Ad Hoc Teamwork in the presence of non-stationary teammates

Pedro Miguel Matias Santos

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. João António Madeiras Pereira
Supervisor: Prof. Francisco António Chaves Saraiva de Melo
Member of the Committee: Prof. Manuel Fernando Cabido Peres Lopes

January 2021

Acknowledgments

I want to thank both of my supervisors, Francisco Melo and Alberto Sardinha, for the opportunity to work with them and for allowing me to explore this research area with so much potential. I am also thankful for steering me in the right direction, helping me whenever I had questions, and providing adequate resources when I needed. A special appreciation to João Ribeiro for helping me with the implementation of certain algorithms and for work and life advice.

I want to thank my parents, my brothers, and my sister for teaching me so much and support me throughout my life. To my dear Bárbara, for always believing in me and give me strength throughout these years. To my friends and colleagues who stayed with me from the beginning of my academic journey, until now. Thank you all for always being there for me.

Abstract

In this dissertation we address the Ad Hoc Teamwork Problem in the presence of non-stationary teammates. The current approaches for this problem are still distant from real-world conditions, the agent is not able to cooperate with teammates who have the ability to change behavior during interaction. This is a significant limitation for the cooperation with humans, and other intelligent systems, which do not follow strict policies. Despite that, previous works traditionally assume stationary teammates. So in this work (i) we explore the current limitations of the state-of-the-art on the Ad Hoc Teamwork Problem, and (ii) we propose a solution that combines the state-of-the-art approaches with adversarial algorithms. The proposed architecture is called PLASTIC-Policy with Adversarial Selection, or PPAS, enabling the agent to deal better with non-stationary teammates using transfer learning. To evaluate the proposed solution, we used a complex benchmark environment, the half field offense simulation domain. With the results obtained, we prove that it is possible to cooperate ad hoc with non-stationary teammates in complex environments. We make our source code publicly available at github.

Keywords

Intelligent Agents; Multi-agent Systems; Ad hoc Teamwork Problem; Adversarial Algorithms; Transfer Learning; Reinforcement Learning; Deep Learning.

Resumo

Nesta dissertação, abordamos o desafio *Ad Hoc Teamwork Problem* na presença de colegas de equipa não estacionários. As abordagens actuais para este problema ainda estão distantes das condições do mundo real, o agente não é capaz de cooperar com colegas de equipa que tenham a capacidade de mudar o comportamento durante a interacção. Esta é uma limitação significativa para a cooperação com seres humanos, e outros sistemas inteligentes, que não sigam políticas fixas. Contudo, os trabalhos anteriores usualmente assumem colegas de equipa estacionários. Assim, neste trabalho (i) exploramos as limitações do estado da arte feito no contexto deste problema, e (ii) propomos uma solução que combina abordagemns anteriores que atingiram o estado da arte com algoritmos adversariais. A arquitectura proposta é chamada PLASTIC-Policy com Selecção Adversarial, or PPAS, o que permite a um agente cooperar melhor com colegas de equipa não estacionários através de transferência de conhecimento. Para avaliar a solução proposta, utilizámos um ambiente complexo, normalmente utilizado para aprendizagem por reforço, chamado half field offense. Com os resultados obtidos, provamos que é possível cooperar ad hoc com colegas de equipa não estacionários em ambientes complexos. Disponibilizamos publicamente o nosso código fonte em github.

Palavras Chave

Agentes Inteligentes; Sistema Multiagente; Ad hoc Teamwork Problem; Algoritmos Adversariais; Transferência de Conhecimento; Aprendizagem por Reforço; Deep Learning.

Contents

List of Figures

List of Algorithms

Acronyms

HFO	half field offense
MDP	Markov decision process
RL	reinforcement learning
MAS	multi-agent systems
MAB	multi-armed bandit
PPAS	PLASTIC-Policy with Adversarial selection
SPP	Stochastic PLASTIC-Policy

1 Introduction

Contents

Robots and intelligent systems can be found everywhere. As the number of autonomous agents grows, and they start performing day to day activities, the need for smart autonomous agents able to cooperate becomes a necessity. Cooperative multi-agent systems (MAS) have been studied in the past years, in several fields, from disasters response (Schurr et al., 2005), health care systems (Alves et al., 2018), to naturally emergent curriculum (Leibo et al., 2019). Significant progress has been done, in great measure due to reinforcement learning (RL) techniques (Sutton and Barto, 1988) and deep learning (Schmidhuber, 2015). An example of that is the results achieved in complex multi-agent videogames, such as DOTA 2 (Berner et al., 2019) and StarCraft II (Vinyals et al., 2019).

Previous works on MAS traditionally assume that i) the agents share the same communication protocols, and/or ii) the coordination strategy is acquired a priori. These assumptions can be a problem, as the number of agents increases in different areas, we can not assume that agents will share the same cooperation protocols since different companies from all over the world will use their own frameworks and languages. The challenge of developing an autonomous agent capable of cooperating with unknown teammates, without explicit coordination and knowledge, in a common task is known as the ad hoc teamwork problem, presented by Stone et al. (2010).

Ad hoc agents are already needed for environments where unknown agents with diverse capabilities and no common framework must quickly work as a team. To illustrate the utility of an ad hoc agent, consider Example 1:

Example 1: A natural disaster happens, and due to lack of time or resources, it is not possible to reprogram the existing heterogeneous robots deployed in the area and provide them with the knowledge of each other's capabilities to assist the search and rescue operations.

In the above scenario, only ad hoc agents would be able to work together without the need to be explicitly provided with strategies in advance.

The ad hoc teamwork problem has been studied for several years (Bowling and McCracken, 2005; Rodrigues, 2018; Ravula et al., 2019; Stone, 2016; Chen et al., 2020). Stone (2011) presented a cooperative multi-armed bandit approach, and more complex works, dealing with bounded-memory teammates (Melo and Sardinha, 2016). Melo and Sardinha (2016) even extended the problem by adding the "task identification" as a sub-task of the ad hoc teamwork problem.

Recent work proposes (Barrett et al., 2017; Rodrigues, 2018) the use of reinforcement learning (RL) and transfer learning techniques. Barrett et al. (2017) presented PLASTIC, an algorithm that reached state-of-the-art results, by reusing knowledge learned from previous teammates to quickly adapt to new teammates.

These past works assume that teammates will present a stationary behavior, meaning that they will keep the same behavior during the interaction. Such limitations can not be ignored, because in real-world conditions the ad hoc agents will have to cooperate with a great spectrum of agents with multiple levels of adaptation.

In this thesis, we address the challenge of non-stationary teammates in the ad hoc teamwork problem. Also, in order to approximate real-world conditions, we use a complex environment, the half field offense HFO environment. It is a well-known benchmark for reinforcement learning, which was also used by Barrett et al. (2017). It presents a continuous multi-dimensional state space, with real-time actions.

1.1 Problem Description

The ad hoc teamwork problem presents the challenge of creating an agent able to cooperate on the fly with unknown teammates. Some works achieved good solutions in complex environments, which is the case of Plastic-Policy (Barrett et al., 2017).

As Ravula et al. (2019) pointed out, most of the previous works in ad hoc teamwork assume that teammates follow stationary policies, which means that they assume teammates will always present the same behavior over the interaction. However, the ad hoc agent's performance can reduce or even may lead it to not complete a task, when it has to cooperate with non-stationary teammates.

So, in this thesis, we propose a novel approach that combines the policy structure of PLASTIC-Policy with a novel feature to enable an ad hoc agent deal with non-stationary teammates by using an adversarial prediction strategy, called PLASTIC-Policy with Adversarial selection (PPAS). And we tested three related questions:

1. Does PLASTIC-Policy perform poorly in the presence of non-stationary teammates?
2. In the presence of stationary teammates, is PPAS able to obtain the same performance as the original Plastic-Policy?
3. In the presence of non-stationary teammates, does PPAS outperforms the original Plastic-Policy?

These three questions will help us investigate the main research question of this thesis, "How can an ad hoc agent successfully cooperate with non-stationary teammates in complex environments?".

1.2 Contributions

The main contributions of our dissertation are:

i. Test state-of-the-art algorithms with non-stationary teammates: The state-of-the-art Plastic-Policy algorithm was only previously tested using stationary teammates. In this work, we test and analyze the impact of the presence of non-stationary teammates.

ii. Introduce PPAS: We introduce a new algorithm for ad hoc teamwork, PPAS, which combines the policy architecture of PLASTIC-Policy with an adversarial teammate prediction.

In addition to the main contributions above, our work also led to several secondary contributions of interest:

1. Evaluate PPAS in a complex environment: We evaluate our algorithm using the complex half field offense (HFO) environment, where we test it using two distinct settings;

2. Present a detailed exploration of the HFO environment: This environment, is a well-established benchmark for reinforcement learning, which is also used for ad hoc teamwork. However, this environment is very challenging and presents certain aspects poorly covered in the literature. In this thesis, we describe in more detail the steps needed to develop an ad hoc agent in HFO.

3. An open-source ad hoc agent for HFO: We release a documented implementation of Plastic-Policy, following the description on the paper Barrett et al. (2017). Also, we released our PPAS approach, and the players developed for the half field offense, on github.

1.3 Document Outline

This work is organized as follows:

Chapter 2 provides a background overview of the multi-armed bandit, Adversarial algorithms, Markov Decision Processes, deep learning, and reinforcement learning.

Chapter 3 provides both an overview of the relevant literature regarding ad hoc teamwork while highlighting the main algorithms and techniques.

Chapter 4 presents and discusses our solution to the problem, in terms of architecture and functionality.

Chapter 5 lays an evaluation approach to testing the three hypotheses in section 1.1, explains how the agent was concretely tested, and presents the obtained results, providing an answer to the hypotheses.

Chapter 6 concludes the dissertation by providing an overall balance and highlighting possible enhancements that need to be made in ad hoc teamwork, in order to be ready for the real world.

2 Background

Contents

In this section, we cover background material. We start by presenting the adversarial algorithms, but first, we introduce the Muli-Armed Bandit Problem, and how it was used to study, develop, and test these algorithms. Then, we describe the Markov Decision Process, used to model our problem, and two techniques used in our approach, Reinforcement Learning and transfer learning. Finally, we introduce the environment used to test our solution, half field offense, and why is this benchmark environment so relevant for our problem.

2.1 Multi-Armed Bandit Problem

The multi-armed bandit (MAB) or K-armed bandit problem (Robbins, 1952) is a classic reinforcement learning problem used to study the exploration vs exploitation problem. In MAB, a gambler must choose which of K slot machines to play, in order to maximize the sum of rewards earned through a sequence of arm pulls. At each time step, the gambler pulls the arm of one of the machines and receives a reward. As reward distributions differ from arm to arm, the goal is to choose a sequence of actions that maximize the total reward.

The problem of exploration vs exploitation can be explained as, if the agent acts greedy and always pulls the same arm, it might be missing out on some opportunities, as it cannot know whether it is picking the optimal one. But if it keeps exploring and changing arms, it is giving up some of the rewards as it is not always choosing the optimal arm.

The performance can be measured in terms of “regret” R. This is the difference between the expected return of the optimal strategy (pulling consistently the best arm) and the gambler’s expected return. So the goal is to minimize the expected regret:

$$E[R_T] = E\left[\max_{i=0,1,\ldots,K} \sum_{t=0}^{T} X_t^j - \sum_{t=0}^{T} X_t^a\right] \tag{2.1}$$

where t is the time step, a_t is the arm selected, X_t^a is the reward received by selecting arm a, $j \in [K]$ is the jth possible arm, K is the total number of arms, X_t^j is the reward returned by the optimal arm j.

MAB have been extended to multiple scenarios over the years. The adversarial model removes the statistical assumptions about the generation of rewards, while in the stochastic model we assume that the rewards of a given arm is independent and identically distributed sequence of random values.

2.2 Adversarial Bandit

The Adversarial Multi-Armed Bandit (Bubeck and Cesa-Bianchi, 2012) removes the statistical assumptions about the generation of rewards. Instead of a stochastic process where initially a reward is assigned to each arm $X^1, \dots, X^K \in [0,1]$, an adversary has complete control over the payoffs. So at each time step t, the gambler chooses an arm i, and at the same time an adversary assigns to each arm $j = 1, ..., K$ the reward X_t^j.

2.2.1 Prediction with experts

Now suppose that the gambler has available a set of N experts Θ. An expert is a predictor, that at each time step returns a prediction vector, with a probability value for each arm. So at each time step t, each expert gives a prediction in the form of a probability vector $\xi^i(t) \in [0,1]^K$, where K is the number of arms, and $\sum_{j=1}^{K} \xi_j^i(t) = 1$ for $i = 1, ..., N$. Being $\xi_j^i(t)$ the probability, also called advice of arm j, for expert i, on step t.

