Intelligent Agents Coordination in Ad Hoc Teams

Rodrigo Catarino Antunes de Perez Monteiro

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors:  Prof. Francisco António Chaves Saraiva de Melo
              Prof. José Alberto Rodrigues Pereira Sardinha

Examination Committee
Chairperson:  Prof. António Manuel Ferreira Rito da Silva
Supervisor:   Prof. Francisco António Chaves Saraiva de Melo
Member of the Committee: Prof. Manuel Fernando Cabido Peres Lopes

November 2016
Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisor Professor Francisco Melo and my co-advisor Professor Alberto Sardinha, for the continuous support of my MSc study and research, for their motivation and for the knowledge they were eager to share with me. Their guidance helped me in all the time of research and writing of this thesis.

Besides my advisor(s), I would like to thank Professor Pedro Sequeira for his encouragement, insightful comments, and hard questions during the project stage of my thesis.

My sincere thanks also goes to Professor Babak Esfandiari of Carleton University, in Canada, whose insights on how to build a Robocup 2D Simulation League were priceless to help my research, my boss at INCM, Luís de Matos, for the flexibility he provided me with, allowing me to dedicate time to my dissertation development and my colleague Paulo Carvalho for the proofreading of my dissertation document.

I also thank, in no particular order, my fellow classmates at Instituto Superior Técnico: Pedro Francisco, Tiago Sampaio, Miguel Deus, Paulo Martins, Ricardo Sequeira, Tiago Catarino, António Pólvora and Patrícia Santos, for the sleepless nights we were working together before project deadlines, for the numerous occasions in which we spent the day studying together for exams and for all the fun we have had in the last five years.

Last but not the least, I would like to thank my family: my brother Gonçalo, my parents Adriano and Gisela, and also my stepfather Francisco, for all the support I have been given throughout the years, without which none of this would have been possible.
Resumo

O principal alvo deste documento é o campo científico do trabalho de equipa Ad Hoc (Ad Hoc Team-work), no qual um agente inteligente ocupa uma posição numa equipa de agentes previamente construída. Este agente deve ser capaz de se coordenar com a equipa, sem ter tido conhecimento prévio das suas estratégias ou protocolos de comunicação, com o objetivo de atingir uma meta comum, no nosso caso em particular, ganhar um jogo da Liga de Simulação 2D da RoboCup. A dificuldade de desenvolver uma solução que tenha boa performance em domínios complexos constitui um grande desafio.

Assumimos uma abordagem que consiste em identificar a tarefa, os colegas de equipa e, posteriormente, elaborar planos para atingir o nosso objetivo. Aplicando técnicas de aprendizagem automática, integramos o nosso agente num domínio dinâmico, multi-agente (Liga de Simulação 2D RoboCup), cuja complexidade é muito superior à dos domínios tradicionais usados nos problemas de trabalho de equipa Ad Hoc.

Para atingir este objetivo, modelámos um Processo de Decisão de Markov (MDP) em que a função de recompensa é inicialmente desconhecida e, mais tarde, será obtida usando Aprendizagem por Reforço Inversa (IRL). Os restantes agentes e a formação da equipa são classificadas usando técnicas de aprendizagem supervisionada. Finalmente, o planeamento consiste em resolver o MDP usando um algoritmo de Iteração de Políticas. Com os resultados que obtivemos utilizando esta abordagem, provamos que é possível usar o trabalho de equipa Ad Hoc em domínios multi-agente complexos.

**Palavras-chave:** Agentes inteligentes; Sistemas multi-agente; Aprendizagem automática; Coordenação Ad Hoc; Processos de Decisão de Markov; Aprendizagem por Reforço Inversa
Abstract

This document's main target is the scientific field of Ad Hoc teamwork, in which an intelligent agent secures a position in a previously assembled team of agents. It must be able to coordinate with that team, without formerly acquired knowledge about their team strategies or communication protocols with the purpose of achieving a common goal (in our case, winning a RoboCup 2D Simulation League match). The difficulty of developing a solution that performs in complex domains poses a great challenge for us to solve.

We assume an approach that consists in identifying the task, teammates and planning towards achieving that goal. By applying machine learning techniques, we deployed our agent in a dynamic multi-agent environment (RoboCup 2D Simulation League), whose complexity is far bigger than the conventional settings used in Ad Hoc teamwork problems.

To achieve this goal, we modeled a Markov Decision Process (MDP) where the reward is initially unknown and is later obtained using Inverse Reinforcement Learning (IRL). The other agents and the formation are classified using supervised learning techniques. Finally, the planning consists in solving the MDP via a Policy Iteration algorithm. With the results obtained with this approach, we prove that it is possible to successfully use Ad Hoc teamwork in complex multi-agent domains.

Keywords: Intelligent Agents; Multi-agent Systems; Automated Learning; Ad Hoc Teamwork; Markov Decision Processes; Inverse Reinforcement Learning
# Contents

<table>
<thead>
<tr>
<th>Acknowledgments</th>
<th>iii</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resumo</td>
<td>v</td>
</tr>
<tr>
<td>Abstract</td>
<td>vii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xi</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xi</td>
</tr>
<tr>
<td>Nomenclature</td>
<td>xiii</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Problem Description  
1.2 Contributions  
1.3 Document Outline  

## 2 Background

2.1 Markov Chains  
2.1.1 Markov Chain Monte Carlo - MCMC  
2.2 Markov Decision Process - MDP  
2.2.1 Policy Iteration  
2.2.2 Inverse Reinforcement Learning  
2.3 RoboCup  
2.3.1 RoboCup 2D Soccer Simulation League  

## 3 Related Work

3.1 Historical Perspective  
3.2 Impromptu Teams  
3.3 ad hoc agent teams  
3.3.1 Apprenticeship learning  
3.3.2 Agents modelling  
3.3.3 Leading teams of agents  
3.4 The ad hoc teamwork problem in RoboCup  

## 4 Multi-layer approach to Ad Hoc Teamwork

4.1 Architecture  

ix
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.1 Domain Specific Architecture for RoboCup 2D Simulation League</td>
<td>27</td>
</tr>
<tr>
<td>4.2 Team construction and Domain information</td>
<td>28</td>
</tr>
<tr>
<td>4.3 Teammate identification</td>
<td>32</td>
</tr>
<tr>
<td>4.3.1 Player classifier</td>
<td>32</td>
</tr>
<tr>
<td>4.4 Task identification</td>
<td>34</td>
</tr>
<tr>
<td>4.4.1 MDP modelling</td>
<td>34</td>
</tr>
<tr>
<td>4.4.2 Formation classifier</td>
<td>34</td>
</tr>
<tr>
<td>4.4.3 Obtaining the reward function via IRL</td>
<td>36</td>
</tr>
<tr>
<td>4.5 Planning</td>
<td>39</td>
</tr>
<tr>
<td>5 Results</td>
<td>41</td>
</tr>
<tr>
<td>5.1 Evaluation Method</td>
<td>41</td>
</tr>
<tr>
<td>5.2 Teammate Identification Tests</td>
<td>43</td>
</tr>
<tr>
<td>5.3 Task Identification Tests</td>
<td>44</td>
</tr>
<tr>
<td>5.4 In-Match Agent and Team Performance</td>
<td>49</td>
</tr>
<tr>
<td>5.4.1 Baseline Performance</td>
<td>49</td>
</tr>
<tr>
<td>5.4.2 Ad Hoc Agent/Team Performance</td>
<td>51</td>
</tr>
<tr>
<td>6 Conclusions</td>
<td>55</td>
</tr>
<tr>
<td>6.1 Approach Overview</td>
<td>55</td>
</tr>
<tr>
<td>6.2 Main Contributions Overview</td>
<td>55</td>
</tr>
<tr>
<td>6.3 Limitations and Weaknesses</td>
<td>56</td>
</tr>
<tr>
<td>6.4 Future Work</td>
<td>56</td>
</tr>
<tr>
<td>Bibliography</td>
<td>57</td>
</tr>
<tr>
<td>A Approach Diagram</td>
<td>61</td>
</tr>
<tr>
<td>B Domain Specific Approach Diagram</td>
<td>63</td>
</tr>
<tr>
<td>C Player Classification</td>
<td>65</td>
</tr>
</tbody>
</table>
List of Tables

4.1 Our playbook's set of formations ............................................................... 29
4.2 States' definition based on agent's observations ................................. 30
4.3 Players' policies by role ........................................................................... 31
4.4 List of actions available to the agent ...................................................... 32
4.5 A teammate's best predictor classification example ................................. 33
4.6 A teammate's mixed predictor classification example ............................ 33

5.1 Teammate identification test results (average scores and classifications) - clear data . . . 44
5.2 Teammate identification test results (average scores and classifications) - noisy data . 44
5.3 Task identification test results - best predictor model - formation selection - clear data . . 45
5.4 Task identification test results - best predictor model - role selection - clear data . . . 45
5.5 Task identification test results - mixed predictor model - formation selection - clear data . . 46
5.6 Task identification test results - mixed predictor model - formation selection - noisy data . . 47
5.7 Task identification test results - mixed predictor model - role selection - clear data . . . 48
5.8 Task identification test results - mixed predictor model - role selection - noisy data . . . 49
5.9 Test matches' data - regular teams ............................................................... 50
5.10 Test matches' statistics - regular teams ..................................................... 50
5.11 Test matches' statistics - ad hoc teams - noisy data .................................. 52
5.12 Test matches' data - ad hoc teams - agent stats - noisy data ................. 53
List of Figures

2.1 MDP example ................................................................. 7
2.2 Example IRL problem .................................................. 9
2.3 RoBoCup 2D Simulation League SoccerServer environment ...... 11

3.1 Referential understanding in a human dialogue ....................... 14
3.2 Successful completion in a human dialogue ........................ 14
3.3 Corner play example from CMDragons team's playbook .......... 15
3.4 Graphical representation of the pursuit domain ..................... 19
3.5 Example of a flock of two agents led by four ad hoc agents ...... 22
3.6 Two teams of NAO robots playing in an SPL game ................. 23

4.1 The three steps of our approach to ad hoc teamwork ................ 25
4.2 Approach architecture illustration .................................... 27
4.3 Domain specific approach architecture diagram .................... 28
4.4 Swarming behavior example ............................................. 30
4.5 Example ad hoc Formation ............................................... 35
4.6 The BIRL model ............................................................ 37

A.1 Approach architecture illustration - enlarged ......................... 62

B.1 Domain specific approach architecture diagram - enlarged ....... 64

C.1 Player classification process ............................................. 65
## Nomenclature

### Greek symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta$</td>
<td>Step size.</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Discount factor.</td>
</tr>
<tr>
<td>$\pi$</td>
<td>Policy.</td>
</tr>
</tbody>
</table>

### Roman symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>Set of actions.</td>
</tr>
<tr>
<td>$O$</td>
<td>Set of observations.</td>
</tr>
<tr>
<td>$R$</td>
<td>Reward function.</td>
</tr>
<tr>
<td>$S$</td>
<td>Set of states.</td>
</tr>
<tr>
<td>$T$</td>
<td>Transition function.</td>
</tr>
<tr>
<td>$V$</td>
<td>Discounted value of a policy.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Autonomous agents usage is increasing in recent years, with applications in areas such as health (Nealon and Moreno, 2003) and robotics (Jones et al., 2006; Abbeel, 2008). The need for cooperative agents and agent teams arises, which is supported by the notion that a team of agents that coordinate their efforts to reach a common goal has a better performance, in general.

Even though attaining cooperation is a complicated goal, there is a considerable amount of literature on successful techniques for putting together a team of agents that performs well. A good example of these techniques is the early work of Tambe (1997) on flexible teamwork in complex, dynamic scenarios, such as the RoboCup synthetic soccer domain, where the author advances a model of teamwork that enables a team of agents, with inconsistent views of the world, to act in a coherent manner. The premise is that providing the agents in a team with the same teamwork model will enable them to overcome the conflicts created by the complexity of the environment and incoherent world models.

We often face situations where cooperation amongst agents that do not have prior knowledge about one another, do not communicate in the same way and may even have different world models, is needed in order to achieve certain goal or fulfill a given task. Some newly developed agents may need to be deployed in an environment where other agents are already operating. In some of these cases, the agents in place may no longer be altered, either due to financial reasons (the cost of upgrading them is not worth the effort) or their developer being out of business, for instance. These agents are known as legacy agents. To illustrate the utility of an ad hoc agent in a similar scenario, consider Example 1.

Example 1. Consider an industrial scenario, where several agents (robots) perform different tasks in an assembly line, from the manufacturing of the components to the assembly of the final product. If one of the agents malfunctions or has a performance below standard, it jeopardizes the entire business and costs the company money and time. In order to achieve maximum efficiency, that agent must be replaced, but that is a costly effort and the new agent may not know what its function is, causing the entire team to under-perform. Therefore, it is worthwhile to have an agent that observes the behaviour of its teammates and can quickly use that knowledge to fill in the void.
This concept of collaboration without explicit coordination and knowledge about the teammates has been discussed under the name of *ad hoc teamwork*. The main challenge in ad hoc teamwork, as described by [Stone et al. (2010)](#), is "to create an autonomous agent that is able to efficiently and robustly collaborate with previously unknown teammates on tasks to which they are all individually capable of contributing as team members".

In different domains, some related work has already been developed. For instance, [Bowling and McCracken (2005)](#) introduce the concept of *impromptu teams* in robot soccer, where an agent is placed in a team with previously unknown teammates and must learn the team’s strategies and game plan so that its behaviour maximizes the chance of its team winning the match.

More recent work, by [Melo and Sardinha (2016)](#), goes even further than the aforementioned by considering not only that the teammates are unknown but also the task to be performed, which introduces a new requirement to ad hoc teamwork, *task identification*. Despite these improvements, as the computational weight of the techniques applied increases dramatically with the number of actions available to the agent at a given time and the overall complexity of the task to be performed, the set of domains where these methods can be applied successfully is still somewhat limited. This is the main issue addressed in this dissertation.

### 1.1 Problem Description

[Melo and Sardinha (2016)](#) described ad hoc teamwork as a 3-step problem which includes *task identification, teammate identification and planning*. Although these have already been thoroughly scrutinized, there is a scalability issue: to the best of our knowledge, there still is no solution that successfully applies this 3-step approach to a complex domain where there are many agents in a team, many possible tasks to be performed and many possible actions available to each agent at a given time (with each action impacting the performance of the team as a whole in a different manner).

The techniques that have been studied so far, and combinations thereof, are still unable to provide an answer to the ad hoc teamwork problem which allows its application in a more complex domain, namely RoboCup simulation league.

