniques was presented. The solution is shown to be smooth and the robots reach a final constant position.

Then we turn our attention to the problem of learning in differential games in general and in the problem of pursuit-evasion in particular. We present a control structure with fuzzy controllers and fuzzy critics that measure the performance of

the controllers. Then, an algorithm to adapt the controllers and critics based on reinforcement learning is also presented. We show that, in the problem of one pursuer and one evader, such structure and algorithm converges to the optimal solution of the game.

Finally, we broaden the problem to encompass several pursuers and evaders. In order to do so we use *Markov games*. We present an algorithm and show that it converges to the Nash equilibrium. The proof is designed in three stages. In the first, we show that the algorithm may be represented as an ordinary differential equation (ODE). In the second stage we show that the ODE will lead the policies for each state of the game to its Nash equilibrium. And, finally, in the third stage, we show that as the policies converge to the Nash equilibrium, the strategies also converge to the Nash equilibrium in the strategy space.

In conclusion, we may say that all the modeling tools and techniques used in this thesis open new venues that may be explored in order to find more and better solutions for the problem of relationships among robots.

References

[1] M. Andrecut and M. K. Ali. Fuzzy reinforcement learning. *International Journal of Modern Physics C*, 13(5):659–674, 2002.

[2] G. Arslan and J. S. Shamma. Anticipatory learning in general evolutionary games. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 6289–6294, 2006.

[3] R. J. Aumann and J. H. Dreze. Cooperative games in coalition structures. *International Journal of Game Theory*, 3:217–237, 1974.

[4] R. Axelrod. *The evolution of cooperation.* Basic Books, New York, 1984.

[5] T. Balch and R. C. Arkin. Behavior-Based Formation Control for Multirobot Teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, 1998.

[6] T. Basar and G. J. Olsder. *Dynamic Noncooperative Game Theory.* SIAM, Philadelphia, 2 edition, 1999.

[7] G. Beni. From swarm intelligence to swarm robotics. In E. Sahin and W. M. Spears, editors, *Swarm Robotics: SAB 2004 international workshop, Santa Monica, CA, USA, July 17, 2004 : revised selected papers*, pages 1–9, Berlin, Heidelberg, 2005. Springer-Verlag.

[8] D. P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, Belmont, MA, 1995.

[9] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, New York, New York, 1999.

[10] E. Bonabeau, G. Theraulaz, E. Arpin, and E. Sardet. The Building Behavior of Lattice Swarms. In *Artificial Life IV*, pages 307–312, Cambridge, MA, 1994. MIT Press.

[11] E. Bonabeau, G. Theraulaz, J. L. Deneubourg, S. Aron, and S. Camazine. Self-organization in social insects. *Trends in Ecol. Evol.*, 12:188–193, 1997.

[12] J. Borenstein and Y. Koren. Real time obstable avoidance for fast mobile robots. *IEEE Trans. on Systems, Man and Cybernetics*, 19(5):1179–1187, 1989.

[13] B. Brandstatter and U. Baumgartner. Particle Swarm Optimization - Mass-Spring System Analogon. *IEEE Transactions on Magnetics*, 38(2):997–1000, 2002.

[14] J. V. Breakwell and A. W. MERZ. Football as a differential game. *Journal of Guidance, Control, and Dynamics*, 15(5):1292–1294, 1992.

[15] C. Breder. Equations descriptive of fish schools and other animal aggregations. *Ecology*, 35(3):361–370, 1954.

[16] G. W. Brown. Iterative solutions of games by fictitious play. In T. C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 374–376. Wiley, New York, New York, 1951.

[17] A. E. Bryson and Y. Ho, editors. *Applied Optimal Control: Optimization, Estimation, and Control.* Taylor & Francis, Levittown, PA, 1975. Rev. printing.