The gambler selects one arm j, and receives the reward X_t^j. The performance of the expert $i \in \Theta$ is measured by the loss l of their prediction and the actual outcome, the loss function can be defined as $l : \xi_j^i \times X_t^j \to \mathbb{R}$.

2.2.2 The exponentially weighted average forecaster

In the exponentially weighted average forecaster (Cesa-Bianchi and Lugosi, 2006), a weight is assign to each expert, w_i for $i \in 1, ..., N$. In this algorithm, the gambler calculates a probability vector $P(t)$, at each time-step t. This probability vector makes use of the weights w assign to each expert and the advice vector ξ given by the experts as:

$$P(t) = \frac{\sum_{i \in \Theta} w_i^{(t)} \xi^i(t)}{\sum_{j \in \Theta} w_j^{(t)}}$$

The probability vector $P(t)$ is used to select the arms, according to the strategy followed by the gambler. Also, at each time-step t, the gambler receives a reward vector X_t and calculates the weights of each expert, as $w_i(t) = exp\big(- \gamma_t \cdot l \left(\xi^i(t), X_t \right)\big)$, for $i \in 1, ..., N$. Being γ_t a positive value.

2.2.3 Exponentially weighted average forecaster for the ad hoc teamwork problem

The algorithm exponentially weighted average forecaster for the ad hoc teamwork problem, Algorithm 1, extends the exponentially weighted average forecaster algorithm. This online algorithm, presented by Melo and Sardinha (2016), is an adversarial approach developed for the ad hoc teamwork problem. It attempts to detect the current task by using a set of weights w. This algorithm assumes that the agent collected previous experiences in the form of “experts” E.

Algorithm 1 Exponentially weighted average forecaster for the ad hoc teamwork problem.

```
1: Initialize $w_\tau^{(0)} = 1$, $h = \emptyset$, $t = 0$
2: for all $t$ do
3:     Let $t \leftarrow t + 1$
4:     Let $P(h, a) = \frac{\sum_{\tau \in \mathcal{T}} w_\tau^{(t)} E_\tau(h,a)}{\sum_{\tau' \in \mathcal{T}} w_{\tau'}^{(t)}}$
5:     Select $\hat{A}(t) = argmax_{a \in \mathcal{A}} P(h, a)$
6:     Observe action $A_{-\alpha}(t)$
7:     Compute loss $\ell_\tau(h, A_{-\alpha}(t))$ as in $(4)$, $\tau \in \mathcal{T}$
8:     Update $w_\tau^{(t)} \leftarrow w_\tau^{(t+1)} \cdot e^{-\gamma_t \ell_\tau(h, A_{-\alpha}(t))}$
9: end for
```

So at each time step t, the agent selects an arm $\hat{A}(t)$ using the "advice vector" $E_\tau(h, a)$, probability vector over the available arms A, and the weights of each task w_τ. After observing the arms selected by each teammate $A_{-\alpha}$, it calculates the loss for each task ℓ_τ, and updates the weights w.

Associated with each task $\tau \in T$, they define an expert as a mapping $E_\tau : H \times A \to [0, 1]$, where H is the set of all finite histories and A the set of available actions. The expert E_τ provides at the same time, a prediction of the behavior of the teammate agent and a suggestion of the best action to make if the target task is τ, based on the history. More generally, we define a predictor as any mapping $P : H \times A \longrightarrow [0, 1]$. The loss $\ell_\tau(h, A_{-\alpha}(t))$ is the expert E_τ expected loss, given the history h, the teammates actions $A_{-\alpha}$, at time-step t. γ_t is a positive parameter.

2.3 Markov Decision Process

A Markov decision process (MDP) is defined as a 5-tuple (X, A, T, R, γ) where:

- X is a set of states;
- A is a set of actions available to the agent;
- $T(x(t+1)|x(t), a)$ is the transition function, which determines the probability of next state being $x(t+1)$ when coming from state $x(t)$ and using action a;
- $R(x(t), a)$ is the reward received when the agent transitioned from state $x(t)$ to state $x(t+1)$ using action a;
- $\gamma \epsilon [0, 1)$ is the discount factor.

At each time step, the agent observes the current state $x(t)$, and chooses an action a. The environment transitions to a new state $x(t+1)$ according to T, and the agent gets a reward $R(x(t), a)$, and observes $x(t+1)$.

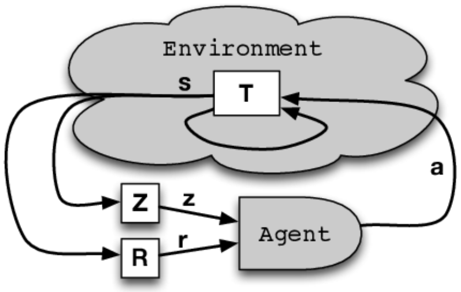

Figure 2.1: The agent interacts with the environment, performing an action a that affects the state environment according to a function T , producing the state $x(t)$. The agent perceives an observation $x(t+1)$ (in the figure corresponds to z) about the environment (given by a function Z) and obtains a reward r (given by a function R). Source: (Hernandez-Leal et al., 2017a)

Solving an MDP will yield a policy $\pi : S \rightarrow A$, which is a mapping from states to actions. An optimal policy $\pi^{\star}$ is the one that maximizes the expected discounted sum of rewards

Markov Decision Processes also respect the Markov Property, which affirms that "predictions are made based only on the current state of the system, and not on any previous states". This framework can model a huge variety of decision problems. X, A, and γ are always assumed to be known but T and R can be unknown. If everything is known, the optimal action to choose at each time step can be computed by Value Iteration (VI). In the cases where T and R are unknown, we are facing a Reinforcement Learning (RL) problem.

2.4 Reinforcement Learning

Reinforcement Learning RL formalizes the interaction of an agent with an environment using a Markov decision process (MDP), section 2.3. RL algorithms learn from experience by interacting with the environment in discrete time steps, Figure 2.1. The goal is to learn an optimal policy $\pi^{\star}$.

Deep reinforcement learning combines deep learning, i.e., neural networks, with RL. These algorithms emerged from the need to learn from large unprocessed inputs, overcoming the need for the agent designer to specify which features are relevant for the task at hand. The neural networks are used as function approximators.

In deep reinforcement learning, one of the most well-known algorithms is Deep Q-learning Neural Networks (DQN) (Mnih et al., 2015). The policy π and q values are represented with deep neural networks. The parameters of these networks are trained by gradient descent to minimize some suitable loss function.

At each step, based on the current state, the agent greedily selects an action and adds a transition $(x(t), A_t, R_{t+1}, \gamma_{t+1}, x(t+1))$ to a replay memory buffer, where *t* is a time step randomly picked from the replay memory, $x(t)$ is the state in the time-step t, A_t is the action executed, R_{t+1} is the reward of transitioning to the next state, γ_{t+1} is the discount factor, and $x(t+1)$ is the next state. The memory buffer holds a significant number of transitions, usually around a million transitions. So, the online network is updated using the gradient of the loss over the replay memory buffer. The neural network update consists on optimizing the parameters of the neural network by using stochastic gradient descent to minimize the loss, as described in Equation 2.2:

$$(R_{t+1} + \gamma_{t+1} \max_{a'} q_{\overline{\theta}}(x(t+1), a') - q_\theta(x(t), A_t))^2 \tag{2.2}$$

Finally, two techniques are used to stabilize the learning, experience replay, and target networks. The experience replay consists of sample training data from the replay memory buffer, instead of using the latest experience. The target network is a copy of the main neural network. The target network's Q values are used to train the main Q-network. This network's parameters are not trained, but they are periodically synchronized with the parameters of the main Q-network.

2.5 Half Field Offense

In this section, we will introduce HFO environment developed by Kalyanakrishnan et al. (2006), a subtask of Robotcup 2D. This was the environment chosen to study the ad hoc teamwork problem, due to its complexity. We cover this environment in more detail since a big part of our work was the exploration of the environment.

2.5.1 Robotcup 2D

The RoboCup 2D simulation domain has served as a platform for research in AI, machine learning, and multiagent systems for more than two decades. In this multiagent environment, there are two teams of 11 autonomous agents playing soccer on a simulated 2D field. Each game usually takes 6,000 simulation steps, each lasting 100 ms. Learning in the RoboCup soccer domain presents several challenges, such as:

- Continuous multi-dimensional state space - States are a tuple of continuous values, for example coordinates x and y are continuous, $x, y \in [-1, 1]$;
- Noisy sensing and actions - The observations have some margin of error, and actions sometimes fail;
- Multiple agents (including adversaries);

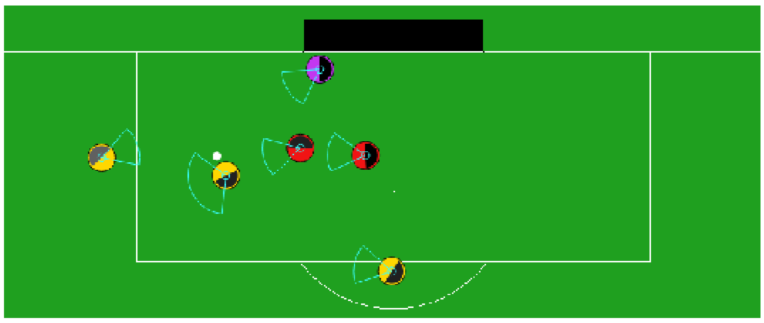

Figure 2.2: Half field offense game example. This is a screenshot taken of a game 3vs3, in which 3 offenses (yellow balls) play against 2 defenders (red balls) and 1 goalie (purple ball). The white point is the soccer ball and the black rectangle is the goal. At this moment the offense agents search for an opening in the defensive formation. The defenders and the goalie strive to intercept the ball or force it out of bounds. Source: (Hausknecht et al., 2016)

- Real-time actions - Every agent selects an action at the same time, at each step;
- Sparse rewards - The rewards are only available at the end of the game, and each game usually takes a significant number of steps. The environment does not return any rewards, but it is expected a positive return in case of winning, and a negative reward otherwise;

2.5.2 Half Field Offense

In 2006 half field offense (HFO) was presented by Kalyanakrishnan et al. (2006) as a novel subtask of Robotcup 2D, since the full RoboCup 2D task was too complicated. However, only in 2016 Hausknecht et al. (2016) released an open-source version of the half field offense (HFO).

Half field offense was developed as a reinforcement learning problem. In this subtask of RobotCup 2D, there are still two teams, one team only attacks and the other team only defends. The offense team attempts to outplay a defense team to shoot goals. Half field offense extends a simpler subtask of RoboCup soccer in which one team must try to keep possession of the ball within a small rectangular region, the keep-away subtask (Stone et al., 2005). The domain presents all the challenges of RobotCup2D, but on a smaller scale, which makes this environment suitable to test algorithms for single and multiagent learning, ad hoc teamwork, and imitation learning.

Environment

The HFO Environment, represented in Figure 2.3, builds upon the competition-ready RoboCup 2D server but supports smaller teams consisting of arbitrary mixes of automated teammates (NPCs, for "non-player characters") and player-controlled agents, up to ten players per side.

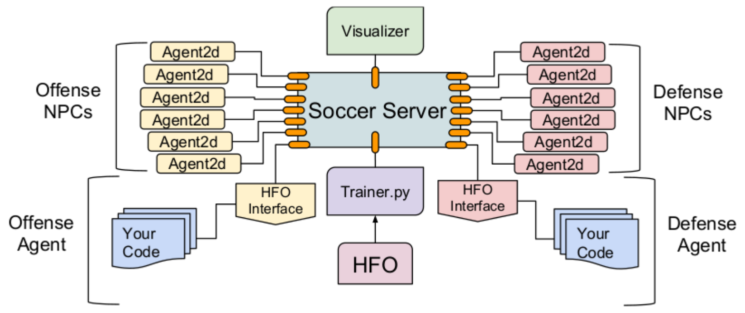

Figure 2.3: The HFO Environment is comprised of many separate processes that communicate over the network with the RoboCup 2D soccer server. HFO starts these processes, ensures they communicate, and oversees the games. A user needs only to specify the number of offensive and defensive agents and NPCs and then connect their agent(s) to the waiting Agent Server(s) via the HFO interface. Source: (Hausknecht et al., 2016)

Game Set-Up

In half field offense, an offense team of m players has to outsmart the defense team of n players, including a goalie, to score a goal, typically $n \geq m$. The task is played over one half of the soccer field and begins near the half field line. It ends when one of four events occurs:

1. **Goal**: A goal is scored;
2. **Out of Bounds**: The ball is out of bounds;
3. **Captured by Defense**: A defender gets possession of the ball (including the goalie catching the ball);
4. **Out of Time**: The time of the episode was exceeded;

State Representation

The Environment provides two types of state representations, the Low-level representation which provides more features with less pre-processing, and the High-level representation which provides fewer, more informative features. In our work we use the High-level representation that has a minimum of 12 continuous features with an additional 6 for each teammate and 3 for each opponent;

Actions

The HFO domain provides three levels of actions. In our work we use a mix of Mid-level actions and High-level actions. The three levels of actions are:

- **Low-level actions** *(parametrized)* are provided for locomotion and kicking;

- **Mid-level actions** *(parameterized and discrete)* are still mostly but capture high-level activities such as dribbling;
- **High-level actions** *(discrete)* are available for moving, shooting, passing, and dribbling. Each high-level behavior is ultimately composed of low-level actions, but also incorporates Helios' strategy for choosing and parameterizing the low-level actions.