RoboCup 2D Simulation League is a branch of the RoboCup tournament which consists in a 2D soccer match between two teams of eleven agents each, each of them having several sensory inputs and several possible actions at a given time. The main mid-term research issues involved in this area of the RoboCup challenge, as described by [Kitano et al. (1998)](#), involve:

1. Machine learning in a multi-agent, collaborative and adversarial environment
2. Multi-agent architectures, enabling real-time multi-agent planning and plan execution in service of teamwork
3. Opponent modelling

Ensuing the above stated, we can conclude that the RoboCup 2D Simulation League provides us with a good domain to investigate the scalability of our Ad Hoc Teamwork approach.
We face a situation in which our ad hoc agent is deployed in a complex, adversarial environment and will try to coordinate with its team, to the best of its abilities, aiming towards an initially unknown goal. Moreover, this environment is unusually convoluted when compared to the classic ad hoc teamwork settings. Such environments include the pursuit domain, where a set of 4 agents tries to surround a prey (which moves randomly) in a grid world (Stone and Veloso 2000) and half field offense, which is a succession of episodic tasks in which an offense team of \( m \) players has to outsmart the defense team of \( n \) players to score a goal (Kalyanakrishnan et al. 2006). The latter can be framed as a learning problem that has the goal of increasing the goal-scoring performance of the attacking team, while the defensive team keeps a fixed strategy. These domains are somewhat limited in their complexity and number of agents. Thus, the problem we have in hands can be epitomized by the following question: "How can ad hoc teamwork be scaled to complex environments containing a large number of agents?"

1.2 Contributions

As mentioned before, we aim to create an agent that successfully integrates a team of agents in a complex environment, without being able to explicitly coordinate with its team members.

When deploying our agent in the “Robocup 2D Simulation League environment, we want to create an autonomous agent that is able to:

1. Replace one of the agents in a previously assembled team
2. Figure out what the task at hand is
3. Identify its teammates
4. Become a part of the team
5. Improve the team’s performance

With the successful application of our agent in a testing environment such as the Robocup 2D Simulation League, we distance ourselves from the conventional ad hoc teamwork environments, towards a more complex one, thus addressing the scalability issue frequently associated to ad hoc teamwork solutions, which is the main contribution of our work.

Furthermore, we reach beyond the traditional view of ad hoc teamwork by implementing a solution that takes into account the fact that planning is not the only issue in play, despite its importance. Following on the work of Melo and Sardinha (2016) we adopt a perspective on ad hoc teamwork which regards task identification, teammate identification and planning as the three key steps involved in this problem.

Unlike the aforementioned approach, we address these particular issues in a slightly different order. First, we tackle teammate identification using a supervised learning technique based on observations collected from the teammates. This provides us with the teammates’ models. Then, we address task identification, where we find a suitable reward policy for a Markov Decision Process (MDP), that represents our domain, using Inverse Reinforcement Learning (IRL) based on observations from a specific
teammate, which is regarded as an expert in the domain. Finally, planning consists in solving the MDP via a Policy Iteration algorithm.

1.3 Document Outline

In the rest of the document we will proceed as follows: in the second chapter we present some details pertaining to the context where we developed our work (RoboCup 2D simulation league), then in the third chapter we describe, in some detail, the work that has already been developed in this field and how it relates to our own, by analyzing the relevant literature. In Chapter 4, we present our work’s architecture, in detail. We wrap the document by describing how we evaluated our solution and what results we obtained. Finally, Chapter 6 concludes this document by summarizing our main achievements, our approach’s weaknesses and some future work that may be build upon ours.
Chapter 2

Background

In this chapter we cover background material that will be useful in the remainder of the document. We start by advancing some concepts with respect to Markov Decision Processes, which will be used to model our particular problem and Markov Chains / Markov Chain Monte Carlo algorithms, which we will use in the task identification step of our approach. Afterwards, we present the RoboCup and Robocup 2D Simulation League domain, in which we will test our approach to ad hoc teamwork.

2.1 Markov Chains

In this section, we will define some basic notions regarding Markov Chain(s) that will be useful when defining what a Markov Decision Process is. This knowledge will be critical to the implementation of our approach.

Modern probability theory studies chance processes for which the knowledge of previous outcomes influences predictions for future experiments. In principle, when we observe a sequence of chance experiments, all of the past outcomes could influence our predictions for the next experiment. A Markov Chain is a type of chance process in which the outcome of a given experiment can affect the outcome of the next experiment [Hastings 1970]. Markov Chains respect the Markov property: predictions are made based only on the current state of the system, and not on any previous states.

We define a Markov Chain as a tuple \((S, A, T)\) where:

- \(S = \{s_1, \ldots, s_n\}\) is a finite set of \(n\) states.
- \(A = \{a_1, \ldots, a_k\}\) is a finite set of \(k\) actions.
- \(T : S \times A \times S \to [0, 1]\) is a transition probability function that represents the probability of transitioning from any given state to any of the states \(s \in S\).

2.1.1 Markov Chain Monte Carlo - MCMC

Markov Chain Monte Carlo (MCMC) methods are a class of algorithms for sampling from a probability distribution based on constructing a Markov Chain that has the desired distribution as its equilibrium
distribution. It is a technique used for estimating by simulation the expectation of a statistic in a complex model (Gilks, 2005). An example of an MCMC algorithm is Policy Walk, which is a modified version of another MCMC algorithm, Grid Walk (Vempala, 2005), that generates a Markov Chain. We use the Policy Walk algorithm in our approach and it is described in detail in Chapter 4.

2.2 Markov Decision Process - MDP

In this section, we will define some basic notions regarding Markov Decision Process(es) that will be used throughout the document and is, therefore, essential for interpreting it correctly.

The notion of Markov Decision Process has been around since the 1950’s (Bellman, 1957). Markov Decision Processes provide a mathematical framework for decision making and are frequently used in optimization problems in robotics, economics and other scientific fields. Markov Decision Processes also respect the Markov Property, analogously to Markov Chains.

A Markov Decision Process is an extent of a Markov Chain in which there are multiple available actions per state and there are rewards that an agent gets for the transition from a state to another. It describes the problem of an agent that selects its action in each state so it maximizes the discounted rewards it gets.

Ramachandran and Amir (2007) define a Markov Decision Process (also known as an MDP) as a tuple \((S, A, T, \gamma, R)\) where:

- \(S = \{s_1, ..., s_n\}\) is a finite set of \(n\) states.
- \(A = \{a_1, ..., a_k\}\) is a finite set of \(k\) actions.
- \(T : S \times A \times S \rightarrow [0, 1]\) is a transition probability function that represents the probability of transitioning from any given state to any of the states \(s \in S\).
- \(\gamma \in [0, 1]\) is the discount factor, which represents the difference in importance between future rewards and present rewards.
- \(R : S \times A \times S \rightarrow \mathbb{R}\) is a reward function.

An example of a small MDP can be found in Figure 2.1. In an MDP, the goal of the agent is to select a policy (a function \(\pi(s)\) that for each state, \(s\), dictates which action the decision maker will choose from a set of available actions when in that state) that gets as much discounted reward as possible, i.e., to select a policy \(\pi^*\) such that:

\[
\forall s \in S, \pi^* = \arg \max_{\pi} V^\pi(s) = \arg \max_{\pi} E_{T}[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t, S_{t+1})]
\] (2.1)

This process is commonly known as "solving the MDP".

We refer to \(V^\pi(s)\) as the value of policy \(\pi\) at state \(s\), corresponding to the expected total discounted reward that the agent receives if it starts in state \(s\) and follows policy \(\pi\). Similarly, we define \(Q^\pi(s, a)\) as
the expected total discounted reward that the agent receives if it starts in state $s$, selects action $a$, and follows policy $\pi$ afterwards.

Furthermore, we will be using the additional following notation:

- $A_s$ is the finite set of actions available from state $s$.
- $P_a(s, s') = T(s, a, s')$, i.e., it is the probability that action $a$ in state $s$ leads to state $s'$.
- $R_a(s, s') = R(s, a, s')$, i.e., is the reward that the agent gets for the transition from state $s$ to state $s'$.
- $\pi : S \rightarrow A$ is a policy.
- $V^\pi(s) \rightarrow \mathbb{R}$ is the discounted value of a policy $\pi$ at state $s \in S$.
- $Q^\pi(s, a) \rightarrow \mathbb{R}$ is the expected utility of taking action $a$ in state $s$ and following the policy $\pi$ afterwards.

Two of the classical results regarding MDP’s are the Bellman equations in theorem 1. They will be useful when trying to find a solution for the inverse reinforcement learning process described in Section 4.4.3. Furthermore, theorem 2 describes the conditions in which a policy $\pi$ is considered optimal (in simple terms, $\pi$ is an optimal policy if for every state, $s$, in the set of states $S$, its value is the highest of all available actions, $a$, in that state, in $Q^\pi(s, a)$).

**Theorem 1 (Bellman Equations).** Let a Markov Decision Problem $M = (S, A, T, \gamma, R)$ and a policy $\pi : S \rightarrow A$ be given, then: $\forall s \in S, \forall a \in A, V^\pi$ and $Q^\pi$ satisfy:

\[
V^\pi(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') V^\pi(s') \\
Q^\pi(s, a) = R(s) + \gamma \sum_{s'} T(s, a, s') V^\pi(s')
\]

**Theorem 2 (Bellman Optimality).** Let a Markov Decision Problem $M = (S, A, T, \gamma, R)$ and a policy $\pi : S \rightarrow A$ be given, then $\pi$ is an optimal policy for $M$ iff $\forall s \in S$:

\[
\pi(s) \in \arg\max_{a \in A} Q^\pi(s, a)
\]
In order to solve an MDP, we use an algorithm known as Policy Iteration, which guarantees that we attain an optimal policy \( \pi^* \) that for each state, \( s \), returns an action for the agent to perform, \( a \), with the highest expected discounted reward. This solution method is explained in the following section.

### 2.2.1 Policy Iteration

*Policy Iteration* is a dynamic programming algorithm that iterates over a policy, generating a sequence of monotonically improving policies \( \pi(s) \) and value functions \( V^\pi \) iteration after iteration, until the optimal policy is reached.

Given a baseline policy, an improved policy can be computed. By repeating this process we can generate monotonically improved policies, using the policy from the previous iteration. Since there are a finite number of states and a finite number of actions, this will eventually terminate with a policy that cannot be further improved. This is, in fact, an optimal policy.

The policy iteration algorithm consists in two simple steps: *policy evaluation* and *policy improvement.* In the first step we obtain the value for the discounted policy \( \pi \) for each state \( s \in S \), \( V^\pi(s) \). In the second and final step, we improve the policy, \( \pi \), by selecting for each state, \( s \), the action, \( a \), that yields the best expected value, considering our current transition function, \( T^\pi \), and our current reward function, \( R^\pi \).

This process is summed up in the Algorithm 1 description, below.

**Algorithm 1:** Policy iteration algorithm

<table>
<thead>
<tr>
<th>Data:</th>
<th>Random initial policy ( \pi )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result:</td>
<td>Optimal policy ( \pi^* )</td>
</tr>
<tr>
<td>while ( \pi ) is changed do</td>
<td></td>
</tr>
<tr>
<td>1. policy evaluation solve</td>
<td></td>
</tr>
<tr>
<td>[ V^\pi(s) = \sum_{s' \in S} T^\pi(s, \pi(s), s')[R^\pi(s, \pi(s), s') + \gamma V^\pi(s')] ] ( \forall s \in S ) \hspace{1cm} (2.5)</td>
<td></td>
</tr>
<tr>
<td>2. policy improvement</td>
<td></td>
</tr>
<tr>
<td>[ \pi(s) \leftarrow \arg \max_a \sum_{s' \in S} T^\pi(s, a, s')[R^\pi(s, a, s') + \gamma V^\pi(s')] ] \hspace{1cm} (2.6)</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

The value function of a policy is just the expected reward (discounted) that will be gained, at each state, by applying that policy. Once we know the value of each state under the current policy, we consider whether the value could be improved by changing the first action taken. If it can, we change the policy to select the new action when it is in that situation. This step is guaranteed to strictly improve the performance of the policy. When no improvements are possible, then the policy is guaranteed to be optimal. Since there are at most \( |A|^{|S|} \) distinct policies, and the sequence of policies improves at each step, this algorithm terminates in at most an exponential number of iterations [Hazeghi](1995).
2.2.2 Inverse Reinforcement Learning

The MDP that represents our domain has an initially unknown reward function, which we want to find in the task identification step of our approach to ad hoc teamwork. The way we do that is through a technique known as Inverse Reinforcement Learning (IRL).

The problem of deriving a reward function from observed behavior is referred to as Inverse Reinforcement Learning (Ng et al., 2000). We assume that the expert is trying to optimize an unknown reward function that can be expressed as a linear combination of known features, as described by Abbeel (2008). An example of an IRL problem can be found in Figure 2.2.

Figure 2.2: An example IRL problem. Bold lines represent the optimal action $a_1$ for each state and broken lines represent some other action $a_2$. Action $a_1$ in $s_1$ has probabilities 0.4, 0.3 and 0.3 of going to states $s_1$, $s_2$, $s_3$ respectively, and all other actions are deterministic, from Ramachandran and Amir (2007).

The IRL problem is to find a reward function (for an MDP) that can explain the observed behavior. We begin with a tuple $(S, A, T, \gamma)$ where:

- $S = \{s_1, ..., s_n\}$ is a finite set of $n$ states.
- $A = \{a_1, ..., a_k\}$ is a set of $k$ actions.
- $T : S \times A \times S \rightarrow [0, 1]$ is a transition probability function that represents the probability of transitioning from any given state to any of the states $s \in S$.
- $\gamma \in [0, 1]$ is the discount factor, which represents the difference in importance between future rewards and present rewards.

and we want to find a set of possible reward functions, $R : S \times A \times S \rightarrow \mathbb{R}$ such that the corresponding policy, $\pi$, is an optimal policy in the MDP $(S, A, T, \gamma, R)$ (Ng et al., 2000).
2.3 RoboCup

In this section we will introduce the Robocup environment in which we will test our approach to ad hoc teamwork. We will also cover some notions regarding the information provided by the Robocup 2D Simulation League Soccer Server and the actions available to an agent in that domain.

RoboCup is an annual international robotics competition founded in 1997 aiming to stimulate interest in artificial intelligence and robotics research. Its ambitious goal was set from the beginning:

> By the middle of the 21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup.

(Robocup-Federation 2015)

2.3.1 RoboCup 2D Soccer Simulation League

RoboCup 2D Soccer Simulation League is the branch of the competition we are going to focus on, which consists in simulated soccer matches between two teams of autonomous agents, each playing in a two-dimensional virtual soccer stadium, all this running on a server called SoccerServer (illustration in Figure 2.3). SoccerServer enables autonomous agents with programs written in several different languages to be put together in teams and play a soccer match against each other, in a client/server exchange, where each client represents an agent that communicates with the server using UDP/IP sockets (Chen et al., 2002).

The server provides the agents with the information regarding their three sensors:

1. Aural sensor: perceives the messages sent from the server (also known as the referee), the coach and the other agents/players.
2. Body sensor: perceives information regarding the agent itself (the speed it is moving at, which direction it is looking and its stamina level).
3. Visual sensor: perceives the surroundings of the agent, such as ball position and team mates’ positions, that are in the agent’s scope of vision.

Using this constant influx of information from the server to simulate self-awareness, each agent has to work towards maximizing the team’s overall performance. According to Nycander and Andersson (2013), who performed an analysis on a very successful team (WrightEagle), the key aspects that influence the performance of a virtual team are passing accuracy, stamina management, dribbling efficiency and team strategic positioning.

Each agent also has a set of attributes, whose values are available to it and change over time, which includes:

1 User Datagram Protocol (UDP) is a host-to-host layer protocol defined for transmitting datagrams between hosts on Internet Protocol (IP) networks.
• **ViewQuality**: varies from *low* to *high*. The higher it is, the less error on the input

• **ViewWidth**: varies between *narrow*, *normal* and *wide*. Determines the angle at which the agent can observe its surroundings.