[18] W. M. Buijtenen, G. Schram, and R. Babuska an H. B. Verbruggen. Adaptive fuzzy control of satellite attitude by reinforcement learning. *IEEE Transactions on Fuzzy Systems*, 6(2):185–194, 1998.

[19] Y. U. Cao, A. S. Fukunaga, and A. B. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4:7–27, 1997.

[20] R. Chalmers, D. Scheidt, T. Neighoff, S. Witwicki, and R. Bamberger. Cooperating unmanned vehicles. In *AIAA 1st Intelligent Systems Technical Conference*, 2004.

[21] A. M. Colman. *Game theory and its applications in the social and biological sciences.* Butterworth-Heinemann, Boston, Massachusetts, 1995.

[22] A. Czirok, E. Ben-Jacob, I. Cohen, and T. Vicsek. Formation of complex bacterial colonies via self-generated vortices. *Phys. Rev. E*, 54(2):1791–1801, 1996.

[23] A. Czirok and T. Vicsek. Collective behaviour of interacting self-propelled particles. *Physica A*, 281:17–29, 2000.

[24] X. Dai, C. Li, and A. B. Rad. An approach to tune fuzzy controllers based on reinforcement learning for autonomous vehicle control. *IEEE Transactions on Intelligent Transportation Systems*, 6(3):285–293, 2005.

[25] C. Darwin. *The expression of the emotions in man and animals.* University of Chicago Press, Chicago, Illinois, 1965.

[26] R. Dawkins. *The selfish gene.* Oxford University Press, New York, new ed. edition, 1989.

[27] C. Derman. *Finite State Markovian Decision Processes.* Academic Press, New York, New York, 1970.

[28] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Trans. Syst. Man Cybern. B*, 26:29–41, 1996.

[29] M. Dorigo, V. Trianni, E. Sahin, and et. al. Evolving self-organizing behaviours for a swarm-bot. *Autonomous Robots*, 17:223–245, 2004.

[30] T. Driessen. *Cooperative games, solutions and applications.* Kluwer Academic Publishers, Boston, MA, 1988.

[31] M. Dorigo et. al. The swarm-bots project. In E. Sahin and W. M. Spears, editors, *Swarm Robotics: SAB 2004 international workshop, Santa Monica, CA, USA, July 17, 2004 : revised selected papers*, pages 1–9, Berlin, Heidelberg, 2005. Springer-Verlag.

[32] J. Filar and K. Vrieze. *Competitive Markov Decision Processes.* Springer-Verlag, New York, New York, 1997.

[33] M. H. Foley and W. E. Schmitendorf. A Class of Differential Games with Two Pursuers Versus One Evader. *IEEE Transactions on Automatic Control*, pages 239–243, 1974.

[34] D. Fudenberg and D. K. Levine. *The Theory of Learning in Games.* The MIT Press, Cambridge, Massachusetts, 1998.

[35] V. Gazi. Stability Aggregations Using Potentials and Sliding-Mode Control. *IEEE Transactions on Robotics*, 21(6):1208–1214, 2005.

[36] V. Gazi and K. Passino. Stability Analysis of Swarms. *IEEE Transactions on Automatic Control*, 48(4):692–697, 2003.

[37] V. Gazi and K. Passino. Stability Analysis of Social Foraging Swarms. *IEEE Transactions on Systems, Man, and Cybernetics*, 34(1):539–557, 2004.

[38] J. Ge, L. Tang, J. Reimann, and G. Vachtsevanos. Hierarchical decomposition approach for pursuit-evasion differential game with multiple players. In *IEEE Aerospace Conference*, page 7pp., 2006.

[39] J. Ge, L. Tang, J. Reimann, and G. Vachtsevanos. Suboptimal approaches to multiplayer pursuit-evasion differential games. In *AIAA Guidance, Navigation and Control Conference*, pages 5272–5278, 2006.

[40] F. Giulietti, L. Pollini, and M. Innocenti. Autonomous Formation Flight. *IEEE Control Systems Magazine*, 20:34–44, 2000.