3

Related Work

Contents

Stone et al. (2010) saw the importance of having autonomous agents able to efficiently and robustly collaborate without prior coordination, which they presented as the ad hoc teamwork problem.

This is a complex challenge, previous works focused on different sub-tasks. In our case, we will focus on the ability to deal with non-stationary teams, we present and analyze the most relevant work that has already been developed in this field according to the teammate's adaptation capabilities.

We start by presenting how the ad hoc teamwork problem evolved over the years. Then, we present the work that has been developed, subdivided using the taxonomy presented by Hernandez-Leal et al. (2017a). Finally, we provide a global overview of the literature regarding transfer learning and a summary of the most important works for our approach.

3.1 Ad hoc teamwork problem

3.1.1 Problem Definition

Stone et al. (2010) bring forward the notion of ad hoc autonomous agents, in which an agent must cooperate with a team of unknown agents, with hidden behavior, without any pre-coordination. This differs from the normal multiagent setting because the ad hoc agent may not share the same algorithm as its teammates and there is no pre-coordination available to learn the strategy that the team follows. The authors organize teamwork situations along three dimensions:

- Teammate characteristics: features of the individual teammates such as action capabilities, sensing capabilities, decision making, and learning capabilities, whether they can communicate directly, and prior knowledge;
- Team characteristics: features of the collection of team members such as whether they are homogeneous or heterogeneous, how many teammates are on the team, and whether they can observe each other's actions;
- Task characteristics: features of the cooperative task to be performed such as the goal, the time horizon, whether it is turn-taking, and how carefully coordinated the agents need to be in their actions. Can they divide the task at a high level and then act independently, or do they need to coordinate low-level actions;

More recent works redefined and extended the ad hoc teamwork problem (Albrecht and Ramamoorthy, 2013; Melo and Sardinha, 2016). Albrecht and Ramamoorthy (2013) described it in a more general way based on three concepts: (i) Flexibility, which is the ability to solve a task with diverse agents; (ii) Efficiency, which is the difference between the agents' payoff and the time they needed to solve the task; (iii) Prior coordination, which is one requirement of the ad hoc problem is no prior coordination, which means that the agents do not know how other agents behave; So, according to the authors, the goal is to design an autonomous agent able to reach optimal flexibility and efficiency with no prior coordination between the ad hoc agent and the other agents.

On the other hand, Melo and Sardinha (2016) extended the problem. They added the ”task identification” as a sub-task of the ad hoc teamwork problem, by formalizing this challenge as a sequential decision problem, subdivided into i) task identification, ii) teammate identification and iii) planning. So in this setting, the ad hoc agent does not know in advance the task to be performed or how its teammates act towards performing it. In our work, we assume that our agent already knows the task and who are its teammates. However it does not know the teammates’ behavior, and it has to focus on the planning step.

3.1.2 Approaches

One important aspect of an autonomous agent is the ability to reason about the behaviors of other agents (Albrecht and Stone, 2018). This ability consists of modeling other agents’ actions, goals, and beliefs. It has been one of the priorities in the solutions for the ad hoc teamwork problem. Nevertheless, different works made different assumptions about their teammates. According to Hernandez-Leal et al. (2017a) there are three types of possible behaviors:

- **No adaptation Teammates** - These are teammates that follow a stationary strategy during the entire period of interaction;
- **Slow adaptation** - These teammates show non-stationary behaviour. However, it is a limited adaptation, for example providing bounds to the possible change in the current strategy between rounds;
- **Drastic or abrupt adaptation** - If the above assumptions are not in place, non-stationary teammates may show abrupt changes in their behavior, for example, changing to a different strategy (no limits) from one step to the next;

No adaptation teammates

These works usually addressed the ad hoc problem using ”types”. They utilize beliefs over a set of hypothetical types for the teammates (Albrecht and Ramamoorthy, 2013; Albrecht and Stone, 2017; Barrett et al., 2017; Rodrigues, 2018). If the agent is able to model the teammate’s behavior correctly, then this method can lead to fast and efficient teamwork in the absence of explicit prior coordination.

One of the first works to use types was Albrecht and Ramamoorthy (2013). They model the problem using the stochastic Bayesian game, which means the agent type determines its behavior. They proposed the Harsanyi-Bellman ad hoc Coordination (HBA), which computes probability distributions over the user-defined types and utilizes them in a planning procedure to find optimal actions. They even made a human-machine experiment and achieve a significantly higher winning rate in Rock-Paper-Scissors than the human participants. But this solution presents some challenges, such as, it requires the user to insert the types, which is not practical. They assume that the teammates' behavior follows one of the types, not addressing the problem of non-stationary agents. Also, the planning approach can be slow in complex environments, even more, if a model of the environment is needed. In our work, we also use a set of beliefs about the type of teammates, but these types are learned using RL, and they are used as "advice" to our algorithm.

Macke et al. (2021) present an approach that allows the ad hoc agent to detect the teammates' policies faster by looking into the history of the teammates' actions to disambiguate between potential policies, using a metric EDP. They also present a novel planning algorithm that uses the EDP metric, and it is able to plan when and what to communicate to its teammates. However, communicate with teammates has an associated cost, so the ad hoc agent can ask questions to the teammates to gain more information, but it receives a negative reward by doing so. This novel approach shows to be able to detect the policies faster than previous works. However, the planning algorithm can be too heavy for complex environments like HFO, and contrarily to Macke et al. (2021), our algorithm is able to detect the current team, without having access to the teammates' actions, and without sharing a communication protocol.

One more successful approach using types is "PLASTIC". Barrett et al. (2017) presented a new general-purpose algorithm based on transfer learning and RL techniques. PLASTIC reuses knowledge learned from previous teammates or provided by experts to quickly adapt to new teammates. They presented two versions of this algorithm:

- PLASTIC-Model, a model-based approach that builds models of previous teammates' behaviors and plans behaviors online using these models.
- PLASTIC-Policy, a policy-based approach that learns policies for cooperating with previous teammates and selects among these policies online.

Both algorithms assume that there are similarities between the new and old teammates' behaviors. The authors focused on enabling the ad hoc agent to cooperate with a variety of teammates in a range of possible environments.

The algorithms were tested on two benchmark environments. PLASTIC-Model was tested on the pursuit domain. This domain consists of a grid world where four predators try to catch the prey. The game ends when the predators surround the prey. The goal is to catch the prey as fast as possible. PLASTIC-Model was able to achieve good results. It allowed agents to reuse knowledge about previous teammates to quickly adapt by exploiting similarities in the new teammates' behaviors. However, PLASTIC-Model also showed to be slow and had difficulty dealing with high complexity environment. On the other hand, PLASTIC-Policy was able to deal with more complex environments. They tested it on HFO environment, described on Section 2.5.2. PLASTIC-Policy was able to scale well and efficiently select good policies for cooperating with its current teammates.

This work showed that learning a policy directly prevents the ad hoc agent from learning to exploit actions that work well in the model, but not in the real environment. Given that the policy learned will depend heavily on the teammates that the agent is cooperating with, it is desirable to learn a policy for each type of teammate. Then, the ad hoc agent will try to pick which policy best fits new teammates it encounters. That is why we use a model-free approach in our work.

Inspired by the previous work, Rodrigues (2018) proposed a new approach that combines model-based reinforcement learning with Monte Carlo Tree Search (MCTS) (Kocsis and Szepesvari, 2006) to enable an ad hoc agent to learn the underlying task and coordinate with its teammates. Their approach differs from the one proposed byBarrett et al. (2017) since they assume that the task, teammates' behaviors, and environment dynamics are unknown. So the authors address the ad hoc problem in full: 1) environment modeling, 2) teammate behavior modeling, and 3) planning. To target the first challenge, they use model-based reinforcement learning to learn a model of the environment and task. To target the second challenge, they use a fictitious-play-like approach to model the teammates. And finally, to plan online, they used the Monte Carlo tree search (MCTS) algorithm (Kocsis and Szepesvari, 2006). In the same way as Barrett et al. (2017), the authors used the pursuit domain to test their novel approach. The Pursuit Domain is well suited for the ad hoc teamwork problem, as the goal can only be achieved if all agents cooperate, and therefore, the ad-hoc agent should learn how its teammates behave to maximize its performance. The authors were able to reach similar performance to the state-of-the-art PLASTIC-Model while using less information regarding the team and the task. This work presented advantages compared with previous work because it allows the agent to reuse the environment information when teams change, unlike PLASTIC-Policy for example. However, as pointed out before model and planning only works fine in simple environments.

Using types to classify the teammates proved good results, but types are viewed as black box mappings from interaction histories to probability distributions over actions, which make them very limited. To deal with it, Albrecht and Stone (2017) proposed a general method that allows an agent to reason about both the relative likelihood of types and the values of any bounded continuous parameters within types. The innovative part is the method that maintains individual parameter estimates for each type and selectively updates the estimates for some types after each observation. They assume that their agent knows other agents' action space and that their past actions. However, the agent does not know the teammates' parameter values and type. Their approach allows minimizing computation costs by performing selective updates of the types' parameter estimates after new observations are made. Their method achieved substantial improvements in task completion rates compared to random estimates while updating only a single parameter estimate in each time step. However, their work was not tested in complex environments, so it is hard to take serious conclusions.

Despite archiving great results, these ”type” based approaches still present serious limitations since this approach completely relies on finding the most suitable policy for the current team. Firstly, this means that during the exploration phase the performance is low, which when dealing with critical tasks can be harmful. Also, it relies on the fact that the team follows one stationary policy, already known or very similar to past experiences. If this is not the case, the agent will keep changing between policies during the interaction, putting at risk the task. Despite the current limitations, the architecture used by PLASTIC-Policy proved to able to handle complex environments and adapt fast to new teammates. So in our work, we also adopt the PLASTIC-Policy architecture. In this way, we will use the same mechanism to learn policies and team models. And when cooperating on the fly we keep the previously learned policies and team models in memory and try to identify the most similar team. Our approaches differ in the action selection and beliefs update steps, in each we use an adversarial approach similar to Melo and Sardinha (2016).

Slow adaptation teammates

In terms of slow adaptation teammates, the most common teammate used is the ”memory-bounded” agent, which adapts its behavior according to the past history of steps, presenting a non-stationary behavior. These agents usually use a fixed number of past observations to select their actions.

Chakraborty and Stone (2013) focused on cooperating with even a broader class of teammates, Markovian teammates. These type of agents encapsulates a broad class of behaviors, including the ”memory-bounded” described before. The authors proposed a novel algorithm, Lcm, that converges to optimal cooperation with any Markovian teammate but also ensures good results cooperating with any other type of teammates, using ”safety measure”. However these ”safety measures” are fixed policies, which mean they are not nearly optimal. In our algorithm, we try to find the optimal policy to cooperate with the team, but if we can not find it, we combine knowledge from different learned policies, in the form of advice, which guarantees good performance.

Melo and Sardinha (2016) also dealt with ”memory-bounded” teammates. They proposed an ”on-line approach” able to deal with slow adaptation teammates, bounded-rationality best-response. Their algorithm is capable of detecting the task being performed by the teammates and acting accordingly. They called the algorithm ”Exponentially weighted forecaster for the ad hoc teamwork problem”, Section 2.2.3. It keeps a set of beliefs about which task is currently being performed, which are updated over time. Also, they use the ”advice” from ”experts” to select the actions, through a weighted approach. This algorithm is able to deal with teammates’ non-stationary behavior, so in our work, we adapt this algorithm for team identification instead of task identification. However, this model-based approach presents some drawbacks. In complex environments, it is very hard to learn a model of the world, which puts at risk the planning step. As Monteiro (2016) attempted to implement in the RoboCup 2D environment. Despite the success of implementing it, they also felt the difficulties of modeling and planning in such hard environments. So, as pointed out before, in our work we learn the policies directly, which makes it faster.

Although dealing with non-stationary teammates is still a hard task, some solutions (Chakraborty and Stone, 2013; Melo and Sardinha, 2016) were presented, they are too heavy for complex environments.

Drastic or abrupt adaptation teammates

The works presented above assume that the teammates' types remain stationary over the interaction or present slow adaptation. However, if the teammates follow an algorithm similar to PLASTIC, which means that they can be alternating between policies during the interaction, these assumptions do not hold.