• **Stamina**: decreases with each *dash* and increases a fixed amount, periodically. Its management is vital to the agent’s performance.

• **Effort**: is a value between effort *min* and effort *max*. It is dependent on the stamina management of the player.

• **AmountOfSpeed**: the speed at which the agent is moving.

• **DirectionOfSpeed**: the direction which the agent is moving towards.

• **HeadDirection**: the direction where the agent is “looking”.

The *Robocup* Simulation League agents also have a set of possible actions available to them at any given time, which allow it to interact with the environment that surrounds it. Those are:

• **turn**: changes the direction of the agent’s body.

• **turn-neck**: changes the direction of the agent’s neck, keeping the body as is.

• **catch**: catches the ball (if possible). This action is only available to goalkeepers.
• **dash**: accelerates the agent in the direction of its body.

• **kick**: given a kick angle and kick power (between $minpower$ and $maxpower$), the agent kicks the ball in that direction with that amount of power.

• **move**: places the agent in a given place on the field. Unlike dash, which can be used anywhere and at any time (as long as the game is under way), this command can only be used when the game is being set up and is not available when the game is under way.

• **say**: broadcasts a message from the agent.

and a set of internal counters that keep track of the amount of actions of each type that the agent has already performed:

• DashCount

• TurnCount

• SayCount

• TurnNeckCount

• CatchCount

• MoveCount

• ChangeViewCount

Granted that, given the amount of sensory inputs and actions available to the agent in a Robocup 2D Simulation League match, this domain poses a great challenge, it is also a challenging environment in which to test our approach. It provides us with a great extent of complexity, which we need in order to assess the validity of the approach to ad hoc teamwork we propose.
Chapter 3

Related Work

In this chapter we present and analyse the work that has already been developed in the field of ad hoc teamwork and how it relates to our own, by analyzing the relevant literature.

3.1 Historical Perspective

Early teamwork and coordination dates back to the late 80s/early 90s and, in one of the first contributions, Levesque et al. (1990) extend and improve upon their previous work on modelling the communication as rational interactions in a dynamic multi-agent domain (Cohen and Levesque, 1988) by defining joint intentions as shared commitments to perform an action while the group is in a certain shared mental state. The authors illustrate this distinction with a useful example: ordinary traffic (where there is coordination, provided by the traffic signals) versus driving in a convoy of cars, which requires cooperation and a common goal.

With this work, teamwork began to be regarded as less of a union of several individual actions (even if coordinated) and more of a joint action where a group of agents, that share a world model, acts as a single agent with beliefs, goals and intentions that represent those of the group.

Further work from Cohen and Levesque (1991) explores in more depth the importance of communication when performing joint team actions with a common goal, mainly through confirmation messages that an agent uses to inform the teammates of referential understanding, where an agent acknowledges the reception of a message (example in Figure 3.1) or successful completion of an action (example in Figure 3.2). The authors argue that the dialogue itself may be considered a joint action.

In the following examples we can see logs of two different parts of an interaction between a human expert (Exp) and an apprentice (Appr), which illustrate the usage of a task-oriented dialogue in a conversation over the phone. These confirmations are very much present in our daily conversations (making up about 18% of all that we say on the phone), thus proving its importance as a component of dialogue success.

Moving towards collaborative plans and execution of group tasks by a team of agents, Grosz and

---

\[1\] the agent’s world model is the internal interpretation it makes of its surrounding static objects, places, people or other agents.

13
**Exp:** Okay, I want you to take the largest tube, or actually it's the largest piece of anything, that has two openings on the side -

**Appr:** Yeah

**Exp:** - and threads on the bottom.

**Appr:** Yeah.

Figure 3.1: Referential understanding in a human dialogue, from Cohen and Levesque (1991)

**Exp:** And stick it on the en-on-to the uh spout coming out the side. You see that?

**Appr:** Yeah, okay

**Exp:** You got that on, okay?

**Appr:** Yeah.

Figure 3.2: Successful completion in a human dialogue, from Cohen and Levesque (1991)

[1990] Sidner promote a model of collaborative planning, in which two agents together form a plan to perform a complex action that requires both agents to contribute in order to be executed, SharedPlans. Despite its advantages, this approach could only handle activities that could be decomposed into single-agent actions (which is a problem when we want to perform complex joint activities) and the agents involved in the plan only had partial knowledge of the way to execute its part.

This model was later revised (Grosz and Sidner, 1996) to deal with these flaws by incorporating a new view on the concept of plan formation. In this work, the notion of plan is a mental-state view in which the plan is formed, given a set of intentions and beliefs that the agents have (their mental state). When a SharedPlan is being executed, some of the beliefs are individual and some are shared, and this makes the difference when trying to execute a plan that can not be decomposed into individual actions.

Later, Tambe (1997) builds on the work of Cohen and Levesque (1991) and Levesque et al. (1990) to implement a new model of teamwork - STEAM, a Shell for TEAMwork - that allows flexibility in communication and coordination to mitigate the obstacles that derive from the increasingly complex testing environments. Furthermore, in this work, the teammates are no longer assumed to have the same world models, which means that simply creating a model for each agent and each domain is not efficient, thus a more global solution to this kind of problem must be achieved. STEAM combines the concept of joint intentions present in Levesque et al. (1990)'s work with the hierarchy from Grosz and Sidner (1990)'s original formulation of the SharedPlans theory to create a hierarchy of joint intentions.

Although this last piece of work went a step further than its predecessors, by assuming there are joint intentions instead of just a set of combined individual intentions, the coordination mechanisms are still explicit, with all team members working together to complete a combined plan.
3.2 Impromptu Teams

Bowling and McCracken (2005) introduce a new concept of impromptu teams, in which a team is formed by a set of agents, each unknown to each other and each with its own skills and strategies, particularly on the case where only one of the members of the team is replaced by an independent one (the pickup player). Except for the pickup player, all the team members can communicate and coordinate normally, as they did before.

In the domain of robot soccer, the coordination, among the five robots that constitute the team, is achieved through the definition of a team strategy that specifies tactics for each team member, which are individual goals each robot has. Tactics are defined based on the robots’ individual skills, using a motion control algorithm and a path planning technique. Coordination can not be achieved through explicit communication because the agents may not be using the same communication protocols, thus making the exchange of information with the pickup player impossible.

The authors define two stages in the process of the pickup player’s integration in the already existing team, play selection and role selection. In the former, the pickup player decides which play it wants to execute and in the latter it assigns itself and its teammates a role in said play. The 1st stage is tested in two different variations: the adaptive variation where the agent uses learning to figure out which plays work best for its team based on which plays worked best in past situations, using Bowling et al. (2004)’s weight update play selection algorithm, and increases the likelihood of choosing plays that resulted in the pickup team doing well, and the predictive version, in which the agent selects the play based on the current position, trajectory of the teammates and the team’s current play style (defensive, regular or offensive). Both approaches performed equally well, with no significant differences.

Figure 3.3: Corner play example from CMDragons team’s playbook (in their play language) in Bowling and McCracken (2005).

An example of a play from a RoboCup team, CMDragons, where a player crosses a ball to a teammate, from a corner, to create a chance on goal, is depicted in Figure 3.3. The code is explained below:

```
PLAY Two Attackers, Pass from Corner

APPLICABLE offense in_their_corner
DONE aborted !offense

ROLE 1
  pass 3
  mark 0 from_shot

ROLE 2
  block 320 900 -1

ROLE 3
  position_for_pass { R { B 1000 0 } ... receive_pass
  shoot A

ROLE 4
  defend_line { -1400 1150 } ...
```

Figure 3.3: Corner play example from CMDragons team’s playbook (in their play language) in Bowling and McCracken (2005).
APPLICABLE offense in_their_corner

Definition of the predicates that must be true for the action to be selectable: the team must be in the offense and the ball must be in one of the rival team's corners.

DONE aborted !offense

Definition of the possible outcomes of the play and reaction to it: if the team is no longer on offense, then abort play.

ROLE 1
- pass 3
- mark 0 from_shot

ROLE 2
- block 320 900 −1

ROLE 3
- position_for_pass { R { B 1000 0 } ...  
- receive_pass
- shoot A

ROLE 4
- defend_line { −1400 1150 } ...

Definition of the roles of each of the non-goalie players in this play: the agent in role 1 passes the ball, the agent in role 2 blocks some rival on a given position, the one in role 3 runs toward a pre-determined position to wait for the pass and receive it and the last one (on role 4) plays defense.

Note that, unlike the work we propose, in this scenario, the team strategies and task are known a priori. Obtaining a coordinated behavior between the existing team and the new agent is an even bigger challenge for us.

3.3 ad hoc agent teams

Traditionally, teams of agents are programmed by the same person or the same group of people, however, moving towards test environments that are progressively dynamic and complex to mimic the challenges that current multi-agent domains raise and meet autonomy and collaboration expectations that today’s agent teams are expected to have in real life scenarios, [Stone et al., 2010] bring forward the notion of Ad Hoc Autonomous Agent Teams: heterogeneous teams of agents that may have different
acting and sensory abilities and world models, are not even necessarily programmed by the same people and may not share communication protocols, thus shattering any possibility of deploying team strategies a priori. The authors stress the importance of this topic with an example where a team of agents, unknown to each other and programmed by different people, is deployed in an unknown scenario to perform a rescue mission. The authors also use the concept of Ad Hoc human teams to exemplify the situation:

Example 2. Consider a medical emergency scenario, where a biker has an accident and is unconscious. A group of people rush to the scene with the common goal of helping said biker and, although they all know which steps need to be taken in order to help him (check if he is still breathing, call the ambulance, find a nearby policeman), they do not speak the same language and can not coordinate explicitly.

A good ad hoc agent would be able to efficiently examine the group and gauge each member’s abilities: for instance, if someone is a doctor, he/she should be the one to provide the victim with first aid.

Note that, unlike [Bowling and McCracken (2005)](BowlingAndMcCracken2005)'s impromptu teams robot soccer scenario, there is no previously assembled team where only one member was replaced and the rest are still able to coordinate as before, neither is there a previously designed strategy. Their main challenge was to create a single autonomous agent without knowledge of its teammates and without explicit coordination protocols, as we have seen in [Grosz and Sidner (1990)](GroszAndSidner1990)'s SharedPlans, that is robust and reliable in the long term, and whose performance was analysed in two different approaches.

The first of which is the theoretical approach, illustrated by the authors using an instance of the k-armed bandit problem ([Robbins, 1952](Robbins1952)) known as the teacher and the learner: where two agents, a teacher and a learner who select arms (from a set of 3 with different payoffs) alternately, beginning with the teacher. The teacher’s goal is to maximize the expected sum of the payoffs received by the two agents and the following assumptions are made:

1. The payoff distributions of all arms are fully known to the teacher, but unknown to the learner.
2. The learner can only select from among the two arms with the lower expected payoffs.
3. The results of all actions are fully observable (to both agents).
4. The number of rounds (actions per agent) remaining is finite and known to the teacher.
5. The learner’s behavior is fixed and known: it acts greedily, always selecting the arm with the highest observed sample average so far. If there are any previously unseen arms, the learner selects one of them randomly.

([Stone et al., 2010](Stone2010))

The learner is immutable, unable to communicate with the teacher and no previous coordination
strategies are in place, so the teacher must decide whether to always pick the arm with the highest payoff or allow the learner to gather some more information by picking a different one in the first round.

The second, and most pertinent to our work, is the empirical approach to the robot soccer scenario. But this is not the typical RoboCup setting described before, in which the agent teams are built as a single unit with communication protocols and player distributing algorithms to achieve the desired soccer formation. In this scenario (a pick-up game), the players are not able to coordinate prior to the game and the ad hoc agent should be able to identify the team’s vulnerabilities and act accordingly, either by filling in a gap in the formation or behaving in a manner that lessens those frailties, regardless of who its teammates’ are. This is a very interesting approach that already encompasses the task and teammate identification steps, which will be used in our work.

In another work, Barrett et al. (2013) provided a new perspective over this problem, adding uncertainty in the form of limited knowledge of the ad hoc agent’s teammates. The authors assume the agent has previous experience in the domain but not with its current teammates, which have been created by third parties (externally-created teams).

The goal is for the agent to build a library of models of its past teammates and then select its actions using a sample-based planning algorithm instead of learning through standard approaches such as ad hoc learning (Zeng and Sycara, 1998) or using a decision theoretical model such as partially observable Markov Decision Processes (POMDPs) for learning purposes. In order to test this innovative approach, the authors chose a modified version of the pursuit domain (Stone and Veloso, 2000), where the world is a 20x20 square toroidal grid and the predators’ team consists of 4 agents trying to surround the prey (which moves randomly) on all sides, as fast as possible. Also, the following assumptions are made:

1. Every agent can observe the positions of all other agents.
2. At each time step, each agent can stay still or move in any of the four cardinal directions.
3. All agents pick their actions simultaneously.
4. Collisions are decided randomly.

Figure 3.4 illustrates a scenario in this particular pursuit domain version, where the prey is already surrounded by the predators and captured.

In the planning step, the ad hoc agent uses Upper Confidence bounds for Trees (UCT) to run simulations from the current state, with the possible effects of the agent’s actions, until the capture of the prey. This is justified by the branching factor of the problem. UCT, which is Monte Carlo Tree Search (MCTS) algorithm, has been shown to be effective in games with a high branching factor, such as Go (Gelly and Wang, 2006), so it should perform well in an environment with this number of agents.

Using the knowledge from previous observations of each state-action it predicts how long it will

---

2 in the pursuit domain, a set of agents (predators) tries to capture an agent (the prey) in minimal time.
3 in a toroidal grid, an object moving off one side of the grid will enter on the opposite side.
take to capture the prey, from the current state. In this process, the trade-off between exploration and exploitation is controlled by the UCT algorithm, which helps decide when the agent should explore (if it has a low confidence in its action) or exploit (otherwise).

We have seen how the agent uses the model for planning, however, unlike similar works, the model is not known a priori and needs to be learned. This learning process provides the agent with several models based on the agent’s previous observations, which, when the agent is deployed, are treated as experience. Headed towards improving the performance of the agent, the authors also introduce a new transfer learning (TL) algorithm⁴, TwoStageTransfer, that allows learning from multiple source data sets.

Note that this is a challenging testing environment, the breadth of techniques applied here is wide and its results must be taken into consideration when trying to escalate to an even more complex domain.

3.3.1 Apprenticeship learning

A great way of learning is to watch someone else preform the task you aim to learn. That is also true as far as autonomous agents are concerned. Abbeel [2008] contributed with a ground breaking approach to the apprenticeship learning field, where an agent has access to demonstrations of a task being performed by an expert agent and learns how to do it, that way. The author uses the expert demonstrations to get a description of the task at hand in the form of a reward function.

The author models its problem in a Markov Decision Process where the reward function is not explicit and instead an expert is observed while performing the task. The expert is assumed to be trying to maximize a reward function that is a linear combination of its features. This is tackled using an algorithm

---
⁴transfer learning algorithms use information obtained in a source task set to improve learning in a target task.
that retrieves an unknown reward function from the observed behavior. This problem is commonly referred to as inverse reinforcement learning. Solving an MDP means finding an optimal policy, a function $\pi(s)$ that returns the action, $a$, that an agent should perform when in state $s$. The size of the MDP grows exponentially: $|A|^{|S|}$.