[41] S. Givigi and H. M. Schwartz. Evolutionary swarm intelligence applied to robotics. In *Prooceedings of the IEEE International Conference on Mechatronics and Automation*, pages 1005–1010, 2005.

[42] S. N. Givigi and H. M. Schwartz. Swarm robot systems based on the evolution of personality traits. *Turkish Journal of Electrical Engineering & Computer Sciences (Elektrik) : Special Issue on Swarm Robotics*, 15(2):257–282, 2007.

[43] S. N. Givigi and H. M. Schwartz. Swarm formation viewed as a differential game. In *Proceedings of the 10th IASTED International Conference on Control and Applications*, 2008.

[44] P. P. Grassé. La Reconstruction du nid et les Coordinations Inter-Individuelles chez Bellicositermes Natalensis et Cubetermes sp. La théorie de la Stigmergie: Essai d'interprétation du Comportement des Termites Constructeurs. *Insect. Soc.*, 6:41–80, 1959.

[45] M. E. Harmon and L. C. Baird III. Residual advantage learning applied to a differential game. In *Proceedings of the International Conference on Neural Networks*, pages 1–6, 1996.

[46] M. E. Harmon, L. C. Baird III, and A. H. Klopf. Reinforcement learning applied to a differential game. *Adaptive Behavior*, 4(1):3–28, 1995.

[47] S. Hart. Adaptive heuristics. *Econometrica*, 73(5):1401–1430, 2005.

[48] S. P. H. Heap and Y. Varoufakis. *Game theory: a critical introduction.* Routledge, London; New York, 1995.

[49] J. P. Hespanha and M. Prandini. Nash equilibria in partial-information games on markov chains. In *40th IEEE Conference on Decision and Control*, volume 3, pages 2102–2107, 2001.

[50] J. P. Hespanha, M. Prandini, and S. Sastry. Probabilistic pursuit-evasion games: A one-step nash approach. In *39th IEEE Conference on Decision and Control*, pages 2272–2277, 2000.

[51] Y. C. Ho, A. E. Bryson, and S. Baron. Differential games and optimal pursuit-evasion strategies. *IEEE Transactions on Automatic Control*, 10(4):385–389, 1965.

[52] J. Hofbauer and G. Sorger. A differential game approach to evolutionary equilibrium selection. *International Game Theory Review*, 4(1):17–31, 2002.

[53] R. A. Howard. *Dynamic Programming and Markov Processes.* M. I. T. Press, Cambridge, MA, 1960.

[54] D. G. Hull. *Optimal Control Theory for Aplications.* Springer, Inc., New York, New York, 2003.

[55] R. Isaacs. *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization.* John Wiley and Sons, Inc., New York, New York, 1965.

[56] H. Ishibuchi, R. Sakamoto, and T. Nakashima. Learning fuzzy rules from iterative execution of games. *Fuzzy Sets and Systems*, 135:213–240, 2003.

[57] S. N. Givigi Jr. and H. M. Schwartz. A game theoretic approach to swarm robotics. *Applied Bionics and Biomechanics*, 3:1–12, 2006.

[58] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[59] J. P. Kahan and A. Rapoport. *Theories of Coalition Formation.* Lawrence Earlbaum Associates, Publishers, Hillsdale, NJ; London, 1984.

[60] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks,*, pages 1942–1948, 1995.

[61] H. J. Kushner and G. G. Yin. *Stochastic Approximation Algorithms and Applications.* Springer-Verlag, New York, New York, 1997.

[62] D. F. Lawden. *Analytical Methods of Optimization.* Scottish Academic Press, Edinburgh, 1975.

[63] D. Lee and M. W. Spong. Stable Flocking of Multiple Inertial Agents on Balanced Graphs. *IEEE Transactions on Automatic Control*, 52(8):1469–1475, 2007.