Only Ravula et al. (2019) directly approaches the problem of teammates being able to switch their behaviors in the ad hoc problem. Also, despite not being in the ad hoc context, Hernandez-Leal et al. (2017a) also target this problem, and Hernandez-Leal et al. (2017b) introduce a new algorithm, DriftER, which detects drift between a learned set of types and the agent's current behavior, to help decide when the current typeset is not expressive enough of the behavior. This algorithm was able to detect switches with high probability. To test their work, they used standard form games such as the prisoner's dilemma and the Power TAC simulator.

While Hernandez-Leal et al. (2017b) work aims to develop a collection of behavior types, Ravula et al. (2019) focused on the problem of cooperation in the presence of type-changing teammates, by proposing a novel convolutional-neural-network-based change point detection algorithm for ad hoc teamwork. They considered the problem of agents dynamically switching between types through the course of the task, formulating it as a change point detection (CPD) problem. To identify the teammates' type, they used MAP type estimation as defined in the work of Albrecht and Stone (2017). According to MAP type estimation, the ad hoc agent maintains individual probability for each possible type and updates them after each observation. To plan the ad hoc agent's actions, they used simple planning algorithms such as A*, contrary to previous works that used MCTS. The reason was to reduce computational complexity and simplify the implementation. Finally, to detect the changing points, when the teammate agent changes his behavior type, they decided to use convolutional neural networks. They adopted these neural networks since detecting such a pattern requires both horizontal (time) and vertical (type) detailed analysis, similar to images. Their algorithm outperformed Bayesian CPD algorithms on a modified predator-prey domain.

These works approached a relevant problem, which is the cooperation with abrupt adaptation teammates. However, these algorithms were tested in simple environments, having access to full observable environments. These algorithms are too heavy for complex environments, such as half field offense. They also cannot detect slow adaptation teammates, due to their smooth change of behavior. So contrarily we do not detect drastic changes of behavior, our algorithm based on beliefs needs some interactions in order to detect the type of the teammate, but we guarantee good performance even in these cases, by using advice from multiple policies. Also, our algorithm is able to scale and achieve good results in complex environments such as HFO.

3.2 Transfer learning

The Reinforcement Learning (RL) learning process is still a high time-complexity process, mainly in complex multiagent environments. In order to accelerate the learning process, the reuse of previous knowledge, known as transfer learning, has been used (Hernandez-Leal et al., 2017b; Barrett et al., 2017).

As da Silva and Costa (2019) pointed out, the literature has shown that the reuse of knowledge significantly accelerates the learning process. Transfer learning algorithms use the experience gained in learning to perform one task to improve learning performance in a related. This is especially important to the ad hoc teamwork problem. For example, to cooperate with a new team, previous information (for example, in the form of models or policies) will be useful to quickly have an acting policy. These ideas (e.g., reusing past policies) have inspired recent works on multiagent systems. Our approach also relies on transfer learning to reuse previous knowledge to interact on the fly with different teams. In the same way as Barrett et al. (2017), we have in memory the policies and team models from past teams. So while interacting with the teammates, we use the team models to calculate the similarities between teams and use the policies to select the actions.

3.3 Summary

The ad hoc teamwork problem is very complex, and it is almost impossible to tackle all the challenges at once. Therefore, most of the works address only a small subset of possible ad hoc teamwork scenarios and make different assumptions. One problem that was poorly addressed is the cooperation with non-stationary teammates in complex environments. That is the focus of our work.

Out of all the approaches, PLASTIC-Policy (Barrett et al., 2017) and the ”Exponentially weighted forecaster for the ad hoc teamwork problem” (Melo and Sardinha, 2016) are the ones to give the most promising results for our approach. PLASTIC-Policy scales well in complex domains such as HFO, by using transfer learning techniques, and reinforcement learning to learn the policies. We also rely on these two techniques, but the policies selection mechanism used by PLASTIC-Policy is only based on ”types”, so it only works well when new teammates behave similarly to the chosen teammate type. To target this issue we adapted the online approach presented by Melo and Sardinha (2016). It is able to achieve results nearly as good as using the ”best policy”, which means that even if the agent is unable to detect the correct task, it is still able to do good choices using all its previous experiences. So, by following this approach we tackle the problem of cooperating with teams that display non-stationary behaviors, in the ad hoc teamwork problem.

4

Solution

Contents

In this section, we describe our solution to the problem of ad hoc teamwork in the presence of non-stationary agents, in terms of approach and architecture. We will describe and present our new algorithm PLASTIC-Policy with Adversarial selection (PPAS), capable of handling non-stationary teammates in complex environments. Our algorithm makes use of the Plastic architecture (Barrett et al., 2017), augmented with adversarial algorithms (Melo and Sardinha, 2016). All the source code is public, available at `https://github.com/pedMatias/matias_hfo`. We implemented our agent in Python 3.6 and set up the model using the Keras library, running all the experiments on a laptop.

4.1 Approach

We propose and test a novel solution, that combines elements from already existing work discussed in the related work section. These are the state-of-the-art Plastic-Policy (Barrett et al., 2017), and an online approach presented by Melo and Sardinha (2016), exponentially weighted average forecaster for the ad hoc teamwork problem, which is an extension of exponentially weighted average forecaster algorithm (Cesa-Bianchi and Lugosi, 2006).

In our solution, we use a policy-based approach, instead of a model-based one. Even though model-based approaches, such as Plastic-Model (Barrett et al., 2017), be able to achieve very good results in the ad hoc teamwork problem, in complex domains the performance drops. This low performance is partially due to the planning algorithms, such as UCT or MCTS, which may achieve the worst results due to the inaccuracies of the environment models, also MCTS was not designed for states with continuous features (only discrete features). Besides, planning a sufficiently effective behavior is not always possible due to time constraints between the decisions. Therefore, we choose to go for a policy-based approach, which allows us to directly learn a policy for acting in the environment, rather than planning online.

As exposed before, our approach extends the Plastic-Policy architecture (Barrett et al., 2017). When interacting on the fly with a new team, this architecture makes use of previously learned policies, and it keeps a probability or belief vector of the similarity between the new team and the past teams. At each time step, PLASTIC-Policy selects an action according to the most similar policy. This algorithm learns a policy and a team model for each past team. The policies are learned using RL techniques, and the team models are built using the collected experiences. In terms of updating the beliefs, PLASTIC-Policy uses the algorithm polynomial weights algorithm (Nisan et al., 2007).

However, there are some issues with this approach. When in the presence of teams that are not in the agents' team library, the agent may not be able to select an adequate policy (i.e., behaves almost randomly). And when in the presence of non-stationary teammates, the agent will keep trying to identify the best team for a long period, and may never be able to do it at all. So when interacting with a team able to change its own behavior, the agent will keep switching between policies, which threatens the

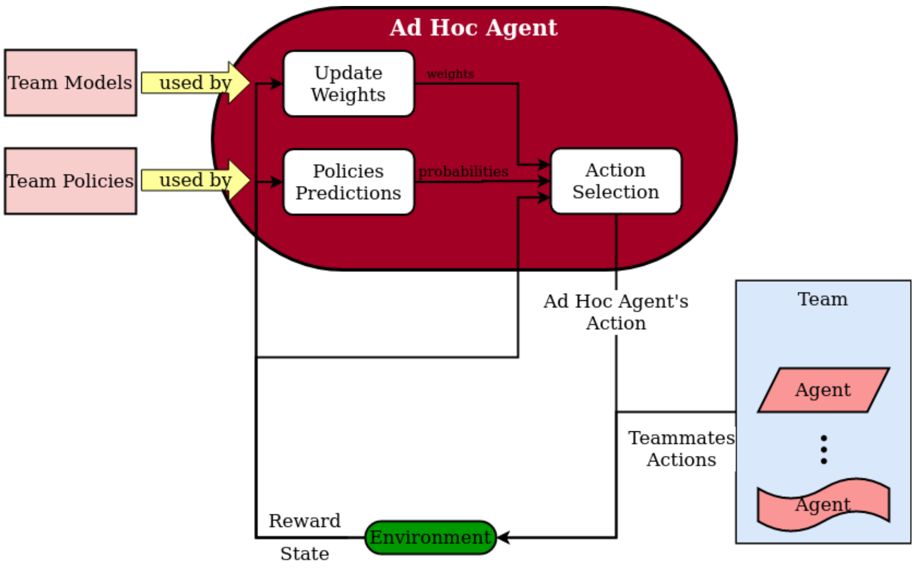

Figure 4.1: Overview of using the PPAS to cooperate with unknown teammates.

success of the task.

So in our approach, called *PPAS*, instead of selecting just one policy, we take into consideration the prediction of all policies, using a greedy weighted majority. With this approach we tackle two problems:

1. In stationary scenarios, early in the interaction, the agent can do good predictions, nearly as good as the "best policy";

2. In non-stationary scenarios, the agent can achieve better results, tracking the slowly changing teammates using the information in its team library;

Also, in this work we assume that the task is known, all teammates share the same goal, the environment is partially observable, the agent only has access to its local reward, and the teammates are homogeneous, i.e., all the teammates have the same type of behavior (follow the same policy).

4.2 Architecture

The architecture of the solution proposed can be seen in Figure 4.1, which is based on PLASTIC-Policy. The different boxes represent functions developed in this work. At every time-step, the ad hoc agent observes the new state $x(t+1)$ and reward r given by the environment, updates the weights of the old teams according to the similarity to the new team, and selects the best action a according to the policies trained before.

In terms of data structures we used:

- Policy Model - For each team, we saved a neural network, which takes as input a state and returns a probability vector of the available actions.
- Team Model - For each team, we saved a KD-Tree containing all the transitions collected while interacting with that team. It allows us to calculate the similarity between teams, as we explain in 4.2.2.

4.2.1 Training the Team Policies and Team Models

We use a policy-based model, so the agent directly learns a policy for acting in the environment. Since we treat the teammates as part of the environment, the policy will depend on the teammates that the agent is cooperating with. So we need to learn a policy for each past team.

The agent interacts with the team for a certain number of episodes. It stores its experiences as the tuple $< x(t), a(t), r(t), x(t+1) >$, where $x(t)$ is the original state, $a(t)$ is the action, $r(t)$ is the reward, and $x(t+1)$ is the resulting state. While playing with each team our agent learns a policy using deep q-network (Mnih et al., 2015). To save the model we use neural networks, in which the input of the model is the observation state, and the output is an array with the size of the number of actions available. The output array values are the Q-values for each action.

To save the team models we use a combination of KD-Trees and arrays. The KD-Trees nodes contain the $x(t)$ state and the corresponding transition id, and the arrays contain $x(t+1)$. These structures are used to calculate the similarity between teams, as explained in Section 4.2.2.

4.2.2 Update Weights

While playing with a new team, the agent will keep a vector of weights with size N, N being the number of past teams. So, the weight w_i is the probability that the old team i corresponds to the new one, in other words, it can be seen as the level of similarity between them.

So, at each time-step, the environment returns the new state $x(t+1)$, and the agent calculates the similarity with the past teams. For each old team i, the agent finds the stored state $\hat{x}(t)$ closest to $x(t)$ using the algorithm k-nearest neighbors, over the KD-Tree. Each node in the KD-Tree is identified by the Transition id, so after finding the stored state $\hat{x}(t)$, we can get its next state $\hat{x}(t+1)$, which is saved in a separated array. Then, it computes the Euclidean distance between $x(t+1)$ and $\hat{x}(t+1)$. Which gives us a vector d with the distances for each old team. The smaller the value d_i, the more similar the prediction is for that old team i.

To update the probability of the new team being similar to each old team, we use the exponentially weighted average forecaster (Cesa-Bianchi and Lugosi, 2006). So for i in $1, ..., N$:

$$w_i^{(t)} \longleftarrow w_i^{(t-1)} \cdot e^{-\gamma d_i}$$

4.2.3 Policies Predictions

When cooperating with an unknown team, at each time-step the environment returns the new state $x(t+1)$. Given $x(t+1)$ for each of the old team policies as input, it returns the probability vectors of the actions, which we will call the advice vectors $\xi(t)$. These values will be used to select the best action.