This algorithm is proven very reliable and the author demonstrates the learning agent achieves a performance comparable to that of the expert agent it observes, which, by mitigating the difficulty that is inherent to the specification of a reward function, eases the learning process.

This approach is particularly useful to our work since we need to determine the task at hand, and this is a good way of achieving that purpose.

### 3.3.2 Agents’ modelling

Modelling both your teammates and your opponents plays a crucial role in the RoboCup Simulation League (a particular ad hoc teamwork setting). Pourmehr and Dadkhah (2012) performed an analysis on the several existing approaches to this problem.

The authors identify opponent modelling as one of the major aspects for generating a competitive team and list a set of possible approaches to it. Among those, we underline the following:

Firstly, one can classify the current opponents into one of several predefined models (Riley and Veloso, 2000). This approach consists in:

1. Feature identification - identifying the other agent’s attributes from observation.
2. Model construction - building a model for the agent with the data gathered in the first step.
3. Classification to the predefined models - matching the model obtained in the second step with one of a list of models previously established.

Secondly, instead of classifying the agent itself, Riley and Veloso (2001) propose an approach that maps the opponent’s behavior to a set of known, pre-designed behaviors, using decision trees. This information is then used to select adaptive plays based on the opponent’s recent behaviors, online.

Another work, from Lattner et al. (2006) uses association rule mining to predict other agents’ behavior by applying a sequential pattern mining algorithm that extracts frequent patterns in data that describes the agent’s behavior. Then, these patterns are used to create rules that are enforced when trying to predict what will happen in the future. Although this is a powerful technique, the high complexity of its learning algorithm makes it unpractical.

Although there is no correct answer for the modelling of other agents, the one that makes more sense to use, for its simplicity and adequacy to the problem, is the first mentioned in this section: a classifier that maps the model built from observing the other agents’ behaviour to one of several pre-defined behaviors.

---

5Opponent modelling in the RoboCup Simulation League consists in identifying or predicting other agents’ behavior or teams’ formations, tactics, etc.
3.3.3 Leading teams of agents

In a slightly different but somewhat related topic, Agmon and Stone (2012) try to answer one of the challenges presented by Stone et al. (2010), leading teams of agents (by influencing the teammate’s behavior) that have no previous knowledge of each other and no explicit communication, to reach the optimal possible joint-utility.

Unlike Bowling and McCracken (2005), who suggested an adaptive and a predictive approach to insert a single agent in an already existing team, the authors focus on how an agent can influence the behavior of others, pursuing an optimal behaviour.

In this work, the authors consider that the already existing agents pick their next action based on the knowledge that they gather observing the former actions of their teammates (i.e. they are best response agents). They consider the case of one ad hoc agent leading a small team of two best response agents, a single agent leading an N-sized team and even the case where a team of ad hoc agents leads another team formed by best response agents. In the first case, the problem is represented in a graphical manner: the graph $G = < V, E >$, where $V$ is a set of all possible joint actions and $E$ is the set of edges, each representing the possibility to move from a joint action to another. Then, they find the optimal reachable steady cycle and compute the shortest path to reach it using Dijkstra’s algorithm. If the edges that close a cycle are not added, the graph becomes a directed acyclic graph (DAG) and the shortest path can be found in $O(|E| + |V|)$.

The solution for the case where an ad hoc agent leads a team of $N$ best response agents (where $N \geq 3$) is analogous to the previous one (where $N = 2$), however in this particular instance they reach the following corollary:

\begin{quote}
In a team of $N$ agents, for every $N \geq 3$, where at least two team members are best response agents, $m^*$ may remain unreachable.
\end{quote}

Agmon and Stone (2012)

which attests to the growing difficulty of reaching an optimal joint utility, as the number of ad hoc agents increases. Furthermore, if you add in the uncertainty of not having the knowledge about how the teammates will react, the difficulty of successfully solving this problem becomes evident, if solvable at all.

Genter et al (2013) specify a new scenario for studying this issue, the flocking problem. Flocking is a naturally occurring emergent behaviour, the global consequence of local interactions of individuals in the system’s population, that can be observed in swarms of bees, flocks of birds and other species’ populations. In the agent domain, this manifests itself when, a team of agents, each displaying simple local behaviours, results in a group behaviour or formation, without explicitly programed to do so.

The main focus of this work was to determine the most efficient way of using one or more agents to lead a team by influencing it to head towards a given direction, without directly manipulating their behaviour.
The aforementioned was tested in a combination of settings, with the leading agents being stationary at first and then moving. Plus, the authors tested the flocking behavior towards the ad hoc agent(s) or towards their visibility cone (Figure 3.5) and the latter works best, which is comprehensible, since the agent moving towards the visibility cone begins being influenced sooner (when it enters the cone).

Figure 3.5: Example of a flock of two agents in the same position \((a_4, a_5)\), led by four ad hoc agents \((a_0, a_1, a_2, a_3)\), adapted from Genter et al. (2013). The diagonal lines represent the visibility field/cone of the agents in the flock.

### 3.4 The ad hoc teamwork problem in RoboCup

Shifting to the RoboCup domain, a new RoboCup league, which focuses on teamwork without pre-coordination, was created in 2013 as an optional technical challenge (MacAlpine et al., 2014). In the following year it was upgraded to an official league, SPL Drop-in Player competition and the number of participants rose significantly. Genter et al. (2015) performed an analysis on this competition at RoboCup 2014.

SPL (Standard Platform League) is one of the leagues within the Robot Soccer division at RoboCup. In SPL all teams must use a uniform robotic platform, NAO robots\(^6\), to compete in 5 on 5 soccer matches. Figure 3.6 depicts two teams of NAO robots playing against each other at a competitive SPL match, in the German Open.

By analyzing the drop-in competition’s score schemes, namely the judge score, which is the score given to a team, by a human, based on several performance criteria, the authors conclude that the key indicators of the team’s proper cooperation are the pass performance and how often they push/bump into their own teammates. As for the player’s strategy, a great majority of the teams went with the following:

*If the ball is close and no other robot wants to play the ball, then play the ball. Otherwise take supporting position.*

\((\text{Genter et al., 2015})\)

\(^6\)NAO is an interactive humanoid robot which was originally created for educational purposes and is now used in RoboCup’s standard platform league (SPL).
Figure 3.6: Two teams of NAO robots playing in an SPL game at the 2012 German Open in Magdeburg, Germany.

The above is not easy to ensure since the decision to play the ball depends on the teammates current positions and intentions and some robots can not communicate with their teammates, making it impossible to determine their intentions, at a given time. However, this is a simple but effective strategy, since the teams’ performance was quite good, as can be perceived by the close result of a match between the overall Standard Platform League champion and a team formed by the best 5 ad hoc teammates from the Drop-in Player Challenge (4-2 in favour of the previously assembled team). This analysis of the SPL Drop-in Player Challenge yields a valuable insight regarding the key indicators of a team’s teamwork performance, which we will use when asserting the validity of our work.
Chapter 4

Multi-layer approach to Ad Hoc Teamwork

In this chapter we will provide a description of our solution to the problem described in the beginning of the document.

4.1 Architecture

First of all, we start by defining the global strategy we adopted, on a macro level, when facing the ad hoc teamwork problem. Melo and Sardinha (2016) presented a 3-step approach to ad hoc teamwork which encompasses task identification, teammate identification and planning. As Melo and Sardinha (2016) pertinently pointed out, most of the currently existing work on the ad hoc teamwork problem focuses mostly on the planning step (Stone et al., 2013) (Agmon and Stone 2012), however planning alone is not enough to solve our problem, since both teammates and task are unknown to our ad hoc agent and that information is essential for planning, so we need to retrieve it first.

We propose a new approach to ad hoc teamwork, based on Melo and Sardinha (2016)’s 3-step approach. A very simple view of our approach is depicted in Figure 4.1.

Figure 4.1: The three steps of our approach to ad hoc teamwork.

25
Although the approach can logically be segmented in these three challenges, none of them is independent from the other two and, more often then not, some parts of one challenge must be figured out before we are able to complete another of them.

The first challenge is *teammate identification*, which consists in building a model of the ad hoc agent’s teammates based on stored observations of their past behavior, in order to predict their future behavior. As far as the implementation goes, we use a *teammate classifier* that maps the model created using the observed agents’ behavior to the model that best fits its description in a set of previously established standard behaviors, as we have seen in Section 3.3.2. In this stage, the first thing we do is feature extraction (where we gather the necessary features from the data that results from the observation of the other agents), then model construction (where we build a model for those agents based on those features that describes the agents’ behavior) and classification (where we match the generated model to one of those that we have created previously and stored in our behavior models’ library).

Note that if we classify the remaining agents properly, the *task identification* process becomes simpler.

The second challenge is *task identification*. It consists in determining and recognizing the task at hand which is not usually considered in this kind of problem, but we believe that it contributes to a more robust agent capable of adaptation to different task-oriented domains.

For the purpose of this task we modeled an MDP where the reward function, \( R : S \times A \times S \rightarrow \mathbb{R} \), is unknown, as we have seen in Section 3.3.1. The other agents are regarded as expert agents in the domain and the reward function is obtained by applying inverse reinforcement learning, using the observations of the experts’ behavior. In order to increase the efficiency of this process and obtain the best reward function possible we use the results from the first step (*teammate identification*) to help us identify the best agents to use as expert(s). We then use a task classifier that matches the observed task to one of the tasks in a library of pre-established tasks.

The third, and final challenge is *planning*. At this stage, once we have a reward function for our MDP, the planning step consists in solving the MDP. It may seem straightforward enough, but remember the MDP grows exponentially: \(|A|^{|S|}\) and our domain is a complex one. To solve this, as we are dealing with a large set of possible policies, we implemented a Policy Iteration algorithm (and corresponding Policy Evaluation algorithm), a dynamic programming algorithm that iterates over a policy, generating a sequence of monotonically improving policies \((\pi(s))\) and value functions \((V^\pi)\) iteration after iteration, until the optimal policy is reached. A more detailed explanation on this topic can be found on Section 2.2.1.

A simplistic diagram of our approach overview is depicted in Figure 4.2 (you can find an enlarged version of this diagram on Appendix A).
4.1.1 Domain Specific Architecture for RoboCup 2D Simulation League

In the environment in which we are going to be testing this particular approach, the teammate identification process consists in building and classifying the models of the ten remaining players of our team. If we can properly identify each of the ten remaining teammates’ roles, we can feed that information to the task classifier, that will build a task model based on in and try to match it to one of the known task models in its task library.

In Robocup 2D Simulation League, a formation consists in a set of eleven positions and eleven roles on the field and there are only four types of roles (Goalkeeper, Defender, Midfielder and Forward). In our case, the task is the team formation in which our ad hoc team is going to play, as well as the position of our ad hoc agent on the field (defender, midfielder or forward) and the behavior it should adopt in order to increase its efficacy in the role it is assigned in the team.

For the task identification step, the task classifier includes a formation classifier that matches the observed ad hoc formation (the formation of the ad hoc team) to one of the formations in a library of pre-established formations.

A simplistic diagram of our domain specific approach overview is depicted in Figure 4.3 (you can find an enlarged version of this diagram on Appendix B).
4.2 Team construction and Domain information

To be able to test our solution, we first had to create our own team(s) of agents (Robocup Soccer Server [RCSS] clients). For simplicity, we are simply going to call these teams of agents, RCSS teams. We could have used other existing teams but most of the available binaries of other teams are not accompanied with a detailed description of the agents’ architectures and we wanted to be able to have full knowledge about our testing environment.

Building a team of RCSS clients is a great challenge that can be decomposed into five main steps:

1. Understanding the RCSS communication protocols and successfully establishing communication with the server via UDP sockets.
2. Parsing RCSS server messages with sensory inputs and interpreting sensory data.
3. Defining team formations.
4. Segmenting a team into field sectors, according to the roles each of the agents plays on the team.
5. Implementing different subsumption architectures (Brooks [1986]) for the agents with different assigned roles.

Since our work’s focus is not the process of building the team per se, we procured a simple RCSS client, Krislet [Floyd and Esfandiari, 2009], which greatly eased the implementation of steps 1 and 2 of the previous list. Although the communication stopped being such a big issue, the Krislet agent has a fairly limited behavior that is described on code listing 4.1 below.
We wanted our agents to have much more complex behavior and also to behave differently depending on the roles assigned to them. With that in mind we created a set of different formations (which we named “Playbook”). A formation is a set of agents’ descriptions that contain the agent's initial position on the field (X, Y) and its role (Goalkeeper, Defender, Midfielder or Forward). The agent description is passed on to the agent when it is created and the formation emerges from all of the agents in each team acting according to their individual description. A more detailed description of our formations can be found on Table 4.1.

<table>
<thead>
<tr>
<th>Formation Name</th>
<th>Defenders</th>
<th>Midfielders</th>
<th>Forwards</th>
<th>Formation Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-4-3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>attack-oriented</td>
</tr>
<tr>
<td>3-6-1</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>attack-oriented</td>
</tr>
<tr>
<td>4-2-4</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>attack-oriented</td>
</tr>
<tr>
<td>2-6-2</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>balanced</td>
</tr>
<tr>
<td>5-4-1</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>defense-oriented</td>
</tr>
<tr>
<td>6-3-1</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>defense-oriented</td>
</tr>
</tbody>
</table>

Table 4.1: Our playbook’s set of formations

These individual agent behaviors were defined as subsumption architectures (Brooks, 1986) based on some previous knowledge about real football players’ behaviors on a number of frequent situations encountered on a real football match. To achieve these architectures first we had to define a number of states in which the agents may find themselves and a number of compound actions available to them when in a given state. Each state is defined as a set of conditions that describe a common situation on the field, as can be observed on Table 4.2 (note that the states never overlap).

All agents share similar behaviors when the ball is far away or within reach, but not near its feet. If the ball is not visible or very far from the agent, it should turn at an angle and observe its surroundings until the situation changes (the forwards are an exception, because they run towards the ball even if it is very far from them). When the ball in neither far nor near, but it is within its reach, the agent must check if a teammate is closer to it. If that is the case, the agent should stay back to avoid swarming. Otherwise, it should run to it.