[64] H. Levine and W. J. Rappel. Self-organization in systems of self-propelled particles. *Physical Review E*, 63(1):017101–1–017101–4, 2001.

[65] J. Lewin. *Differential Games: Theory and Methods for Solving Game Problems with Singular Surfaces.* Springer-Verlag, Berlin; New York, 1994.

[66] F. L. Lewis and V. L. Syrmos. *OptimalControl.* Wiley-Interscience, New York, New York, 2 edition, 1995.

[67] Y. Liu, K. Passino, and M. M. Polycarpou. Stability Analysis of M-Dimensional Asynchronous Swarms With a Fixed Communication Topology. *IEEE Transactions on Automatic Control*, 48(1):76–95, 2003.

[68] R. K. Maloy, K. Y. Lee, and L. H. Sibul. A pursuit-evasion differential game in relative polar coordinates with state estimation. In *Proceedings of the Americal Control Conference*, pages 2463–2467, 1996.

[69] M. Mamei, F. Zambonelli, and L. Leonardi. Cofields: a physically inspired approach to motion cordination. *IEEE Pervasive Computing*, 3(2):52–61, 2004.

[70] S. Mannor and J.S. Shamma. Multi-agent learning for engineers. *Artificial Intelligence, Special issue on "Foundations of Multi-Agent Learning"*, pages 417–422, 2007.

[71] J. Maynard-Smith. *Evolution and the Theory of Games.* Cambridge University Press, Cambridge, Massachusetts, 1982.

[72] A. Mehlmann. *Applied Differential Games.* Plenum Press, New York, 1988.

[73] A. W. Merz. The homicidal chauffeur. *AIAA Journal*, 12(3):259–260, 1974.

[74] R. B. Myerson. *Game Theory: Analysis of Conflict.* Harvard University Press, Cambridge, Massachusetts, 1991.

[75] M. Mynsk. *The Society of Mind.* Simon & Schuster, New York, New York, 1986.

[76] J. F. Nash. The Bargaining Problem. *Econometrica*, 18(2):155–162, 1950.

[77] J. F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950.

[78] J. F. Nash. Noncooperative Games. *Annals of Mathematics*, 54(2):289–295, 1951.

[79] J. F. Nash. Two-Person Cooperative Games. *Econometrica*, 21(1):128–140, 1953.

[80] M. F. Norman. *Markov Processes and Learning Models.* Academic Press, New York, New York, 1972.

[81] A. Okubo. Dynamical aspects of animal grouping: swarms, schools, flocks, and herds. *Advances in Biophysics*, 22:1–94, 1986.

[82] R. Olfati-Saber. Flocking for Multi-Agent Dynamic Systems: Algorithms and Theory. *IEEE Transactions on Automatic Control*, 51(3):401–420, 2006.

[83] A. S. Posnyak and K. Najim. *Learning Automata and Stochastic Optimization.* Springer, Berlin; New York, 1997.

[84] A. S. Poznyak and C. J. Gallegos. Multimodel Prey-Predator LQ Differential Games. In *Proceedings of the Americal Control Conference*, pages 5369–5374, 2003.

[85] J. Robinson. An iterative method of solving a game. *The Annals of Mathmatics*, 54(2):296–301, 1951.

[86] D. Ruelle. *Chance and Chaos.* Princeton University Press, Princeton, NJ, 1991.

[87] A. P. Sage and C. C. White. *Optimum Systems Control.* Prentice Hall, Inc., Englewood Cliffs, New Jersey, 2 edition, 1977.

[88] E. Sahin. Swarm robotics: From sources of inspiration to domains of application. In E. Sahin and W. M. Spears, editors, *Swarm Robotics: SAB 2004 international workshop, Santa Monica, CA, USA, July 17, 2004 : revised selected papers*, pages 10–20, Berlin, Heidelberg, 2005. Springer-Verlag.