4.2.4 Action Selection

As pointed out before, the vector of weights w and the policies predictions ξ are used to select the best available action. We present the Algorithm 2, which is an adaptation of the algorithm exponentially weighted average forecaster for the ad hoc teamwork problem, presented by Melo and Sardinha (2016). In their work they tried to predict the correct task they were dealing, while in this case instead of the task, we try to predict the most similar team we encountered before. The Algorithm 2:

Algorithm 2 Exponentially weighted average forecaster using similarity for the ad hoc teamwork problem.

```
1:  Initialize $t = 0$, $w_i^{(0)} = 1$ for $i = 1, ..., N$.
2:  for all t do
3:      Let $t \longleftarrow t + 1$
4:      for $\pi \in$ Policies do
5:          Get advice vectors $\xi(t) = \pi(x(t))$
6:      end for
7:      Let $W_t = \sum_{i=1}^{N} w_i(t)$ and for $a = 1, ..., A$ set
            $$p_a(t) = \sum_{i=1}^{N} \frac{w_i(t)\xi_a^i(t)}{W_t}$$
8:      Select action $\hat{A}(t) = argmax_{a \epsilon A} p_a(t)$
9:      Observe new environment state $x(t+1)$
10:     for $m \in$ TeamModels do
11:         Compute similarity distance $d$ = $m(x(t+1))$
12:         Update
                $$w^{(t)} \longleftarrow w^{(t-1)} \cdot e^{-\gamma d}$$
13:     end for
14: end for
```

where Policies is the set of policies trained with each old i and TeamModels is the set of team models trained with each old team i, for $i = 1, ..., N$.. The other variables, t is the time-step, $w_i^{(t)}$ is the weight of the old team i, $\xi^i(t)$ is the advice vector returned by the policy π and $\xi_a^i(t)$ is the probability of action

a according to policy π, W_t is the sum of all the weights, $\hat{A}$ is the action with the maximum average probability, $x(t+1)$ is the state features of the next state (new observation), d_i is the similarity metric between the new team and each old team i using the model m and γ is a positive value.

So at the beginning of an episode, the agent has a set of policies and team models, previously trained with N old teams. During the episode, the agent updates the weights according to similarities between the new team, and the old teams, and select the actions with higher probabilities. Then, using the weights and the predictions of each policy, the agent computes the probability of the available actions, and it does a greedy selection, by selecting the action with greater probability. The environment transitions to a new state, and returns the new observation features $x(t+1)$. And at each step, the agent updates the weights of each old team using the similarity calculated with the team model, as described in Section 4.2.2.

The goal of this algorithm is to use past experiences in order to be able to adapt to any team. When playing with a known team, the agent should be able to detect the correct team, i.e., the weight of the correct team should be the maximum value of the weights vector w. And even when cooperating with an unknown team, the algorithm should be able to select good actions.

5

Experimental Evaluation

Contents

In this chapter, we describe and discuss how we evaluate our work. First, we introduce the experimental domain half field offense (HFO), described in Section 2.5.2. Second, we present how we collect the data. Finally, we compare our solution, *PPAS*, against the original algorithm PLASTIC-Policy, which we will call now on *Stochastic PLASTIC-Policy*, to show the advantage of our approach in the presence of non-stationary teammates.

We had three questions to investigate:

1. Does PLASTIC-Policy perform poorly in the presence of non-stationary teammates?
2. In the presence of stationary teammates, is PPAS able to obtain the same performance as the original Plastic-Policy?
3. In the presence of non-stationary teammates, does PPAS outperforms the original Plastic-Policy?

5.1 Half Field Offense Domain

Similarly to Barrett et al. (2017), we evaluate our work in the half field offense domain (HFO), presented in more details in Section 2.5.2. HFO is a challenging environment, due to:

- Continuous multi-dimensional state space;
- Real-time actions;
- Noisy sensing and actions;
- Multiple agents (including opponents);
- Sparse rewards;

In this domain there are two teams, the offense team, and the defense team. Our agent belongs to the offense team, and the objective of our team is to score a goal. There are two settings usually used in this Domain:

- **Limited version** with two offensive players (including the ad hoc agent) attempting to score on two defenders (including the goalie), *2vs2*;
- **Full version** with four attackers (including the ad hoc agent) attempting to score on five defenders (including the goalie), *4vs5*;

These settings were suggested by Hausknecht et al. (2016) and used in Barrett et al. (2017). We use both settings in our work, the limited version and the full version, to capture the different levels of complexity in this task.

5.1.1 Game Description

At the beginning of each episode/game, the goalkeeper is near the goal, and the ball and the rest of the agents start in random positions. However, the offensive agents start in a range that ensures that they are closer to the ball than the defenders. At each time step, each agent observes the game state and selects an action. The game ends with one of the events:

- **Goal**: The offense team scores goal;
- **Out of Bounds**: The ball goes out of the game area;
- **Captured by Defense**: The defense team catches the ball;
- **Out of time**: The game exceeds the maximum number of steps allowed, 500 steps.

5.1.2 Evaluate the agent

Our goal is to **evaluate the cooperation level between the ad hoc agent and his team**, in other words, we want to measure how well does our agent cooperate with his teammates. We assume that the goals scored are directly related with the level of cooperation between the agents, so we use the fraction scored as performance metric. The fraction scored is the number of goals scored divided by the number of trials. The number of goals scored is always the same as the number of win games, since each game ends when a goal is scored.

The agents are evaluated for a certain number of trials. One trial consisting of a series of 25 games of HFO with one specific team. However, measuring teamwork is far from straightforward. We evaluate our solution PPAS and the original PLASTIC-Policy, in two scenarios, playing with a team of stationary agents (NPCs) and playing with team non-stationary agents (ad hoc agents). To evaluate the first scenario, where the ad hoc agents cooperate with five different stationary teams, described in more detail in Section 5.1.4, we adapt the algorithm presented by Stone et al. (2010), which evaluates an ad hoc team agent while considering the teammates and domains it may encounter. The Algorithm 3:

Algorithm 3 Algorithm for the evaluation of the ad hoc agent with the team of NPCs

```
1: procedure EVALUATE($a_0, a_1, A, d, n$)
2:     Initialize performance (score rate) counters $r_0$ and $r_1$ for agents $a_0$ and $a_1$ respectively to $r_0 = r_1 = 0$.
3:     Repeat:
4:       - Randomly draw a team of agents $B$, $|B|n$, from $A$.
5:       - Randomly select one agent $b \in B$ to remove from the team to create the team $B^-$.
6:       - Increment $r_0$ by $s(\{a_0\} \cup B^-, d)$.
7:       - Increment $r_1$ by $s(\{a_1\} \cup B^-, d)$.
       • If $r_0 > r_1$ then we conclude that $a_0$ is a better ad-hoc team player than $a_1$ in domain $d$ over the set of possible teams $A$.
```

where a_0 is the PPAS agent, a_1 is the original PLASTIC-Policy agent, A is a set of five NPC teams, d is the task, which is the game of HFO itself, n is the number of elements in the team, B is one the teams in A, B^- is the team B without an agent, and the function s is the score achieved by the team with each of the ad hoc agents, performing the HFO task. Our measure of performance r_0 and r_1 is team score rate, we assume that the better the cooperation between the team and our ad hoc agent, the higher the number of goals.

The second scenario, playing with a team of non-stationary agents (ad hoc agents), do not follows the Algorithm 3, in this test a team of ad hoc agents will play a certain number of games of HFO. First a team of PPAS agents, and then a team of original PLASTIC-Policy agents.

5.1.3 Defensive Players

We use the HFO benchmark automated agents (NPCs) as defenders. The NPCs can follow multiple policies. For our defensive players we chose to use the *agent2d* behavior, a policy derived from Helios, used in the 2013 RoboCup 2D championship. This code was released by Hausknecht et al. (2016). These defensive players follow complex strategies unknown to us, which allow them to exhibit strong but not perfect results.

5.1.4 Teammates

We use two types of teammates: i) NPCs, ii) ad hoc agents. The NPCs are externally-created teammates as part of the 2D simulation league competition. As Barrett et al. (2017), we use the binary releases from the 2013 competition. We use 5 teams from the 2013 competition: *aut*, *axiom*, *cyrus*, *gliders*, and *helios*. The NPCs are used to train the ad hoc agent and for testing. The ad hoc agents' teammates are a team of ad hoc agents of the same type. This team is used as a non-stationary team. We only use this team for testing.

5.1.5 Environment Model

This section describes how we model the HFO domain as an MDP. We use a similar model to Barrett et al. (2017) since it will allow us to compare results in the end.

State

The state matrix $s \in S$ describes the most relevant features, such as current positions, and orientations of the agents. As described in Section 2.5.2, the environment provides two types of state representations. We extracted our features from the high-level state representation, since it is compact, enabling us to speed up the learning process.

The features used were inspired by the ones used by Barrett et al. (2017). The state-space features include eight continuous features with an additional five for each teammate (8 + 5T). In the limited version game (1 teammate) there are 13 features but in the full version game (3 teammates) there are 23 features. The features are:

- Agent's features:
    1. X position (*float [-1, 1]*)
    2. Y position (*float [-1, 1]*)
    3. Orientation (*float [-1, 1]*)
    4. Goal opening angle (*float [-1, 1]*)
    5. Distance nearest opponent (*float [-1, 1]*)
    6. Has Ball flag (*int (-1, 1)*)
    7. Ball X Position (*float [-1, 1]*)
    8. Ball Y position (*float [-1, 1]*)
- For each Teammate i:
    1. X position (*float [-1, 1]*)
    2. Y position (*float [-1, 1]*)
    3. Goal opening angle (*float [-1, 1]*)
    4. Distance to nearest opponent (*float [-1, 1]*)
    5. Agent's pass opening angle to teammate i (*float [-1, 1]*)

Actions

As described in Section 2.5.2, HFO provides multiple action spaces. The actions available to our agent were developed using high-level actions and mid-level environment actions. The goal was to re-create the actions used by Barrett et al. (2017).

In the limited version game (1 teammate) there are 11 discrete actions but in the full version game (3 teammates) there are 13 discrete actions. When the agent has the ball, it can execute the actions:

1. **Shoot**: Executes the best available shot;
2. **Short Dribble**: For 4 steps, advances the ball towards the goal using a combination of short kicks and moves;

3. **Long Dribble**: For 10 steps, advances the ball towards the goal using a combination of short kicks and moves;
4. **Pass**$_0$: pass to teammate 0;
5. **Pass**$_1$: pass to teammate 1;
6. **Pass**$_2$: pass to teammate 2;

When the agent does not have the ball, it can execute the actions:

1. **Stay in the Current Position**: Agent does nothing during 4 steps;
2. **Move towards the ball**: Agent moves in the ball direction during 4 steps;
3. **Move towards the opposing goal**: Moves in the direction of his nearest opponent, during 4 steps;
4. **Move towards the nearest teammate**: Moves in the direction of his nearest opponent, during 4 steps;
5. **Move away from the nearest teammate**: Moves in the opposite direction of his nearest teammate, during 4 steps;
6. **Move towards the nearest opponent**: Moves in the direction of his nearest opponent, during 4 steps;
7. **Move away from nearest opponent**: Moves in the opposite direction of his nearest opponent, during 4 steps;

The reasoning and the implementation of the actions are described in detail in Appendix 7.

Reward Function

The HFO environment does not provide reward signals and instead indicates the ending status of the game. The reward function returns values according to the events:

- **Goal**: 1000;
- **Out of Bounds**, **Captured by Defense**, **Out of Time**: -1000;
- For **each time step** during the game: -1;

Transition Function

The transition function is defined by a combination of the simulated physics of the domain as well as the actions selected by the other agents. The agent does not directly model this function; instead, it learns multiple policies and trains multiple similarity models for each team.

5.1.6 Collecting Data

In our environment model, we treat an action as going from when the action starts until or the action ends, or another agent holds the ball, or the episode has ended. The environment does not directly give the information of which agent has the ball, so we had to define this metric as a metric (explained in more detail in Appendix 7).

Given that we only control a single agent, at each time-step t, the agent collects data in the form $< x(t), a, r, x(t+1), d >$, where a is our agent's action, $x(t)$ the current state, $x(t+1)$ the next state, r the reward, and d is a flag signaling if the episode ended.

We do not have access to the teammates' actions, so we store the resulting world states, which include the effects of its teammates' actions. Also, the agent does not store information about the opponents, it treats them as part of the environment. The sequence of the transition tuples is one training sample, which contains the agent's actions when the agent has the ball as well as when it is away from the ball.

5.1.7 Training

The training step can be divided into two parts: i) train the policy models, and ii) Train the team models. As described before, our agents trained with 5 different teams. For each team, we played 100,000 episodes, collecting training data. Each episode consisted of a HFO games with multiple steps. At the same time, use the Deep-Q neural network algorithm (Mnih et al., 2015) to learn a policy π_j for each past teammate type j. The implementation of the neural networks is described in Appendix 7. The goal was to capture a big range of episodes with each team.

After collecting all the training data, we trained a team model for each team. We followed the idea behind Plastic Barrett et al. (2017). As the authors described, if the agent observed the transition from state $x(t)$ to state $x(t+1)$, it will search for the most similar state to $x(t)$ observed in past experiences for each old team j. After finding that state $\hat{x}_j(t)$, and its next state $\hat{x}_j(t+1)$, the agent needs to calculate the distance between the $x(t+1)$ and $\hat{x}_j(t+1)$. And it will be this metric that is used to calculate the similarity between an old team j and a new team.