1 visibility of an object, in this context, means being able to perceive its location using the sensors available to it.
2 the agent’s reach is a constant that defines the maximum distance at which the ball should be pursued by the agent.
<table>
<thead>
<tr>
<th>State</th>
<th>Ball Position</th>
<th>Opponent Goal</th>
<th>In Shooting Range</th>
<th>Teammate Near</th>
<th>Teammate Closer</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₀</td>
<td>Can’t see ball</td>
<td>Not relevant</td>
<td>Not relevant</td>
<td>Not relevant</td>
<td>Not relevant</td>
</tr>
<tr>
<td>s₁</td>
<td>Far away</td>
<td>Visible</td>
<td>True</td>
<td>Not relevant</td>
<td>Not relevant</td>
</tr>
<tr>
<td>s₂</td>
<td>Far away</td>
<td>Visible</td>
<td>False</td>
<td>Not relevant</td>
<td>Not relevant</td>
</tr>
<tr>
<td>s₃</td>
<td>Far away</td>
<td>Not visible</td>
<td>Not relevant</td>
<td>Not relevant</td>
<td>Not relevant</td>
</tr>
<tr>
<td>s₄</td>
<td>Within range</td>
<td>Not relevant</td>
<td>Not relevant</td>
<td>Not relevant</td>
<td>True</td>
</tr>
<tr>
<td>s₅</td>
<td>Within range</td>
<td>Not relevant</td>
<td>Not relevant</td>
<td>Not relevant</td>
<td>False</td>
</tr>
<tr>
<td>s₆</td>
<td>Very close</td>
<td>Not visible</td>
<td>Not relevant</td>
<td>False</td>
<td>Not relevant</td>
</tr>
<tr>
<td>s₇</td>
<td>Very close</td>
<td>Not visible</td>
<td>Not relevant</td>
<td>True</td>
<td>Not relevant</td>
</tr>
<tr>
<td>s₈</td>
<td>Very close</td>
<td>Visible</td>
<td>False</td>
<td>True</td>
<td>Not relevant</td>
</tr>
<tr>
<td>s₉</td>
<td>Very close</td>
<td>Visible</td>
<td>False</td>
<td>False</td>
<td>Not relevant</td>
</tr>
<tr>
<td>s₁₀</td>
<td>Very close</td>
<td>Visible</td>
<td>True</td>
<td>Not relevant</td>
<td>Not relevant</td>
</tr>
</tbody>
</table>

Table 4.2: States’ definition based on agent’s observations

*Swarming* is an undesirable collective behavior in all Robocup Soccer competitions, in which a large group of agents from the same team end up chasing the ball at the same time (causing the agents to clash together in a kind of swarm), instead of maintaining the formation positions assigned to them and offering passing alternatives to the agent that actually holds the ball (see Figure 4.4). The *swarming* avoidance behavior was implemented in all of our agents and it was a large contributor to the organized team behavior that our teams display. It is manifested in the form of the action *Stay Back* when the ball is in range but a teammate is closer to it (check the agents’ actions on state $s_4$ on Table 4.3).

![Swarming behavior example](image)

Figure 4.4: An example of a situation in which the agents are displaying swarming behavior. Notice how several agents from the yellow and the blue team are gathering near the ball.
In our Robocup 2D Simulation League domain, the agent (player) may fit in one of four predetermined roles in the field: goalkeeper, defender, midfielder or forward.

The **Goalkeeper** is a special kind of agent whose job is simply to prevent the ball from entering its goal. Therefore its behavior consists in kicking the ball away from its goal when the ball is near and keep its position when the ball is far.

The **Defenders’** job is to clear the ball from the surroundings of their goal, when they are near it. Their preference is to pass the ball to a teammate that is far from them, if there is a visible one, otherwise they should kick it forward indiscriminately to prevent the opponents from stealing the ball and scoring a goal.

The **Midfielders** have a mixed behavior with some defensive and some offensive traits. Similarly to the defenders, the midfielders prefer to pass the ball upfield when they have a teammate in sight. However, unlike the defenders (and much like the forwards), the midfielders dribble the ball upfield if they do not have a free teammate to pass the ball to and shoot at goal once they are in shooting range.

The **Forwards** are the agents responsible for most of the goal scoring chances in the team. They do not have defensive-minded behavior which translates to a disregard in keeping their position and a frequent chase of the ball. Whenever a forward has the ball and has the opponent goal in sight, it dribbles towards it until it is close enough to shoot and then shoots.

A summary of the agents’ behaviors depending on their current state and on their role on the field (i.e.: their policy) is summarized in Table 4.3. The actions available to the agent are listed on Table 4.4.

<table>
<thead>
<tr>
<th>State ID</th>
<th>Goalkeeper Action</th>
<th>Defender Action</th>
<th>Midfielder Action</th>
<th>Forward Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>Look Around</td>
<td>Look Around</td>
<td>Look Around</td>
<td>Look Around</td>
</tr>
<tr>
<td>$s_1$</td>
<td>Look Around</td>
<td>Look Around</td>
<td>Look Around</td>
<td>Run To Ball</td>
</tr>
<tr>
<td>$s_2$</td>
<td>Look Around</td>
<td>Look Around</td>
<td>Look Around</td>
<td>Run To Ball</td>
</tr>
<tr>
<td>$s_3$</td>
<td>Look Around</td>
<td>Look Around</td>
<td>Look Around</td>
<td>Run To Ball</td>
</tr>
<tr>
<td>$s_4$</td>
<td>Stay Back</td>
<td>Stay Back</td>
<td>Stay Back</td>
<td>Stay Back</td>
</tr>
<tr>
<td>$s_5$</td>
<td>Run To Ball</td>
<td>Run To Ball</td>
<td>Run To Ball</td>
<td>Run To Ball</td>
</tr>
<tr>
<td>$s_6$</td>
<td>Kick Ball Away</td>
<td>Look Around</td>
<td>Look Around</td>
<td>Look Around</td>
</tr>
<tr>
<td>$s_7$</td>
<td>Kick Ball Away</td>
<td>Pass To Teammate</td>
<td>Pass To Teammate</td>
<td>Look Around</td>
</tr>
<tr>
<td>$s_8$</td>
<td>Kick Ball Away</td>
<td>Pass To Teammate</td>
<td>Pass To Teammate</td>
<td>Dribble To Goal</td>
</tr>
<tr>
<td>$s_9$</td>
<td>Kick Ball Away</td>
<td>Kick Ball Forward</td>
<td>Dribble To Goal</td>
<td>Dribble To Goal</td>
</tr>
<tr>
<td>$s_{10}$</td>
<td>Kick Ball Away</td>
<td>Kick Ball Forward</td>
<td>Kick To Goal</td>
<td>Kick To Goal</td>
</tr>
</tbody>
</table>

Table 4.3: Players’ policies by role
4.3 Teammate identification

The teammate identification step of our approach consists in building a model for each of the ad hoc agent’s teammates and match it to a set of previously established ones, to determine the role attributed to each of them in the ad hoc team.

4.3.1 Player classifier

The feature extraction stage of the classification consists in obtaining the (offline) observations of our ad hoc teammates. An observation, \( o \), from agent \( ag \) is defined as a pair \( o_{ag} = (s_x, a_x) \), which represents that agent \( ag \) performed action \( a_x \) when it was on state \( s_x \).

We obtain a set of observations, \( O \), from each agent (teammate) \( ag \), \( O_{ag} = \{ (s_1, a_1), (s_2, a_2), \ldots (s_k, a_k) \} \) and, based on those, we build a model for each of them that describes their behavior in terms of which action they are expected to perform in each of the states \( s \in S \). Afterwards, a set of predictors, one for each role, generates a score for that agent model in that role. The higher the score, the most likely it is that the agent is, in fact, performing that role.

Then, an algorithm uses those scores to produce a classification of the agent that translates into a role. The process bares some resemblance with the Weight Majority Algorithm described by Littlestone and Warmuth (1994), but simpler.

We used two types of teammate models, best predictor model and mixed predictor model. In a best predictor model, an agent is classified as either a defender, midfielder, or forward, as opposed to the mixed predictor model where an agent may be classified as 60% defender and 40% midfielder, for instance.

The classification can be achieved either by using the best predictor approach or the mixed predictor approach. We tested both variants. In the best predictor approach, the classifier simply selects the role with the highest predictor’s score. In the mixed predictor approach, the classifier returns a membership degree\(^4\) the value returned by the membership function, \( m \), for each of the roles. The membership degree for each role is computed dividing the predictor’s score for that role by the sum of the predictor

\(^4\)The membership degree is a value \([0, 1]\) that measures the “truth” of saying that an element belongs to a set (in our case this value represents the degree of true of saying that an agent is performing a given role).
scores. For example, computing the membership of the defender role consists in the following:

\[ m_{\text{Defender}} = \frac{\text{SCORE}_{\text{Defender}}}{\sum_{\text{role}} \text{SCORE}_{\text{role}}} \]  

(4.1)

The mixed predictor classification result is a tuple:

\[ Cl_{\text{mixed-predictor}} = (m(\text{Defender}), m(\text{Midfielder}), m(\text{Forward})) \]  

(4.2)

An example of a best predictor and a mixed predictor classification of a set of teammates can be found on Table 4.5 and Table 4.6 respectively.

<table>
<thead>
<tr>
<th>Agent</th>
<th>Defender Score</th>
<th>Midfielder Score</th>
<th>Forward Score</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_1)</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>Defender</td>
</tr>
<tr>
<td>(a_2)</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>Defender</td>
</tr>
<tr>
<td>(a_3)</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>Defender</td>
</tr>
<tr>
<td>(a_4)</td>
<td>4</td>
<td>7</td>
<td>1</td>
<td>Midfielder</td>
</tr>
<tr>
<td>(a_5)</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>Midfielder</td>
</tr>
<tr>
<td>(a_6)</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>Midfielder</td>
</tr>
<tr>
<td>(a_7)</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>Midfielder</td>
</tr>
<tr>
<td>(a_8)</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>Midfielder</td>
</tr>
<tr>
<td>(a_9)</td>
<td>0</td>
<td>5</td>
<td>8</td>
<td>Forward</td>
</tr>
</tbody>
</table>

Table 4.5: A teammate’s best predictor classification example.

<table>
<thead>
<tr>
<th>Agent</th>
<th>Defender Score</th>
<th>Midfielder Score</th>
<th>Forward Score</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_1)</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>(\left(\frac{7}{12}, \frac{7}{12}, \frac{7}{12}\right))</td>
</tr>
<tr>
<td>(a_2)</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>(\left(\frac{6}{13}, \frac{6}{13}, \frac{6}{13}\right))</td>
</tr>
<tr>
<td>(a_3)</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>(\left(\frac{7}{12}, \frac{7}{12}, \frac{7}{12}\right))</td>
</tr>
<tr>
<td>(a_4)</td>
<td>4</td>
<td>7</td>
<td>1</td>
<td>(\left(\frac{4}{12}, \frac{4}{12}, \frac{4}{12}\right))</td>
</tr>
<tr>
<td>(a_5)</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>(\left(\frac{3}{12}, \frac{3}{12}, \frac{3}{12}\right))</td>
</tr>
<tr>
<td>(a_6)</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>(\left(\frac{1}{12}, \frac{1}{12}, \frac{1}{12}\right))</td>
</tr>
<tr>
<td>(a_7)</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>(\left(\frac{2}{11}, \frac{2}{11}, \frac{2}{11}\right))</td>
</tr>
<tr>
<td>(a_8)</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>(\left(\frac{3}{11}, \frac{3}{11}, \frac{3}{11}\right))</td>
</tr>
<tr>
<td>(a_9)</td>
<td>0</td>
<td>5</td>
<td>8</td>
<td>(\left(\frac{0}{11}, \frac{0}{11}, \frac{0}{11}\right))</td>
</tr>
</tbody>
</table>

Table 4.6: A teammate’s mixed predictor classification example.
On appendix C you can find a diagram that sums up the player classification mechanism we use.

4.4 Task identification

In this section we will describe the task identification step of our solution, in which our ad hoc agent figures out the formation in which our ad hoc team is going to play, as well as the position and role it is supposed to occupy on the field (defender, midfielder or forward) and the corresponding behavior it should adopt. We must underline that, due to the difficulty of obtaining viable observations of the other agents’ behavior and identifying the task online (while the game in ongoing), our observations are collected from past games and fed to our ad hoc agent afterwards offline. Therefore, the agent completes its learning process before the game starts and does not improve its performance while the game is underway.

4.4.1 MDP modelling

In order to identify the task at hand, we regard our ad hoc agent as a decision maker in a dynamic environment that can be described using a Markov Decision Process (refer to Section 2.2 for a detailed definition). For that purpose, we need to define the elements of the tuple \((S, A, T, \gamma, R)\) that make up the MDP, as defined by Ramachandran and Amir (2007). The set of states, \(S\), and the set of actions, \(A\), match the states and actions described on Section 4.2.

As we are dealing with a stochastic environment\[^5\] the transition function, \(T: S \times A \times S \to [0, 1]\) needs to be defined, since the probability of transition from one state to another given an action is not always 1. It is a map \(<\text{Key}, \text{Value}>\) whose keys are \(\text{state}_{current}, \text{action}_a\) and \(\text{state}_{next}\) and the value is the probability that \(\text{action}_a\) performed on \(\text{state}_{current}\) leads the agent to \(\text{state}_{next}\).

The discount factor \(\gamma\), which represents the difference in importance between future rewards and present rewards is bounded \((0 \leq \gamma < 1)\). Under this criterion, future rewards are worth less than the current reward. If \(\gamma = 1\), this would be the same as the total reward. When \(\gamma = 0\), the agent ignores all future rewards. Having \(0 \leq \gamma < 1\) guarantees that, whenever the rewards are finite, the total discounted value of a policy \(\pi\), \(V^\pi\), will also be finite. We experimented with several values for \(\gamma\) but it ended up being set to \(\gamma = 0.9\).

As we summarily described in Section 4.1, the reward function, \(R: S \to \mathbb{R}\), is initially unknown. To obtain it we use inverse reinforcement learning, which we covered in our background (Section 2.2.2) and is further described in Section 4.4.3.

4.4.2 Formation classifier

An important step in the process of identifying the task is identifying the formation in which it is playing. This is achieved through a classification mechanism that matches the observed ad hoc formation\[^6\] to a

---

\[^5\] a stochastic environment is unpredictable, meaning that the same action \(\text{action}_a\) performed on \(\text{state}_{current}\) may lead to more than one state, as opposed to a deterministic environment.

\[^6\] an ad hoc formation is any formation that includes at least one ad hoc agent.
set of formation models in our formation library.

Our formation library is a set of formation models. Each formation model is an object that describes a formation, namely the number of players that constitute it and the number of players in each of the roles (defender, midfielder or forward; the goalkeeper is present and unique in each formation and thus irrelevant to the classification mechanism). The formation library includes a model for each of the formations listed in Table 4.1. An example of an ad hoc formation (in this particular case, a 3-4-3 formation) can be found in Figure 4.5.

To be able to classify our ad hoc formation, first we need to model it, the same way we modelled the formations in our library. In order to obtain the number of players assigned to each role, we need to classify our ad hoc agent’s 9 teammates that play a field role (any role besides goalkeeper). This is achieved by the teammate classification process described in Section 4.3 and corresponds to the feature extraction and model construction steps of the classification.

In the best predictor approach to teammate classification (Section 4.3), the formation classifier is given as input (beside the formation library) the teammates’ best predictor models. In the mixed predictor approach it is given as input the teammates’ mixed predictor models, instead.

Independently of the approach used for teammate modelling, we compute a ratio for each player role \(d_{\text{Ratio}}, m_{\text{Ratio}}, f_{\text{Ratio}} \in \mathbb{R}^+\), for defenders, midfielders and forwards, respectively) between the number of observed players in that role in the ad hoc formation and the real the number of players in each role in each formation, as described below.