[89] P. S. Sastry, V. V. Phansalkar, and M. A. L. Thathachar. Decentralized learning of nash equilibria in multi-person stochastic games with incomplete information. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(5):769–777, 1994.

[90] J. S. Shamma and G. Arslan. Unified convergence proofs of continuous-time fictitious play. *IEEE Transactions on Automatic Control*, 49(7):1137–1142, 2004.

[91] J. S. Shamma and G. Arslan. Dynamic fictitious play, dynamic gradient play, and distributed convergence to nash equilibria. *IEEE Transactions on Automatic Control*, 50(3):312–327, 2005.

[92] L. S. Shapley. Stochastic games. *Proc. Nat. Acad. Sci. U.S.A.*, 39:1095–1100, 1953.

[93] P. P. Shenoy. On coalition formation: A game-theoretical approach. *International Journal of Game Theory*, 8(3):133–164, 1979.

[94] J. W. Sheppard. Colearning in differential games. *Machine Learning*, 33:201–233, 1998.

[95] S. Stahl. *A Gentle Introduction to Game Theory.* American Mathematical Society, Providence, R.I., 1999.

[96] A. W. Starr and Y. C. Ho. Nonzero-sum differential games. *Journal of Optimization Theory and Applications*, 3(3):184–206, 1969.

[97] P. Stone and M. Veloso. Team-partitioned, opaque-transition reinforcement learning. In M. Asada and H. Kitano, editors, *RoboCup-98: Robot Soccer World Cup II.* Springer Verlag, Berlin, 1999.

[98] P. D. Straffin. *Game theory and strategy.* The Mathematical Association of America, Washington, DC, 1993.

[99] P. Suppes and R. C. Atkinson. *Markov Learning Models For Multiperson Interactions.* Stanford University Press, Stanford, California, 1960.

[100] R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduction.* The MIT Press, Cambridge, Massachusetts, 1998.

[101] T. Takagi and M. Sugeno. Fuzzy identification of systems and its application to modeling and control. *IEEE Transactions on Systems, man, Cybernetics,* 15:116–132, 1985.

[102] M. A. L. Thathachar and P. S. Sastry. *Networks of Learning Automata: Techniques for Online Stochastic Optimization.* Kluwer Academic Publishers, Boston, Massachusetts, 2004.

[103] J. van der Wal. *Stochastic Dynamic Programming.* PhD thesis, Technische Hogeschool Eindhoven, 1980.

[104] J. von Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior.* Princeton University Press, Princeton, 2nd edition, 1947.

[105] P. Vrancx, K. Verbeeck, and A. Nowe. Decentralized learning in markov games. *IEEE Transactions on Systems, Man, and Cybernetics - Part B,* 38(4):976–981, 2008.

[106] L. X. Wang. *A Course in Fuzzy Systems and Control.* Prentice Hall, Englewood Cliffs, NJ, 1997.

[107] Y. Wang and C. Xu. Adaptive algorithm for multi-agent learning optimal cooperative pursuit strategy based on markov game. In *Third International Conference on Machine Learning and Cybernetics*, pages 2973–2978, 2004.

[108] J. W. Weibull. *Evolutionary Game Theory.* MIT Press, Cambridge, Massachusetts, 1995.

[109] R. M Wheeler and K. S. Narendra. Decentralized learning in finite markov chains. *IEEE Transactions on Automatic Control*, 31(6):519–526, 1986.

[110] D. W. K. Yeung and L. A. Petrosyan. *Cooperative Stochastic Differential Games.* Springer Science+Business Media, Inc., New York, NY, 2006.

[111] D. Yingying, H. Yan, and J. Jing-ping. Self-organizing multi-robot system based on personality evolution. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 5, 2002.

[112] P. Zhou, B. Hong, Y. Wang, and T. Zhou. Multi-agent cooperative pursuit based on extended contract net protocol. In *Third International Conference on Machine Learning and Cybernetics*, pages 169–173, 2004.