To do a quick search, able to find the stored state $\hat{x}_j(t)$ closest to $x(t)$, we decided to use the K-Nearest Neighbours algorithm (KNN). To save all this data in memory and quickly access it, we create the team model using a combination of KD-Trees and arrays. The KD-Trees nodes contain the $x(t)$ state and the corresponding transition id, and the arrays contain $x(t+1)$. These structures are used to calculate the similarity between teams, as explained in Section 4.2.

5.2 Results

In this section, we present the experiments we have done to demonstrate the effectiveness of PPAS. First, we introduce the test settings. Second, we present how we collected the metrics. Finally, we pitch PPAS against the version presented by Barrett et al. (2017), henceforth referred to as Stochastic PLASTIC-Policy (SPP), to show the advantage of our approach in the presence of non-stationary teammates.

In detail, we show the results of both algorithms in two scenarios. The first one is playing with a stationary team, a team of NPCs that follow a stationary policy. The NPCs teams present during this test are the same used to train the algorithms, so the ad hoc agents have a correct policy and a team model for these teams in their libraries. In the beginning of each trial one of the five teams of NPCs is selected randomly and the ad hoc agent is added to that team. We measure the score during the 25 games of half field offense.

The second test is playing in a team of non-stationary teammates with abrupt adaptation, a team of ad hoc players. In this case, the ad hoc agents do not have a correct policy to cooperate with their teammates, so the challenge is to use the policies and team models in their libraries to select the best actions. At the beginning of each trial, all the ad hoc agents are restarted, which means that the ad hoc agents weights are initialized to 1. During the trial, the agents keep the previous weights w from the previous episodes. In this scenario the team is always the same, and it is only formed by ad hoc agents similar to the current ad hoc agent being tested. So, first a team of PPAS agents is tested during a set of trials, and then a team of acSPP agents.

We test these two scenarios in the two half field offense settings, the limited version, and the full version, described in Section 2.5.2.

5.2.1 Limited Version (2vs2)

In this version, there are 2 offensive agents (including an ad hoc agent) and 2 defensive agents (including the goalkeeper). So the ad hoc agent teams up with another agent (NPC or ad hoc), against two defensive players (NPCs). We evaluate the agents over 1,000 trials each, so the performance is based on the metric fraction scored, i.e., the number of goals scored averaged over 1,000 trials. Each trial consists of a series of 25 games of half field offense. After each trial, the game, and the agents are restarted.

NPCs Team

In this setting, we test our algorithm PPAS and the Barrett et al. (2017) algorithm SPP playing with an NPC teammate. In each trial, the agent is placed on a team randomly selected from the 5 teams

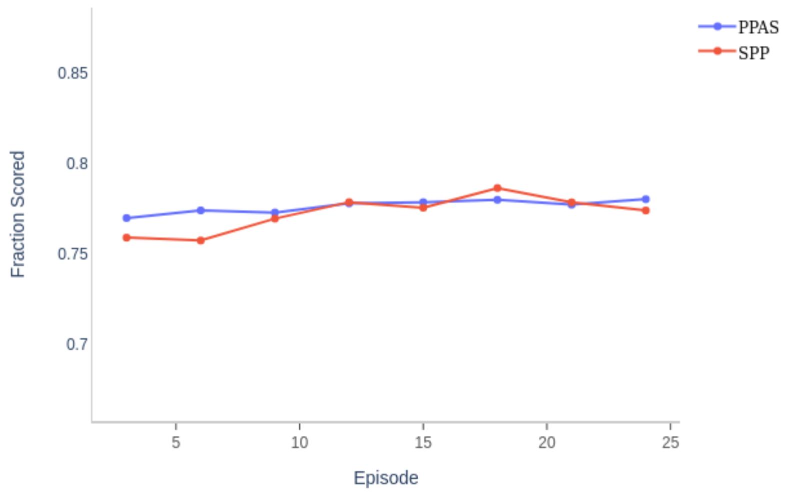

Figure 5.1: Scoring frequency in the limited half field offense task (2vs2) during 25 games. Playing with 1 teammate, type NPC. These results are an average over 1000 trials. Also the circles in the lines are the average fraction scored of the past three games, we did that to improve the visualization of the graph.

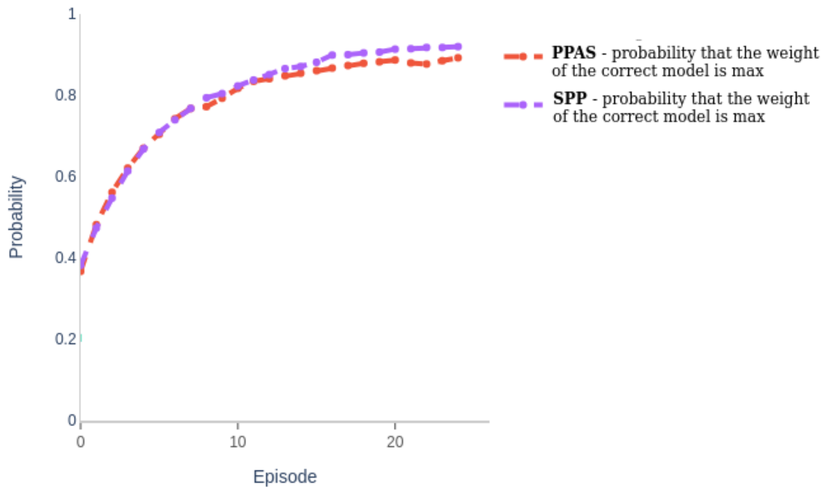

Figure 5.2: Evolution of the probability that the weight of the correct team has the maximum value from all the weights in vector w, when the ad hoc agent is playing in the limited half field offense task (2vs2) during 25 games. Playing with 1 teammate, type NPC. These results are an average over 1000 trials

described in Section 5.1.4, and they play a series of 25 games of half field offense. So in Figure 5.1 we can observe the fraction scored during 25 games by each algorithm, these results are an averaged of

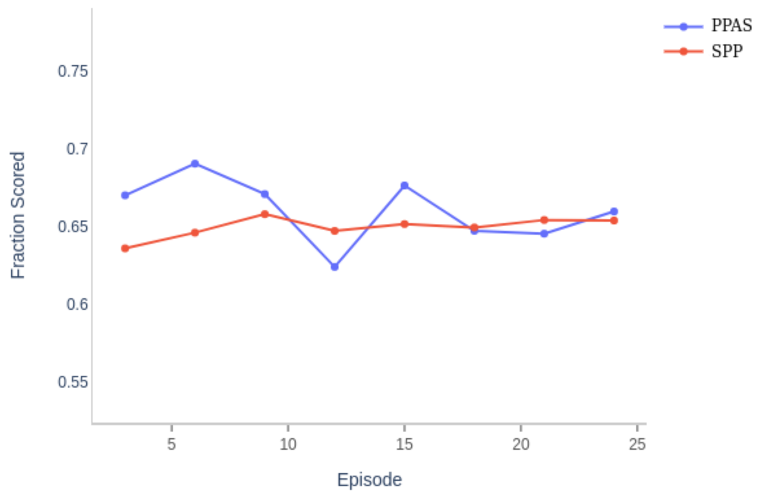

Figure 5.3: Scoring frequency in the limited half field offense task (2vs2) during 25 games. Each team is composed by 2 ad hoc agents of the same type. These results are an average of 1000 trials. Also the circles in the lines are the average fraction scored of the past three games, we did that to improve the visualization of the graph.

1000 trails. And in Figure 5.2 we have the evolution of weights of the correct team during the trials. The ”probability that the weight of the correct model is max” is, as the name suggests, the probability that the weight of the correct team has the maximum value from all the weights in vector w, in practical terms, if the probability increases it shows that the the ad hoc agents were able to identify the correct team, using similarity. If the weight of the correct policy is maximum, the ad hoc agent SPP will always choose the correct policy, and for the agent PPAS it will select the actions according to the weights assigned to each policy in its library, so the highest the weight, the more frequently the agent will choose the actions predicted by the correct team model.

In terms of results, we see that both PPAS and SPP reach almost the same results. In terms of the fraction scored 5.1, they both improve slightly over the trial. On top of that, PPAS can reach better results right away, but after nearly 10 games both algorithms reach the same winning rate. Both teams are able to identify the correct team they are playing with, as we can see in Figure 5.2. We can see that the ”probability that the weight of the correct model is max” is over 0.8, which corresponds to 80%, after only 10 games. So after 10 games 80% of the times, booth agents are able to identify the correct team.

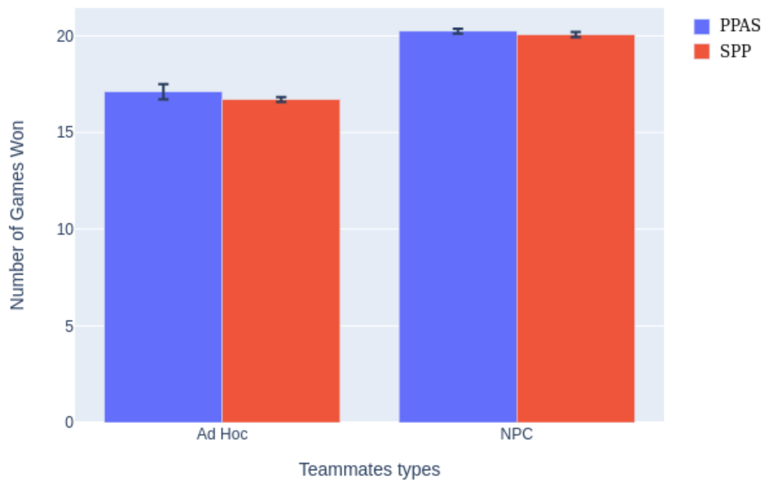

Figure 5.4: Number of win games in the limited half field offense task (2vs2), during 25 games. These results are an average of 1000 trials

Ad hoc team

In this setting, we test our algorithm PPAS and the Barrett et al. (2017) algorithm SPP playing with a similar teammate. In each trial, the agent plays a series of 25 games of half field offense with other ad hoc agent, which means that PPAS agent will play with other PPAS, and SPP agent will play with another SPP agent. So in Figure 5.3 we can observe the fraction scored during 25 games by each algorithm teams, these results are an averaged of 1000 trails. In this setting, it does not make sense to analyze the evolution of the weights over the trial, since there is not any correct team.

In terms of results, PPAS is able to reach better results in the early games. During the game with the weights updates, different policies are selected, which leads to the PPAS team to have a high score variation. By the end of the game, both algorithms achieve very similar results, as we can see in Figure 5.3. On the other side, SPP team starts with a lower wining rate. After nearly 10 episodes achieves the same results.

Discussion

In Figure 5.4, we can see the total number of win games in 25 episodes, in both scenarios, playing with an ad hoc teammate, and playing with an NPC teammate. As we can see both algorithms reach similar scores. The only small difference is when playing with similar ad hoc agents, PPAS can achieve slightly better results than SPP, the PPAS team wins around 17 games, and the SPP team wins 16 games,

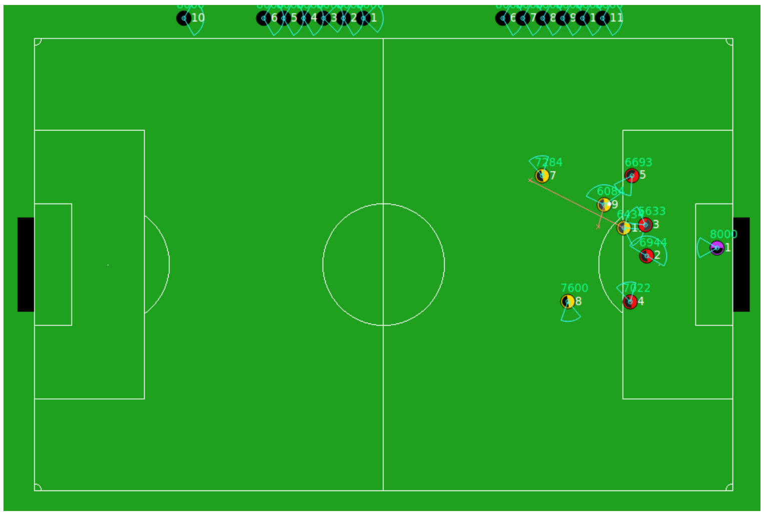
Figure 5.5: Half field offense game in the full Version. Yellow team - offense team; Read Team - defensive team;

during 25 games. So, in this setting, we can conclude that both algorithms are able to quickly identify the correct team when playing with an NPC teammate, and achieve good results. Even when playing with other ad hoc teammate, the differences are not statistically significant.

Despite being able to achieve good results, this setting can be a bit deceiving, because a very competent player is able to score by himself and cooperation is not a must. Individual capabilities have a high impact in this setting. For this, reason we also test the performance of the two algorithms in the full Version (4vs5).