Given a formation \(F\) and an ad hoc formation \(F_{\text{AdHoc}}\):

\[
d_{\text{Ratio}}(F, F_{\text{AdHoc}}) = \frac{\min(\#\text{defenders}(F), \#\text{defenders}(F_{\text{AdHoc}}))}{\max(\#\text{defenders}(F), \#\text{defenders}(F_{\text{AdHoc}}))} \tag{4.3}
\]

\[
m_{\text{Ratio}}(F, F_{\text{AdHoc}}) = \frac{\min(\#\text{midfielders}(F), \#\text{midfielders}(F_{\text{AdHoc}}))}{\max(\#\text{midfielders}(F), \#\text{midfielders}(F_{\text{AdHoc}}))} \tag{4.4}
\]
The similarity, $\text{Sim}(F, F')$, is then computed as a product of those ratios, for every formation, $F_i$ in the ad hoc agent's playbook, as follows:

$$\text{Sim}(F_i, F_{\text{AdHoc}}) = d_{\text{Ratio}}(F_i, F_{\text{AdHoc}}) \times m_{\text{Ratio}}(F_i, F_{\text{AdHoc}}) \times f_{\text{Ratio}}(F_i, F_{\text{AdHoc}})$$ (4.6)

The formation with the highest similarity measure, $F^*$, is then returned (an example of the classifier’s procedure is below). To find the missing role in the selected formation, we simply subtract the Defenders of the ad hoc formation, $F_{\text{AdHoc}}$, to $F^*$ and repeat the process for Midfielders and Forwards. The remainder of one of those operations is going to be 1 (the missing role) and the other two are going to be 0.

**Example 3.** Consider, for instance, an ad hoc formation, $F_{\text{AdHoc}}$ and a small formations library with two formations (3-5-2 → $F_1$ and 5-3-2 → $F_2$).

1. Our best predictor teammate classifier returns 9 teammate models from the observations of our ad hoc team's players, 3 Defenders, 4 Midfielders and 2 Forwards.
2. A formation model is built based on the teammates’ classifications and is fed to the classifier.
3. The classifier computes the ratios and similarity measures for all the formations in the library ($F_1$ and $F_2$):

   $d_{\text{Ratio}}(F_1, F_{\text{AdHoc}}) = \frac{3}{3} = 1.0$
   $m_{\text{Ratio}}(F_1, F_{\text{AdHoc}}) = \frac{4}{5} = 0.8$
   $f_{\text{Ratio}}(F_1, F_{\text{AdHoc}}) = \frac{2}{2} = 1.0$
   $\text{Sim}(F_1, F_{\text{AdHoc}}) = 1.0 \times 0.8 \times 1.0 = 0.8$

   $d_{\text{Ratio}}(F_2, F_{\text{AdHoc}}) = \frac{3}{5} = 0.6$
   $m_{\text{Ratio}}(F_2, F_{\text{AdHoc}}) = \frac{4}{5} = 0.6$
   $f_{\text{Ratio}}(F_2, F_{\text{AdHoc}}) = \frac{2}{2} = 1.0$
   $\text{Sim}(F_2, F_{\text{AdHoc}}) = 0.6 \times 0.6 \times 1.0 = 0.36$

   $\text{Sim}(F_1, F_{\text{AdHoc}}) > \text{Sim}(F_2, F_{\text{AdHoc}})$

4. The classifier returns $F_1$.

**4.4.3 Obtaining the reward function via IRL**

**Selecting an expert**

The other players are regarded as expert agents in the domain and the reward function is obtained using the observations of the experts’ behavior. To improve our chance of obtaining a suitable reward function
for our agent, we simplified the issue by selecting one particular agent to be regarded as the expert agent whose unknown reward function we are trying to discover.

The process of selecting the expert agent to use depends upon a successful classification of the formation in which the ad hoc team is playing (this process is defined in Section 4.4.2), which allows us to deduce which position is missing in said formation by subtracting the observed agent roles, from the remaining team members, to those that are listed on the formation description. To identify these players’ roles, we use the process described in Section 4.3. Once we identify the position that is missing and which role the agent in that position is supposed to perform, we select one of the agents that performs that role in our ad hoc team and use it as an expert. For instance, if we identify our ad hoc formation as a 4-3-3 (4 defenders, 3 midfielders, 3 forwards) and we can observe 4 defenders, 3 midfielders but only 2 forwards in our team, our agent should probably perform the missing forward role and we are going to select a teammate that has been classified as a forward as our expert.

This selection process is an optimization to the learning process we used, we could instead average the behaviors of all observed agents, but then our agent would have an all round behavior, not “fitting” into a particular role description. In some other domains, this selection process might be more difficult or may even be unnecessary.

Bayesian IRL

IRL is currently viewed as a problem of inferring a single reward function that explains an agent’s behaviour. However, there is too little information in a typical IRL problem to get only one answer. A probability distribution is needed in order to represent the uncertainty (Ramachandran and Amir, 2007). Our IRL step’s solution is based on Ramachandran and Amir (2007)’s Bayesian IRL (BIRL), a model for IRL from a Bayesian perspective. The BIRL model is portrayed in Figure 4.6.

![Figure 4.6: The BIRL model, from Ramachandran and Amir (2007).](image)

Given an expert, $X$, operating on our MDP, we assume that a reward function, $R$, is chosen for expert $X$ from a prior distribution, $P_R$. We obtain a set of observations from $X$, $O_X = \{(s_1, a_1), (s_2, a_2), \ldots, (s_k, a_k)\}$ which means that $X$ was in state $s_i$ and took action $a_i$ at time step $i$. $\alpha_X$ represents the degree of confidence we have that $X$ will choose actions with high value. We make the following assumptions:
1. $X$ is attempting to maximize the total accumulated reward according to $R$. For example, $X$ is not using an epsilon greedy policy to explore his environment.

2. $X$ executes a stationary policy, i.e. it is invariant with respect to time and does not change depending on the actions and observations made in previous time steps. (Ramachandran and Amir, 2007)

Since $X$’s policy is assumed to be stationary we can assume that:

$$Pr_X(O_X|R) = Pr_X((s_1, a_1)|R)Pr_X((s_2, a_2)|R) \ldots Pr_X((s_k, a_k)|R) \tag{4.7}$$

We define the optimal Q-function, $Q^*(\ldots, R)$, as the Q-function of the optimal policy, $\pi^*$, for reward function $R$ (see Section 2.2). The expert’s goal is to maximize accumulated reward. This can be posed as finding the action for which $Q^*$ value at each state is at its peak. The larger the $Q^*(s, a)$, the more likely the expert would choose to perform action $a$ when in state $s$. [Ramachandran and Amir, 2007] model this as an exponential distribution for the likelihood of $(s_i, a_i)$ with $Q^*$ as a potential function, as follows:

$$Pr_X((s_i, a_i)|R) = \frac{1}{Z_i} e^{\alpha X Q^*(s_i, a_i, R)} \tag{4.8}$$

where $Z$ is a normalizing constant. Thus, extrapolating for the universe of expert observations, the likelihood is:

$$Pr_X((O_X)|R) = \frac{1}{Z} e^{\alpha X E(O_X, R)} \tag{4.9}$$

where $E(O_X, R) = \sum_i Q^*(s_i, a_i, R)$ and $O_X = \{(s_1, a_1), (s_2, a_2)\ldots(s_k, a_k)\}$ is a set of observations from $X$. If we apply Bayes’ theorem\footnote{Bayes’ theorem mathematical statement: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$} we can compute the posterior probability of the reward function, $R$, as follows:

$$Pr_X(R|O_X) = \frac{Pr_X((O_X)|R)Pr(R)}{Pr(O_X)} = \frac{1}{Z'} e^{E(O_X, R) Pr(R)} \tag{4.10}$$

where $Z'$ is a normalizing constant.

Next, as [Ramachandran and Amir, 2007] explain, it is required to compute the mean of the posterior distribution. Since computing the mean of the posterior distribution is a computationally hard process, we instead generate samples from the distributions and compute their mean, returning it as an estimate of the real mean. The sampling algorithm we use for that purpose is Policy Walk. An auxiliary algorithm,
Policy Iteration, is required for this task. Policy Walk, is portrayed in Algorithm 2. Policy Iteration is described in Section 2.2.1.

**Data:** Distribution $P$, MDP $M$, Step Size $\delta$, Number of Iterations $\max_{\text{iterations}}$

**Result:** Reward vector $R$

1. Pick a random reward vector $R \in \mathbb{R}^{\mid S \mid}$
2. $\pi = \text{PolicyIteration}(M, R)$
3. $n_{\text{iteration}} = 0$
4. while $n_{\text{iteration}} < \max_{\text{iterations}}$ do
   (a) Pick a reward vector $R'$ uniformly at random from the neighbors of $R \in \mathbb{R}^{\mid S \mid} / \delta$.
   (b) Compute $Q^\pi(s, a, R') \forall (s, a) \in S, A$.
   (c) if $\exists (s, a) \in (S, A), Q^\pi(s, \pi(s), R') < Q^\pi(s, a, R')$ then
      i. $\pi' = \text{PolicyIteration}(M, R', \pi)$
      ii. Set $R = R'$ and $\pi = \pi'$ with probability $\min(1, \frac{P(R', \pi')}{P(R, \pi)})$
   else
      i. Set $R = R'$ with probability $\min(1, \frac{P(R', \pi')}{P(R, \pi)})$
   end
   (d) $n_{\text{iteration}} += 1$
end

**Algorithm 2:** Policy Walk sampling algorithm, adapted from Ramachandran and Amir (2007).

The Policy Walk algorithm is a modified version of another Markov Chain Monte Carlo algorithm, Grid Walk (Vempala, 2005), that generates a Markov Chain (see Section 2.1). However, Policy Walk makes use of the auxiliary $Q$-function and is more efficient (Ramachandran and Amir, 2007), so it suits our purpose.

Having completed this process, we now have a reward function, $R : S \times A \times S \rightarrow \mathbb{R}$, for our MDP, and the task identification process is concluded.

### 4.5 Planning

The final challenge of successfully implementing ad hoc teamwork is planning. Once we have a reward function for our Markov Decision Process, the planning step consists in solving the MDP (finding an optimal policy $\pi(S)$ that for each state returns an action for the agent to perform). As we are dealing with a large set of possible policies, we implemented a Policy Iteration algorithm (and corresponding Policy Evaluation algorithm), which consists in continuously improving a policy, every iteration. This process is
thoroughly explained in Section 2.2 and Section 2.2.1.

With this explanation, we conclude our solution’s description and will now present the results we obtained, on the following chapter.
Chapter 5

Results

In this chapter we describe and discuss how we evaluated our work and the criteria we chose to do so, as well as the results we obtained. It presents an empirical evaluation of our proposed approach to ad hoc teamwork in the Robocup 2D Simulation League domain.

5.1 Evaluation Method

Since our approach can be subdivided into three steps, it is only logical that we evaluate each of those steps separately, first, and then as a whole. However, as explained before, the three steps of our implementation can not be explicitly divided as some of them influence others. This also means that the performance of one step will influence the performance of the remaining two steps, and vice-versa.

As far as the teammate identification step goes, its performance can be measured in terms of our agent’s success in identifying which roles its teammates are committed to. As described in Section 4.3, the task identification performance depends hugely upon the success (or failure) of this task.

Regarding the task identification step, its performance can be evaluated by determining our agent’s success rate when identifying the formation in which its team is playing as well as figuring out which role it should be playing in said formation.

When it comes to evaluating our planning step, due to difficulty of evaluating the obtained policy in an intelligible way, our approach was to evaluate the agent performance as a virtual soccer player in the team. Alongside that we also evaluated the performance of the ad hoc team, which the ad hoc agent influences.

First off, we needed to establish some kind of baseline, so we could compare that with the performance of our ad hoc team. Thus, we used a set of 2 RoboCup 2D Simulation League teams (created by us), selected one of them to be our opponent team and the other to be our host team (the team which would later have one of its agents replaced by ours). Then, we ran a set of matches between the host team and the other team, using all the team’s different formations, which, as described in Section 3.2 may have a great deal of influence in the team’s performance.

After having ascertained that, we are ready to replace a randomly selected agent from our host team
with our own ad hoc agent. Having done so, we test the ad hoc team against the same opponent team as before, in the same combinations of team formations.

As Genter et al. (2015) showed, we must analyze the match scores. However, that is insufficient, because the team can have a drop in other performance indicators and still score goals. So, using the expertise gathered from Section 3.4 we also studied our team’s passing accuracy \( \frac{\text{passes completed}}{\text{passes attempted}} \) and passing frequency, as a whole and individually (our agent’s). Furthermore, we analyzed how often the ad hoc agent bumps with its teammates, bumping frequency. If it occurs very frequently, it’s an indicator of bad coordination.

We can also analyze the ad hoc agent’s influence in the team by comparing its individual stats with the team’s, so we can determine if our agent is a good, bad or neutral influence to the team’s performance. The agent’s ability to dribble the ball successfully towards the opponent’s goal is also a good performance indicator. We consider a dribble successful if it is not intercepted by a rival player or if the ball does not cross the field’s limits after the player tries to dribble. A “smarter” agent is able to know when to dribble, to prevent the loss of the ball’s possession.

While we conduct these tests, we will be saving some performance indicators from both teams because, sometimes, in football, a good indicator that your team is performing well is the other team’s inability to perform (for instance, your team’s amount of interceptions will directly influence the other team’s ability to keep the ball and pass it around).

To sum up, here is a list of the criteria that can be used to measure the ad hoc agent and the ad hoc team’s performance, sorted from most to least important, and their meaning:

- **Match Score**: \( \frac{\text{scored goals}}{\text{conceded goals}} \)
- **Shot Accuracy**: \( \frac{\text{shots on target}}{\text{shots attempted}} \)
- **Shot Frequency**: \( \text{shots attempted} \)
- **Passing Accuracy**: \( \frac{\text{passes completed}}{\text{passes attempted}} \)
- **Passing Frequency**: \( \text{passes attempted} \)
- **Dribbling Accuracy**: \( \frac{\text{dribbles completed}}{\text{dribbles attempted}} \)
- **Bumping Frequency**: \( \text{number of bumps} \)
- **Opponent’s Passing Accuracy**: \( \frac{\text{passes completed}}{\text{passes attempted}} \)
- **Opponent’s Passing Frequency**: \( \text{passes attempted} \)

We ran two sets of trials for each individual step and for the in-game tests of our approach. The first set was ran with clear data, straight from the observations of the expert agent(s). The second set was ran with noisy data, in which we randomly replaced \( \sim 10\% \) of the observations fed to the agent by changing the action performed in that state to a randomly selected action from the set of actions. This was achieved by implementing a mechanism that processes the observations and replaces the action
of an observation (pair \( \{ \text{set, action} \} \)) by a random one, every 10 observations. By doing that we intend to prove the robustness and tolerance to perturbations of our approach.

The teammate identification and task identification steps (with focus on formation identification) were tested independently. The test results we present for the teammate identification step are the average results of a series of 20 independent trials, each ran over a different set of data points, matching the observations collected over 20 different matches. For the task identification step, we test its performance by removing one agent from each role, from each formation. The results for each missing role in each formation are the average of 5 trials each.