5.2.2 Full Version (4vs5)

In this version, there are 4 offensive agents (including ad hoc agent) and 5 defensive agents (including the goalkeeper). So the ad hoc agent teams up with 3 agents (NPCs or ad hocs), against 5 defensive agents (NPCs). This setting is significantly more complex than the previous one, as we can see in Figure 5.5, the four defenders (red circles) use complex tactics that force the offense team (yellow circles) to cooperate, in order to score. So teamwork plays a key role in this task. We evaluate the agents over 100 trials each, so the performance is based on the metric fraction scored, i.e., the number of goals scored averaged over 100 trials. We were unable to test this setting in 1000 trials due to time and resource consumption. Each trial consists of a series of 25 games of half field offense. After each trial, the game, and the agents are restarted.

NPCs Team

In this setting, we test our algorithm PPAS and the Barrett et al. (2017) algorithm SPP playing with a team of 3 NPCs. In each trial, the agent is placed on a team randomly selected from the 5 teams

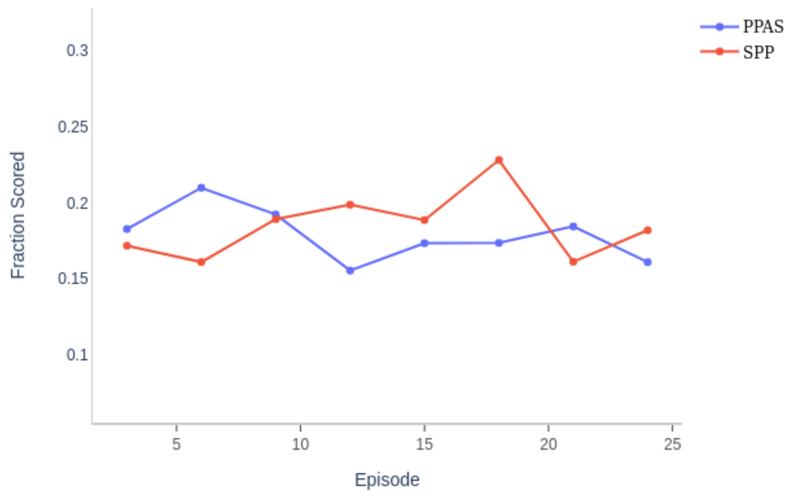

Figure 5.6: Scoring frequency in the full half field offense task (4vs5) during 25 games. Playing with 3 teammate, type NPC. These results are an average over 100 trials. Also the circles in the lines are the average fraction scored of the past three games, we did that to improve the visualization of the graph.

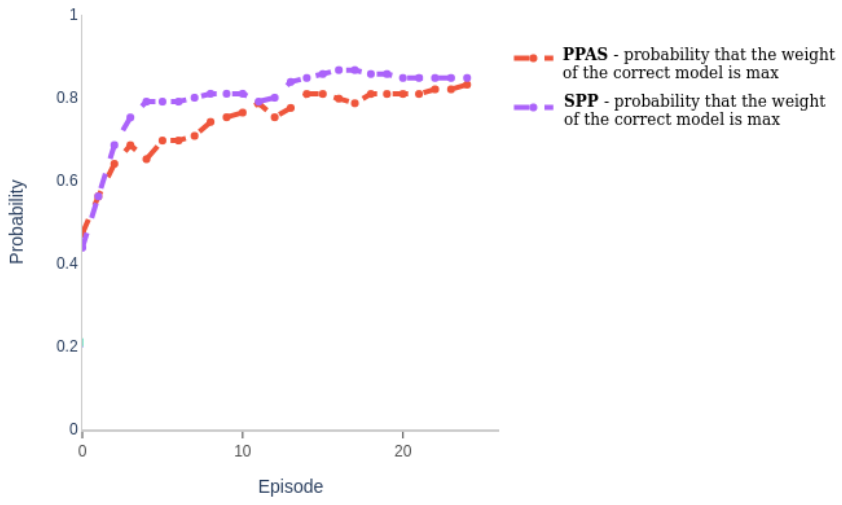

Figure 5.7: Evolution of the probability that the weight of the correct team has the maximum value from all the weights in vector w, when the ad hoc agent is playing in the full half field offense task (4vs5) during 25 games. Playing with 3 teammates, type NPC. These results are an average over 100 trials

described in Section 5.1.4, and they play a series of 25 games of half field offense. So in Figure 5.6 we can observe the fraction scored during 25 games by each algorithm, these results are an averaged of

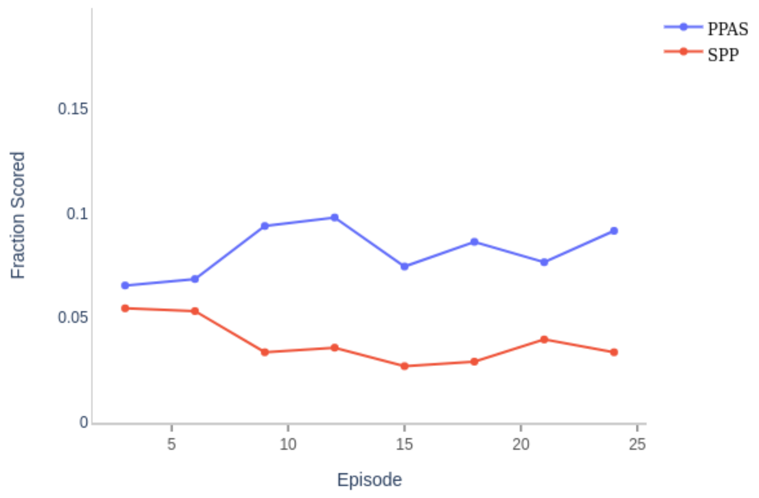

Figure 5.8: Scoring frequency in the full half field offense task (4vs5) during 25 games. Each team is composed by 4 ad hoc agents of the same type. These results are an average of 1000 trials. Also the circles in the lines are the average fraction scored of the past three games, we did that to improve the visualization of the graph.

100 trails. And in Figure 5.7 we have the evolution of weights of the correct team during the trials. As explained before, the higher the weight of the correct team, the better the agent will perform since it will choose the correct policy for that team.

In terms of results, in Figure 5.6, we see that both algorithms reach similar scores. It is curious to note that they outperform one another in different phases of the trial. PPAS can achieve better scores in the first games, but its performance decrease during the middle of the game, and slowly it improves during the rest of the game. On the contrary, SPP starts with the worst performance, but during the game, it is able to slowly improve the winning rate.

With this test, we can see clearly that both algorithms have stronger and weaker sides. On one side, PPAS can select good actions, even with high uncertainty about the team it is playing with, but takes it more time to correctly identify the correct team, as we can see in Figure 5.7, the weights about the correct team increase slowly than for the SPP algorithm. On the other side, SPP is unable to reach good performance while the correct team is unknown. But after 10-15 games, it is able to detect the correct team, in Figure 5.7 we can see that the ”probability that the weight of the correct model is max” is over 0.8, which corresponds to 80%, after 10 games, and then the performance gets better than PPAS.

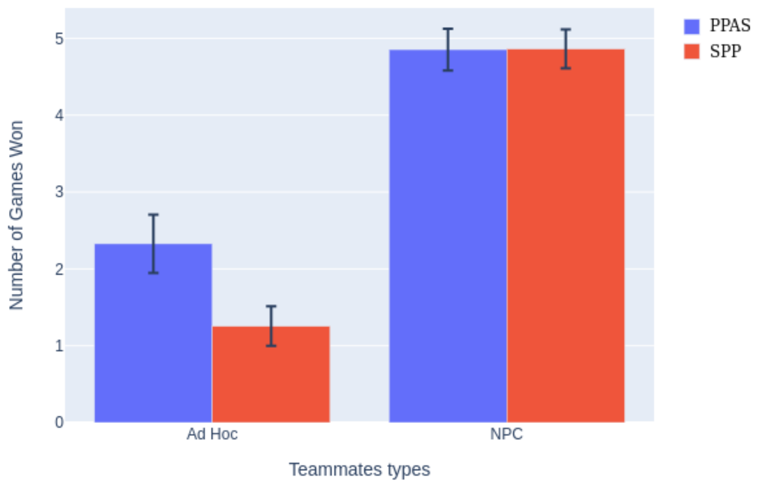

Figure 5.9: Number of win games in the full half field offense task (4vs5), during 25 games. These results are an average of 1000 trials

Ad hoc team

In this setting, we test our algorithm PPAS and the Barrett et al. (2017) algorithm SPP playing with a similar teammate. In each trial, the agent plays a series of 25 games of half field offense with other 3 ad hoc agents, which means that PPAS agent will play with other 3 PPAS agents, and SPP agent will play with another 3 SPP agents. So in Figure 5.8 we can observe the fraction scored during 25 games by each algorithm teams, these results are an averaged of 100 trails. In this setting, it does not make sense to analyze the evolution of the weights over the trial, since there is no correct team in the agents' library.

In terms of results, the team of agents PPAS is superior to a team SPP agents, as we can see in Figure 5.8. SPP and PPAS start with similar performance in the first six games, but from that point onward, PPAS is able to increase its performance during the game. In contrast, SPP performance decreases during the game. These results show us that PPAS is cable to reach good performances in the presence of non-stationary teammates. PPAS does not need to identify the team it is playing with in order to reach good results. The algorithm can take advantage of its past experiences and use them. Also, the results show that SPP needs to be able to identify the team, otherwise the team performance drops, which explains the bad results.

Discussion

In Figure 5.9, we can see the total number of win games in 25 episodes, in both scenarios, playing with a team of NPCs, and with a team of ad hoc agents. As we can see both algorithms reach very different results when playing with NPCs versus when playing with ad hoc agents.

When playing with NPCs, both algorithms reach basically the same number of won games. As we see in Figure 5.6, they do it through different methods, but at the end of the trial, the number of goals is similar.

However, when playing with ad hoc agents, which means playing with agents of the same type, the results are clearly different. PPAS reaches almost two times more wins than SPP. Our algorithm clearly outperforms the Barrett et al. (2017) algorithm in this setting.

The large difference between the results when playing with NPCs, versus when playing with the ad hoc team is due to two factors: i) The NPCs have built-in strategies to cooperate as a team. And the ad hoc agents are never trained as a team; ii) The ad hoc agents are always trying to identify the team they are playing with, which leads them to alternate between policies all the time.

5.2.3 Investigation Questions

We wanted to investigate three questions. The first, *Does Plastic-Policy perform poorly in the presence of non-stationary teammates?"*, addresses the drop of performance when playing in the presence of non-stationary teammates. As we suspected, the performance of standard methods can drop more than 50%. In the Figures [5.4, 5.9], we can see that in booth settings the original Plastic-Policy SPP performance is lower, when cooperating with non-stationary teammates (ad hoc agents), than when cooperating with stationary teams (NPCs).

The second question, *"In the presence of stationary teammates, does PPAS obtains the same performance as the original PLASTIC-Policy?"*, investigates whether our algorithm is still able to correctly identify and cooperate with stationary teams. As we expected, the PPAS reaches similar performances to the original Plastic-Policy SPP. In Figures [5.4, 5.9] we can see that SPP and PPAS reach similar results when playing with a stationary team (NPCs). Booth algorithms can identify the current teams, and get good results.

The last question, *"In the presence of non-stationary teammates, does PPAS outperforms the original PLASTIC-Policy?"* is also answered positively. Our algorithm is able to select actions that allow the team to complete the task efficiently. And, as expected, the original Plastic-Policy is unable to reach reasonable results when cooperating with teammates who do change their behavior during the episodes. This is more evident in the full version since the task is harder, it tests the agent cooperative capabilities even more, as we can see in Figures [5.8, 5.9] when playing with non-stationary teammates (Ad Hoc) our algorithm, PPAS, is superior.

On top of that, our agent proved to be able to reach good performances faster than SPP in every setting, with every team. And since the goal of the ad hoc teamwork problem is to cooperate on the fly, we conclude that our solution is able to reach this goal faster.

6 Conclusions and Future Work

Contents

The Ad Hoc Teamwork Problem has been around for a bit longer than a decade. Significant research has been done with the intent of creating an agent that can cooperate with a team of other agents on the fly, but this problem is far from solved In this work we look in detail into this problem and the state-of-the-art solutions.

One critical problem that we found in previous works is the inability to cooperate with agents who do not follow stationary behaviors. We proposed an PPAS to address this limitation. It was based on the algorithm Plastic-Policy, developed by Barrett et al. (2017), which, to the best of our knowledge, is still one of the state-of-the-art solutions.

In our algorithm, we use PLASTIC-Policy architecture as the basis, so our algorithm collects past experiences with different teams in the form of policy models and team models. And by using transfer learning techniques and adversarial algorithms our agent is able to identify the teams and select actions on the fly, using its past experiences. Even if the team is unknown, and does not follow a stationary behavior, our algorithm is able to efficiently select actions that maximize the reward.