Each match runs through a course of 6000 time steps of 100 ms each, corresponding to 600 seconds or 10 minutes matches. If we consider the gathering of 1 observation (a pair \( \{ \text{state, action} \} \)) per time step, each data set of observations has \( \sim 6000 \) data points. In sum, for each teammate identification trial, we use a data set of \( \sim 6000 \) observations, whereas for a task identification trial we use 9 data sets of the same size (1 data set per teammate). In total, we performed 20 trials for teammate identification and \( 5 \times \#\text{roles} \times \#\text{formations} = 5 \times 3 \times 5 = 75 \) trials for task identification, with both clear and noisy data.

Towards establishing a baseline performance and then testing the ad hoc agent's behavior (a holistic test of our approach), we ran a series of 15 matches for baseline performance and two series of 25 matches with our ad hoc agent on the field (25 with clear data, 25 with noisy data).

### 5.2 Teammate Identification Tests

The teammate identification step is a cornerstone of our approach, since the task identification mechanism depends on it and so does planning, although indirectly. As such, the agent performance as a whole, is highly dependent on a correct classification of the teammates.

The results from the teammate identification process (identifying the teammates’ roles) are shown on Table 5.1. On the 2\textsuperscript{nd}, 3\textsuperscript{rd} and 4\textsuperscript{th} column, we list the average normalized score attributed by the defender predictor, the midfielder predictor and the forward predictor, respectively, based on the offline observations of the agent being classified. On the 5\textsuperscript{th} column we can observe the best predictor model’s average classification and on the 6\textsuperscript{th} (last) column, the mixed predictor model’s average classification (a triplet that consists on the average defender, midfielder and forward normalized scores). Considering the impossibility of gathering data and performing this task while the game is underway (online), these scores were achieved by running tests over the offline observations we gathered from previous matches.

On Table 5.2 we display the results from the teammate identification tests, performed with noisy data, as described in Section 5.1. Although the results obtained in these tests were not as good as those obtained with clear data, as expected, the agent is still able to perform a correct teammate identification, in average.

Overall, the results obtained were very positive, as can be deduced from the previously mentioned tables, thus contributing to better task identification results and, therefore, a better agent performance.
### 5.3 Task Identification Tests

In order to evaluate the task identification step of our approach, we tested our ad hoc agent’s accuracy when identifying the formation in which its team is playing and the role which is missing. As explained before, we used two different approaches to tackle this problem, best predictor model and mixed predictor model. We tested them separately so we could determine which performs best in this task. To properly test this, we placed the agent in every possible scenario, by removing 1 agent from each role from each of the formations in our playbook.

The results from the best predictor model when identifying the formation are shown on Table 5.3. As can be observed, the agent correctly identified the formation in 87% of the scenarios. In the only 2 cases in which it missed, the correct formation was the 2nd with the highest score, which means it was not far off. Since the teammate classifications are absolute in the best predictor models, and the agent got all of them right in these tests, the results of the trials performed with noisy data were the same. So there is no need to elaborate further.
Table 5.3: Task identification test results - best predictor model - formation selection - clear data.

The results from the best predictor model when identifying the missing role are shown on table 5.4.

Once again, our agent's accuracy is approximately 87% and the scenarios in which it fails to identify the missing role coincide with the scenarios in which it fails to identify the formation.

Table 5.4: Task identification test results - best predictor model - role selection - clear data.
Notice that the agent fails to identify the role properly when playing in a 541 formation, in which the Midfielder is missing, and in a 262 formation, with a Forward missing. Although it may appear a coincidence, this can be explained by the mechanism used by our agent (in the best predictor approach) to deduce the missing role from the identified formation. The teammate classification is absolute, in this case, which means that if the agent identifies a 361 formation (last row of the table) it is expecting exactly 3 defenders, 6 midfielders and 1 forward. When it observes 2 defenders, 6 midfielders and 1 forward, the missing role is identified as defender. From this we can extrapolate that whenever the agent fails to identify the formation it will always fail to identify the missing role.

We identified that as the main weakness of this approach, however, note that it only happens when the observed ad hoc formation (with one missing role) is very similar to 2 or more formations in our playbook. This is why the agent scores the correct formation with a high score as well (2nd best). To sum up, this approach proves to be very reliable. As we had discussed with formation selection, the role selection process with noisy data in this approach performed equally to the one with clear data.

The results from the mixed predictor model when identifying the missing formation, when it is fed clear observations data, are shown on Table 5.5. As can be observed, the agent correctly identified the formation in 87% of the scenarios. In the only 2 cases in which it missed, the correct formation was the 2nd with the highest score, as had already happened with the best predictor model. In fact, as far as the formation identification process is concerned, these two approaches perform equally well. However, the best predictor model delivers predictions in which the difference between the scores of the selected formation and the next best formation is generally higher than in the mixed predictor model.

<table>
<thead>
<tr>
<th>Formation</th>
<th>Missing Role</th>
<th>631 Score</th>
<th>541 Score</th>
<th>343 Score</th>
<th>361 Score</th>
<th>262 Score</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>631</td>
<td>Defender</td>
<td>0.524</td>
<td>0.471</td>
<td>0.221</td>
<td>0.215</td>
<td>0.148</td>
<td>631</td>
</tr>
<tr>
<td>631</td>
<td>Midfielder</td>
<td>0.450</td>
<td>0.358</td>
<td>0.152</td>
<td>0.143</td>
<td>0.101</td>
<td>631</td>
</tr>
<tr>
<td>631</td>
<td>Forward</td>
<td>0.732</td>
<td>0.573</td>
<td>0.114</td>
<td>0.229</td>
<td>0.076</td>
<td>631</td>
</tr>
<tr>
<td>541</td>
<td>Defender</td>
<td>0.507</td>
<td>0.614</td>
<td>0.246</td>
<td>0.332</td>
<td>0.164</td>
<td>541</td>
</tr>
<tr>
<td>541</td>
<td>Midfielder</td>
<td>0.649</td>
<td>0.582</td>
<td>0.166</td>
<td>0.232</td>
<td>0.110</td>
<td>541</td>
</tr>
<tr>
<td>541</td>
<td>Forward</td>
<td>0.487</td>
<td>0.519</td>
<td>0.103</td>
<td>0.207</td>
<td>0.069</td>
<td>541</td>
</tr>
<tr>
<td>343</td>
<td>Defender</td>
<td>0.152</td>
<td>0.223</td>
<td>0.706</td>
<td>0.248</td>
<td>0.385</td>
<td>343</td>
</tr>
<tr>
<td>343</td>
<td>Midfielder</td>
<td>0.235</td>
<td>0.232</td>
<td>0.531</td>
<td>0.190</td>
<td>0.254</td>
<td>343</td>
</tr>
<tr>
<td>343</td>
<td>Forward</td>
<td>0.268</td>
<td>0.359</td>
<td>0.465</td>
<td>0.286</td>
<td>0.310</td>
<td>343</td>
</tr>
<tr>
<td>361</td>
<td>Defender</td>
<td>0.234</td>
<td>0.374</td>
<td>0.332</td>
<td>0.614</td>
<td>0.355</td>
<td>361</td>
</tr>
<tr>
<td>361</td>
<td>Midfielder</td>
<td>0.346</td>
<td>0.554</td>
<td>0.331</td>
<td>0.468</td>
<td>0.240</td>
<td>361</td>
</tr>
<tr>
<td>361</td>
<td>Forward</td>
<td>0.264</td>
<td>0.423</td>
<td>0.160</td>
<td>0.439</td>
<td>0.146</td>
<td>361</td>
</tr>
<tr>
<td>262</td>
<td>Defender</td>
<td>0.067</td>
<td>0.107</td>
<td>0.245</td>
<td>0.223</td>
<td>0.671</td>
<td>262</td>
</tr>
<tr>
<td>262</td>
<td>Midfielder</td>
<td>0.120</td>
<td>0.193</td>
<td>0.429</td>
<td>0.299</td>
<td>0.689</td>
<td>262</td>
</tr>
<tr>
<td>262</td>
<td>Forward</td>
<td>0.160</td>
<td>0.256</td>
<td>0.270</td>
<td>0.479</td>
<td>0.519</td>
<td>262</td>
</tr>
</tbody>
</table>

Table 5.5: Task identification test results - mixed predictor model - formation selection - clear data.
On Table 5.6, we present the results of the trials analog to those we just mentioned, but conducted with noisy observations' data. If we weigh the data in this analysis table against the data from the trials with clear data, we can conclude that the performance of the formation selection was very similar. The noisy data used in these trials did not impact the results obtained as much as expected. In the cases where the formation selection fails, the correct alternative is the one with the second highest score. This is another step towards proving our approach’s robustness.

<table>
<thead>
<tr>
<th>Formation</th>
<th>Missing Role</th>
<th>631 Score</th>
<th>541 Score</th>
<th>343 Score</th>
<th>361 Score</th>
<th>262 Score</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>631</td>
<td>Defender</td>
<td>0.474</td>
<td>0.449</td>
<td>0.264</td>
<td>0.228</td>
<td>0.176</td>
<td>631</td>
</tr>
<tr>
<td>631</td>
<td>Midfielder</td>
<td>0.453</td>
<td>0.391</td>
<td>0.180</td>
<td>0.156</td>
<td>0.120</td>
<td>631</td>
</tr>
<tr>
<td>631</td>
<td>Forward</td>
<td>0.803</td>
<td>0.684</td>
<td>0.137</td>
<td>0.274</td>
<td>0.091</td>
<td>631</td>
</tr>
<tr>
<td>541</td>
<td>Defender</td>
<td>0.331</td>
<td>0.461</td>
<td>0.378</td>
<td>0.323</td>
<td>0.252</td>
<td>541</td>
</tr>
<tr>
<td>541</td>
<td>Midfielder</td>
<td>0.474</td>
<td>0.449</td>
<td>0.264</td>
<td>0.228</td>
<td>0.176</td>
<td>631</td>
</tr>
<tr>
<td>541</td>
<td>Forward</td>
<td>0.620</td>
<td>0.778</td>
<td>0.194</td>
<td>0.389</td>
<td>0.130</td>
<td>541</td>
</tr>
<tr>
<td>343</td>
<td>Defender</td>
<td>0.110</td>
<td>0.176</td>
<td>0.675</td>
<td>0.207</td>
<td>0.439</td>
<td>343</td>
</tr>
<tr>
<td>343</td>
<td>Midfielder</td>
<td>0.168</td>
<td>0.201</td>
<td>0.745</td>
<td>0.217</td>
<td>0.289</td>
<td>343</td>
</tr>
<tr>
<td>343</td>
<td>Forward</td>
<td>0.189</td>
<td>0.291</td>
<td>0.662</td>
<td>0.307</td>
<td>0.409</td>
<td>343</td>
</tr>
<tr>
<td>361</td>
<td>Defender</td>
<td>0.157</td>
<td>0.252</td>
<td>0.333</td>
<td>0.445</td>
<td>0.530</td>
<td>262</td>
</tr>
<tr>
<td>361</td>
<td>Midfielder</td>
<td>0.231</td>
<td>0.369</td>
<td>0.450</td>
<td>0.459</td>
<td>0.361</td>
<td>361</td>
</tr>
<tr>
<td>361</td>
<td>Forward</td>
<td>0.323</td>
<td>0.517</td>
<td>0.261</td>
<td>0.767</td>
<td>0.256</td>
<td>361</td>
</tr>
<tr>
<td>262</td>
<td>Defender</td>
<td>0.080</td>
<td>0.128</td>
<td>0.311</td>
<td>0.244</td>
<td>0.731</td>
<td>262</td>
</tr>
<tr>
<td>262</td>
<td>Midfielder</td>
<td>0.126</td>
<td>0.202</td>
<td>0.489</td>
<td>0.295</td>
<td>0.606</td>
<td>262</td>
</tr>
<tr>
<td>262</td>
<td>Forward</td>
<td>0.157</td>
<td>0.252</td>
<td>0.333</td>
<td>0.45</td>
<td>0.530</td>
<td>262</td>
</tr>
</tbody>
</table>

Table 5.6: Task identification test results - mixed predictor model - formation selection - noisy data.

The results from the mixed predictor model when identifying the missing role are shown on Table 5.7. As we mentioned on Section 5.2, the mixed predictor model has a tough time finding out which role is missing, since this depends on accurate player classification. Since all defenders and forwards exhibit some traits that could also be found in midfielders (which have a mixed behavior between defenders and forwards), most defenders and forwards get a high enough midfielder score to disturb the process of finding the missing role.

For instance, consider the case in which a defender performs a pass to a teammate. Performing a short pass to a teammate is a behavior typically associated with midfielders, so the midfielder predictor will increase the agent's midfielder score, which means that it is more likely that the agent in question is a midfielder. This type of event repeated, over time, will contribute to reaching a less than accurate conclusion regarding the missing role in the ad hoc team. This difficulty translates to a poorer role selection than the one observed in the best predictor model test results, with an accuracy of approximately 47% versus the much higher 87% we had seen before.
On Table 5.8, we present the results of the trials analog to those we just mentioned, but conducted with noisy observations’ data. If we weigh the data in this analysis table against the data from the trials with clear data, we come to the conclusion that there are few differences in terms of the classification efficacy. However, in this second test of sets, the classifications have less certainty, i.e., the difference between the selected role and the next best option is smaller than in the previous tests, which denotes a decrease in the quality of this process, due to the noisy data used.

To sum up, we must say that the results from the tests performed to the task identification step of our approach were, in fact, positive. Despite adding noise to the experts’ observations data, the agent was able to correctly identify the formation in which it was playing and select the appropriate role most of the times. Moreover, the noise in the observations did not make a big impact in the agent’s ability to do so, and that is clear when we put the results with clear and noisy observations side by side and compare them.

Although these test results were favourable, we must underline the superior performance of the simpler best predictor teammate models when compared to the more complex mixed predictor teammate models. This becomes evident to a greater extent if we compare the efficacy of the role selection mechanism in both approaches.
5.4 In-Match Agent and Team Performance

5.4.1 Baseline Performance

One of the most important indicators that our agent is performing well (or not) is its behavior when deployed in a match. In order to determine what the introduction of the ad hoc agent changes in the match(es), we first need to assert how the teams perform without its presence.

First off, as discussed in Section 5.1, we started by conducting a set of matches among all formations (all versus all) and extracted some important data, which we display in Table 5.9. The indicators we gathered match with those previously mentioned in the beginning of this chapter. Note that we discarded the matches between two teams using the same formation, as no useful information could be drawn from those matches, since the agents in each team are exactly the same.

Although we could confirm that, as expected, there is a big uncertainty associated with these matches, some conclusions can certainly be drawn from this data. The passing accuracy is generally very high, certainly higher than in real football matches, which contributes to a more pleasant match to watch but does not necessarily lead to wins, as we can deduce from the data in Table 5.10. Bumping into teammates is not an issue as big as expected, since there is only 1 bump per game, on average. The shooting accuracy is also very high and that contributes to a lot of goals being scored every match (an average of $\sim 8.6$ in all matches).
We sum up the statistics from these matches in Table 5.10, which we will now focus on, trying to determine which factors contributed most to the success/failure of certain formations. The shots, goals and passes represented in this table are average values per match.

Starting from the top, the most successful formation was clearly the 541. Even though this formation produced better results than all others, most of the matches it lost were the ones where the shot accuracy was the lowest (this formation averaged 86.21%).

That, combined with the higher shot accuracy of the top 3 most successful formations, brings us to the conclusion that shot accuracy is, in fact, one of the most important factors to consider. Also, shot frequency is the highest in the top formation (the 541). If we turn our attention to passing accuracy, we

Table 5.10: Test matches’ statistics - regular teams - average values per match ordered from best to worst win-draw-loss ratio, disregarding matches between two equal formations.
will find that, against all odds, the team with the highest passing accuracy was the least successful (262 formation). This is due to the fact that a crowded midfield sector (6 players in that role) translates to more short passes, which are easier to complete. Although the team in this formation passes the ball around nicely, the lack of defenders (only 2) leads to a lot of chances from their rivals and a lot of conceded goals.

We can observe that the amount of conceded goals decreases with the increase in the number of defenders in the formation, which is logical, however, not conceding as much is not enough to win (the 631 is a very defensive approach - see Section 4.2 - and its results were only average). On the opposite end of the spectrum, an increase in the number of forwards does not always translate to an increase in goal chances. A crowded defense and a fairly crowded midfield sector, even with less forwards, prevails over an unbalanced attacking formation.

### 5.4.2 Ad Hoc Agent/Team Performance

Now that we have established a baseline performance, we will proceed to reveal the results obtained when replacing one of the agents from the home team with our ad hoc agent, in each match and then compare those with the indicators gathered above. This time around, we also had to run every possible game in which the ad hoc formation was the home formation. Earlier some games were omitted since those games had already been played with the home and away sides reversed (on the previous tests it did not make sense to play a 541 vs 631, since we had already played a 631 vs 541 with no ad hoc players).

The tests referred in this section were ran with noisy data, as described in Section 5.1. The in-game tests with clear data are not explicitly described in this particular section, since the results did not differ much from those conducted with noisy data. Given the imperviousness of the teammate identification and task identification steps to the noise in the observations’ data (see Sections 5.3 and 5.2), it seemed pointless to detail two very identical results, so we selected the one that puts our approach to a harder test.

The aforementioned tests allow us to compare the results and performance indicators prior and posterior to the insertion of the ad hoc agent, since the only factor that changed was exactly its presence. The ad hoc team is always the home team and the away team is always a regular team (with no ad hoc agent). Remember that our agent will sometimes fail to identify the correct missing position due to an error in the task identification process, the teammate identification process, or both.

Having analyzed both the ad hoc team, in general, and the ad hoc agent, in particular, we summed up the test matches’ statistics in Table 5.11. If we compare these numbers with those in Table 5.10, we notice an increase in passing accuracy, as well as an obvious increase in the number of shots fired and goals scored by the home team (ad hoc). The shot accuracy is lower but it is compensated by the shot frequency (which is a determining factor in the team’s success, as can be perceived by the win-draw-loss column). No particular formation performed better than all of the others, however we maintain that the formations with less defenders concede the most goals and vice-versa. The 262 performed worse than
all others due to our ad hoc agent's misidentification of the missing position mentioned before.

If we analyze the remaining collective performance indicators, we reach the conclusion that there was an increase in the passing accuracy and shooting frequency (which seem to be the key drivers of a good performance). The bumping frequency remained fairly low, the passing frequency dropped very slightly and the shot accuracy also dropped due to an increase in the number of shots off target or saved by the goalkeeper.

<table>
<thead>
<tr>
<th>Ad Hoc Formation</th>
<th>Wins-Draws-Losses</th>
<th>Goals (Scored - Conceded)</th>
<th>Passes (Attempted - Completed)</th>
<th>Pass Accuracy (%)</th>
<th>Shots (Fired-Allowed)</th>
<th>Shot Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>541</td>
<td>3-2-0</td>
<td>4.20-3.00</td>
<td>11.20-9.40</td>
<td>83.93</td>
<td>6.20-4.40</td>
<td>67.74</td>
</tr>
<tr>
<td>343</td>
<td>3-1-1</td>
<td>4.40-2.80</td>
<td>19.00-17.40</td>
<td>91.58</td>
<td>6.40-3.80</td>
<td>68.75</td>
</tr>
<tr>
<td>631</td>
<td>3-1-1</td>
<td>3.80-3.00</td>
<td>12.40-11.40</td>
<td>91.93</td>
<td>6.20-4.20</td>
<td>61.29</td>
</tr>
<tr>
<td>262</td>
<td>2-2-1</td>
<td>3.00-3.50</td>
<td>16.20-14.60</td>
<td>90.12</td>
<td>3.40-4.40</td>
<td>88.24</td>
</tr>
<tr>
<td>361</td>
<td>1-1-3</td>
<td>2.60-3.20</td>
<td>17.20-15.20</td>
<td>88.37</td>
<td>4.40-4.80</td>
<td>59.09</td>
</tr>
<tr>
<td>Rounded Average</td>
<td>2.4-1.4-1.2</td>
<td>3.6-3.1</td>
<td>15.2-13.6</td>
<td>89.2</td>
<td>5.9-4.3</td>
<td>69.0</td>
</tr>
</tbody>
</table>

Table 5.11: Test matches’ statistics - ad hoc teams - average values per match - noisy data.

In Table 5.12, we can observe the ad hoc agent's individual statistics in each game. Despite the importance of the team performance, we decided to analyze the performance of the ad hoc agent separately so we could draw some conclusions regarding its influence in the ad hoc team. We measured the ad hoc agent's individual performance in terms of its passing frequency, passing accuracy, dribbling frequency, dribbling accuracy, shot frequency and shot accuracy.

Our agent's passing accuracy was above average, rarely missing a pass. Its dribbling accuracy was also very good, as well as its goal scoring ability. Even considering the fact that the shot accuracy was lower than the team's average, its frequency was high, which resulted in a lot of goals being scored, in some of the matches. With this being said, we can ascertain that the ad hoc agent was a positive influence in the team's performance.

We conclude this chapter by underlining our agent's influence on the ad hoc team's performance, thus contributing to the validity of our approach.
<table>
<thead>
<tr>
<th>Home (Ad Hoc)</th>
<th>Away</th>
<th>Score</th>
<th>Passes Attempted Complete</th>
<th>Dribbles Attempted Complete</th>
<th>Shots-Goals</th>
<th>Pass Accuracy (%)</th>
<th>Dribbling Accuracy (%)</th>
<th>Shot Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>631</td>
<td>631</td>
<td>1-1</td>
<td>2-2</td>
<td>13-8</td>
<td>1-1</td>
<td>100</td>
<td>62</td>
<td>100</td>
</tr>
<tr>
<td>631</td>
<td>541</td>
<td>5-1</td>
<td>3-3</td>
<td>7-5</td>
<td>4-2</td>
<td>100</td>
<td>71</td>
<td>50</td>
</tr>
<tr>
<td>631</td>
<td>343</td>
<td>8-6</td>
<td>1-1</td>
<td>17-12</td>
<td>6-5</td>
<td>100</td>
<td>71</td>
<td>83</td>
</tr>
<tr>
<td>631</td>
<td>361</td>
<td>4-2</td>
<td>2-2</td>
<td>6-4</td>
<td>9-4</td>
<td>100</td>
<td>67</td>
<td>44</td>
</tr>
<tr>
<td>631</td>
<td>262</td>
<td>3-5</td>
<td>2-1</td>
<td>15-10</td>
<td>2-2</td>
<td>50</td>
<td>67</td>
<td>100</td>
</tr>
<tr>
<td>541</td>
<td>631</td>
<td>6-6</td>
<td>2-2</td>
<td>3-2</td>
<td>1-1</td>
<td>100</td>
<td>67</td>
<td>100</td>
</tr>
<tr>
<td>541</td>
<td>541</td>
<td>4-4</td>
<td>5-5</td>
<td>11-8</td>
<td>3-2</td>
<td>100</td>
<td>73</td>
<td>67</td>
</tr>
<tr>
<td>541</td>
<td>343</td>
<td>4-1</td>
<td>1-1</td>
<td>7-4</td>
<td>5-2</td>
<td>100</td>
<td>57</td>
<td>40</td>
</tr>
<tr>
<td>541</td>
<td>361</td>
<td>3-0</td>
<td>1-1</td>
<td>7-5</td>
<td>2-1</td>
<td>100</td>
<td>71</td>
<td>50</td>
</tr>
<tr>
<td>541</td>
<td>262</td>
<td>5-3</td>
<td>1-1</td>
<td>15-11</td>
<td>4-2</td>
<td>100</td>
<td>73</td>
<td>50</td>
</tr>
<tr>
<td>343</td>
<td>631</td>
<td>4-3</td>
<td>2-2</td>
<td>15-13</td>
<td>5-3</td>
<td>100</td>
<td>87</td>
<td>60</td>
</tr>
<tr>
<td>343</td>
<td>541</td>
<td>3-5</td>
<td>3-3</td>
<td>11-8</td>
<td>4-3</td>
<td>100</td>
<td>73</td>
<td>75</td>
</tr>
<tr>
<td>343</td>
<td>343</td>
<td>6-4</td>
<td>1-1</td>
<td>7-5</td>
<td>8-5</td>
<td>100</td>
<td>73</td>
<td>63</td>
</tr>
<tr>
<td>343</td>
<td>361</td>
<td>3-3</td>
<td>0-0</td>
<td>8-6</td>
<td>3-2</td>
<td>100</td>
<td>75</td>
<td>67</td>
</tr>
<tr>
<td>343</td>
<td>262</td>
<td>3-0</td>
<td>1-1</td>
<td>6-6</td>
<td>3-3</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>361</td>
<td>631</td>
<td>6-4</td>
<td>1-1</td>
<td>7-5</td>
<td>6-4</td>
<td>100</td>
<td>71</td>
<td>67</td>
</tr>
<tr>
<td>361</td>
<td>541</td>
<td>4-5</td>
<td>0-0</td>
<td>8-6</td>
<td>4-3</td>
<td>100</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td>361</td>
<td>343</td>
<td>1-2</td>
<td>0-0</td>
<td>3-2</td>
<td>2-1</td>
<td>100</td>
<td>67</td>
<td>50</td>
</tr>
<tr>
<td>361</td>
<td>361</td>
<td>3-3</td>
<td>0-0</td>
<td>11-10</td>
<td>3-1</td>
<td>100</td>
<td>91</td>
<td>33</td>
</tr>
<tr>
<td>361</td>
<td>262</td>
<td>0-1</td>
<td>1-1</td>
<td>10-8</td>
<td>0-0</td>
<td>100</td>
<td>80</td>
<td>100</td>
</tr>
<tr>
<td>262</td>
<td>631</td>
<td>5-5</td>
<td>4-3</td>
<td>0-0</td>
<td>0-0</td>
<td>75</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>262</td>
<td>541</td>
<td>5-4</td>
<td>3-2</td>
<td>0-0</td>
<td>0-0</td>
<td>67</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>262</td>
<td>343</td>
<td>2-4</td>
<td>2-2</td>
<td>0-0</td>
<td>0-0</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>262</td>
<td>361</td>
<td>2-1</td>
<td>3-2</td>
<td>0-0</td>
<td>0-0</td>
<td>67</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>262</td>
<td>262</td>
<td>6-6</td>
<td>4-4</td>
<td>0-0</td>
<td>0-0</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 5.12: Test matches' data - Ad Hoc teams versus regular teams - Ad Hoc agent stats - noisy data.
Chapter 6

Conclusions

6.1 Approach Overview

With this work, we aimed to create an agent that integrated a team of unknown agents in a complex environment (Robocup 2D Simulation League), with whom it was not able to explicitly coordinate, thus making a valuable contribution to the field of ad hoc teamwork. Having chosen a multi-agent environment where the agents have a multiplicity of sensory inputs and actions available to them at any given time, we tried to distance ourselves from the conventional ad hoc teamwork environments, aiming to tackle a scalability issue that has been associated with ad hoc teamwork settings for a long time.

We adopted a view of ad hoc teamwork which regards teammate identification, task identification and planning as the three key steps involved in this problem. Summarily, first, we modeled an MDP where the reward function, \( R : S \times A \times S \rightarrow \mathbb{R} \), is unknown and the other players are regarded as expert agents in the domain. Then, we used a player classifier that maps the model created using the observed agents’ behavior to the model that best fits its description in a set of previously established standard behaviors (Teammate Identification). Afterwards, the reward function is obtained by applying IRL, using the observations of the experts’ behavior (Task Identification). On the third and final stage of the process, we solve the MDP, using Policy Iteration, and retrieve the optimal policy, \( \pi(S) \), that for each state returns an action for the agent to perform (Planning).

6.2 Main Contributions Overview

We can say that the results we obtained matched our initial expectations. Our ad hoc agent became a productive member of the ad hoc team and not only did it not reduce the team's performance but also managed to slightly increase it. Furthermore, we also reached some interesting conclusions regarding the importance of certain factors (such as shot frequency and passing accuracy) in the overall performance of a team in Robocup 2D Simulation League, as we demonstrated in Chapter 5. With this work, we took a step in the right direction in the mission to solve, or at least mitigate, the scalability issue that limits ad hoc teamwork settings and also contributed with yet another analysis of Robocup 2D Simulation League.
League matches.

### 6.3 Limitations and Weaknesses

Despite the favorable results obtained, our solution is not perfect. Although not often, sometimes our ad hoc agent still fails to identify the missing role and therefore tries to interpret a role that was not originally intended, as described in Chapter 5. This is exacerbated by the fact that our formations’ library contains formations that are very similar to each other, which were included on purpose to test our solution as hard as possible.

Furthermore, due to the limitations imposed to us by the Robocup 2D Simulation League’s SoccerServer, namely the restrictions in the sensory data provided to the agent, we were unable to run our agent with online observations and, instead, used agent observations collected from previous matches. You may recall from Section 2.3 that the agent is not able to obtain a full description of the game from the server. Instead it gets a limited description of the field and the other game components based on what direction it is looking to and its field of vision, which impacts our ability to collect accurate online observations.

### 6.4 Future Work

As we underlined before, our work intended to lessen the scalability issues that had been associated with ad hoc teamwork settings and better understand the mechanics of the challenging Robocup 2D Simulation League. With this work, we achieved that, however this does not close the ad hoc teamwork topic.

A possible future solution would be one that is able to surpass the difficulty in collecting observations and learning online, contributing to a more robust and flexible approach. It would also be interesting to apply it to other fields and bigger, even more complex domains.
Bibliography


59


Appendix A

Approach Diagram

Below you can find an enlarged version of our approach overview.
Figure A.1: A graphical representation of the architecture of our approach. The ad hoc agent is on top, the main data structures we will be using are represented in the tables on the bottom. The 4 circles in the middle represent the 3 steps of the implementation (each color matches one of the steps) of ad hoc teamwork. The arrow indicates information exchange between the components of our approach.
Appendix B

Domain Specific Approach Diagram

Below you can find an enlarged version of our domain specific approach overview for Robocup 2D Simulation League.
Figure B.1: A graphical representation of the architecture of our approach, applied to the RoboCup 2D Simulation League domain. The ad hoc agent is on top, the main data structures we will be using are represented in the tables on the bottom and the domain elements are on the left hand side. The 4 circles in the middle represent the 3 steps of the implementation (each color matches one of the steps) of ad hoc teamwork. The arrow indicates information exchange between the components of our approach.
Appendix C

Player Classification

Below you can find a diagram of our player classification mechanism.

Figure C.1: A summary of our player classification mechanism.