In our work, we showed the limitations of solutions such as PLASTIC-Policy (Barrett et al., 2017) when cooperating with non-stationary teammates. We also showed that our algorithm PPAS obtains the same performance as the original as PLASTIC-Policy when cooperating with known stationary teammates. Finally, we showed that PPAS outperforms the original PLASTIC-Policy when cooperating with non-stationary teammates, our algorithm is able to select actions that allow the team to complete the task efficiently. These results are more evident in complex scenarios, such as the full version of half field offense. On top of that, our agent proved to be able to reach good performances faster than the original PLASTIC-Policy in every setting, with every team.

We tested our algorithm in a complex domain, half field offense, with different levels of difficulty, which establishes the efficiency of our solution.

6.1 Future Work

As pointed out before, the Ad Hoc Teamwork Problem is far from solved, and there are still lots of challenges to work on.

Transfer learning is a possible avenue to solve this problem. In the future, it would be interesting to investigate parameterized types, instead of discrete types of agents. It would help to adapt to unknown teammates, and even when cooperating with known teammates, use insights from other teams that could help to improve cooperation.

Another interesting problem would be to investigate how to identify the different levels of behavior. Since teammates can display multiple behaviors, from static policies to learning in real-time.

It would also be interesting to add a reward to our algorithm. To detect the better team, we currently use the similarity between teams, so we do not look into the actual score of the game. Using the reward is a hard approach since it would take a significant amount of games to actually do it. And also, in complex scenarios as the full half field offense version, the result may not depend on our agent.

Another interesting approach would be to use a temporal model, able to look into the sequence of transitions, instead of the last one. By using beliefs, we are already using past history, but we are not using it to its full potential.

This work showed us that the ad hoc teamwork in real-world conditions, with continuous action-spaces and partial observability, which means the agent is only able to observe the state of the world, without any insight about his teammates, is very difficult to solve. Humans have solved this problem millennia ago, so we have to keep looking at ourselves in order to try to solve these reinforcement learning challenges.

Bibliography

Stefano V. Albrecht and S. Ramamoorthy. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. In *AAMAS*, 2013.

Stefano V. Albrecht and Peter Stone. Reasoning about hypothetical agent behaviours and their parameters. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, page 547–555, 2017.

Stefano V. Albrecht and Peter Stone. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Computer Science, Mathematics*, abs/1709.08071, 2018.

F. Alves, A. Pereira, J. Barbosa, and P. Leitão. Scheduling of home health care services based on multi-agent systems. In *PAAMS*, 2018.

Samuel Barrett, Avi Rosenfeld, Sarit Kraus, and Peter Stone. Making friends on the fly: Cooperating with new teammates. *Artificial Intelligence*, 242:132–171, 2017.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Dkebiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal J'ozefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pond'e de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *Computing Research Repository*, abs/1912.06680, 2019.

Michael Bowling and P. McCracken. Coordination and adaptation in impromptu teams. In *AAAI*, 2005.

Sébastien Bubeck and N. Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Found. Trends Mach. Learn.*, 5:1–122, 2012.

N. Cesa-Bianchi and G. Lugosi. Prediction, learning, and games. 2006.

Doran Chakraborty and Peter Stone. Cooperating with a markovian ad hoc teammate. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*, page 1085–1092, 2013.

Shuo Chen, Ewa Andrejczuk, Zhiguang Cao, and J. Zhang. Aateam: Achieving the ad hoc teamwork by employing the attention mechanism. In *AAAI*, 2020.

Felipe Leno da Silva and Anna Helena Reali Costa. A survey on transfer learning for multiagent reinforcement learning systems. *J. Artif. Intell. Res.*, 64:645–703, 2019.

Matthew Hausknecht, Prannoy Mupparaju, Sandeep Subramanian, Shivaram Kalyanakrishnan, and Peter Stone. Half Field Offense: an environment for multiagent learning and ad hoc teamwork. In *AAMAS Adaptive Learning Agents Workshop*, pages 36–42, 2016.

Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity. *Computing Research Repository*, abs/1707.09183, 2017a.

Pablo Hernandez-Leal, Yusen Zhan, Matthew E. Taylor, Luis Enrique Sucar, and Enrique Munoz de Cote. Efficiently detecting switches against non-stationary opponents. *Autonomous Agents and Multi-Agent Systems*, 31(4):767–789, 2017b.

Shivaram Kalyanakrishnan, Yaxin Liu, and Peter Stone. Half field offense in robocup soccer: A multiagent reinforcement learning case study. In *RoboCup*, 2006.

L. Kocsis and Csaba Szepesvari. Bandit based monte-carlo planning. In *ECML*, 2006.

Joel Z. Leibo, Edward Hughes, Marc Lanctot, and Thore Graepel. Autocurricula and the emergence of innovation from social interaction: A manifesto for multi-agent intelligence research. *Computing Research Repository*, abs/1903.00742, 2019.

William Macke, Reuth Mirsky, and Peter Stone. Expected value of communicationfor planning in ad hoc teamwork. In *Proceedings of the 35th Conference on Artificial Intelligence (AAAI)*, February 2021.

Francisco S. Melo and Alberto Sardinha. Ad hoc teamwork by learning teammates' task. *Autonomous Agents and Multi-Agent Systems*, 30(2):175–219, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen. King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518: 529–533, 2015.

Rodrigo Monteiro. Intelligent agents coordination in ad hoc teams. Master's thesis, Instituto Superior Técnico, 2016.

N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani. Algorithmic game theory. 2007.

Manish Ravula, Shani Alkoby, and Peter Stone. Ad hoc teamwork with behavior switching agents. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 550–556, 2019.

H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58:527–535, 1952.

Gonçalo Rodrigues. Ad hoc teamwork with unknown task model and teammate behavior. Master's thesis, Instituto Superior Técnico, 2018.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

N. Schurr, J. Marecki, Milind Tambe, P. Scerri, N. Kasinadhuni, and John P. Lewis. The future of disaster response: Humans working with multiagent teams using defacto. In *AAAI Spring Symposium: AI Technologies for Homeland Security*, 2005.

P. Stone. Autonomous learning agents: Layered learning and ad hoc teamwork. In *AAMAS*, 2016.

P. Stone, G. Kuhlmann, Matthew E. Taylor, and Y. Liu. Keepaway soccer: From machine learning testbed to benchmark. In *RoboCup*, 2005.

Peter Stone, Gal A. Kaminka, Sarit Kraus, and Jeffrey S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, page 1504–1509, 2010.

S. A. Stone. Ad hoc teamwork modeled with multi-armed bandits: An extension to discounted infinite rewards. 2011.

Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 16:285–286, 1988.

Oriol Vinyals, Igor Babuschkin, Wojciech Marian Czarnecki, Michaël Mathieu, Andrew Joseph Dudzik, Junyoung Chung, Duck Hwan Choi, Richard W. Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575:350–354, 2019.

7 Ah hoc agent implementation in HFO

The goal was to develop an Ad Hoc agent similar to the one presented by Barrett et al. (2017). However, some implementation details were missing in their work, so we had to make some assumptions. In this section, we describe our ad hoc agent implementation from features, actions and policy model.

7.1 Auxiliary Metrics

Some metrics were created to help the action executing some actions. The most important one is ”team lost ball”, is a flag used to signalize if any teammate lost the possession of the ball. This flag can mean the teammate passed the ball, shoot the ball, lost the ball while dribbling, a defender intercepted the ball. The environment do not give us this information, so when our agent do not has the ball, if any teammate is closer than 0,15 (calculated throw euclidean distance) we assume that the teammate has the ball.

7.2 Features

We introduced the same features used by Barrett et al. (2017). As basis to our features, we used the high-level state features given by the environment. From that features we selected eight continuous features with an additional five for each teammate (8 + 5T). So, in the limited version game (1 teammate) the agent has access to 13 features, but in the full version game (3 teammates) the agent has access to 23 features. The features are:

- Agent's features:
    1. X position (*float [-1, 1]*): the agent's x position on the field;
    2. Y position (*float [-1, 1]*): the agent's y position on the field;
    3. Orientation (*float [-1, 1]*): the direction that the agent is facing;
    4. Goal opening angle (*float [-1, 1]*): the size of the largest open angle of the agent to the goal;
    5. Distance nearest opponent (*float [-1, 1]*): distance to the closest opponent;
    6. Has Ball flag (*int (-1, 1)*): if the agent has ball, the value is 1, otherwise the value is -1;
    7. Ball X Position (*float [-1, 1]*): the ball's x position on the field;
    8. Ball Y position (*float [-1, 1]*): the ball's y position on the field;
- For each Teammate i:
    1. X position (*float [-1, 1]*): the teammate's x position on the field;
    2. Y position (*float [-1, 1]*): the teammate's y position on the field;
    3. Goal opening angle (*float [-1, 1]*): the teammate's goal opening angle;
    4. Distance to nearest opponent (*float [-1, 1]*): the distance from the teammate to the closest opponent;
    5. Agent's pass opening angle to teammate i (*float [-1, 1]*): the open angle available to pass to the teammate;

All these features are directly given by the environment, no pre-processing was made.

7.3 Actions

HFO provides multiple action spaces. The actions available to our agent were developed using high-level actions and mid-level environment actions. The goal was to to re-create the actions used by Barrett et al. (2017).

We have to make a set of assumptions when implementing these actions, the number of steps each action takes, which environment actions to use in order to get the discrete actions, and when to stop the actions. We created 11 discrete actions for the limited version game (1 teammate), and 13 discrete actions for the full version game (3 teammates). We will describe in detail the behaviour of each action. The actions when the agent has ball:

Shoot: There is an high level action SHOOT, which we assumed that the Plastic authors has used. This High Level actions is described as ”Executes the best available shot. This command only works when the agent has the ball.” There is also other lower level action KICK_TO, described as ”Kicks the ball to the specified target point with the desired speed. Valid values for target $x, y \in [-1, 1]$ and $speed \in [0, 3]$. However, we found some problems with these action

- SHOOT: When in optimal conditions, what means, when the agent is near goal and it has a good angle to score, the action works fine. However this actions fails with a very high frequency. For example when the agent is not near the goal, or the agent has a bad angle, this actions does nothing, which is hard for the agent to learn.
- KICK_TO: When kick strength is higher than 2.4, the agent frequently kicks the ball to random coordinates. We looked for any documentation about this problem, but we found out nothing.

Our Implementation: So we decided to create an hand-made actions, which can shoot to the middle, top or bottom of the goal, according to the best angle. To create this action we used the KICK_TO action. And due to the kick strength problem we used a strength of 2.3 But this lead to weaker kicks, which reduces the accuracy o the kick.

Short Dribble: Executes the High Level action DRIBBLE defined as ”Advances the ball towards the goal using a combination of short kicks and moves.” Executes the action DRIBBLE during 4 steps. Although, when the actions finishes, if the agent does not has the ball, the agent executes GO_TO_BALL until, gets the ball again.

Long Dribble:Executes the High Level action DRIBBLE defined as ”Advances the ball towards the goal using a combination of short kicks and moves”. This action is executed during 10 steps. Although, when the actions finishes, if the agent does not has the ball, the agent executes GO_TO_BALL until, gets the ball again.

Pass$_0$, **Pass**$_1$, **Pass**$_2$: There is an high level action PASS, which we assumed that the Plastic authors had used. This High Level actions is described as ”Passes to the teammate with the provided uniform number. Does nothing if the player does not have control of the ball or the requested teammate is not detected.” However, we found some problems with these action

- PASS: When in optimal conditions, what means, when the agent is near the teammate, has a good angle to pass, and the teammate is looking at the ball, the action works fine. However this actions fails otherwise. This kind of behavior makes it hard for the agent to learn.

Implementation: The agent executes the action PASS three times. If it fails, the agent kicks the ball in the teammates direction using KICK_TO, to the teammates coordinates with a strength of 1.7.

When the agent does not have the ball, it can execute the actions. Note: All the actions bellow are interrupted if the flag "team lost ball" is true: **Stay in the Current Position**: Executes the action NOOP defined as "Indicates that the agent should take no action". This action is executed during 4 steps.

Move towards the ball: Firstly we used an high level action Go_To_Ball, described as "Makes the agent go towards the ball.". But currently we are using the action INTERCEPT, an lower level action described as "Moves to intercept the ball, taking into account the ball velocity. More efficient than chasing the ball." Problems found with the action GO_TO_BALL: This action works fine, and initially we believed that this was the action used by the Plastic Action. Although during game our agent was having difficulties catching the ball, after the teammate passes it. So we saw the videos of the Plastic agent carefully, and we noticed that their agent took into consideration the speed and direction of the ball, something that this action is unable to do. And also, their agent was faster. So we started using the action INTERCEPT, which improved the performance of our agent. In terms of implementation, the agent executes the action INTERCEPT 10 times. Although, if the agent is very close to the teammate, and the teammate has the ball, the action stops. The idea behind it is to not allow the agent to steal the ball to the teammate.

Move towards the opposing goal: Currently we are using the action mid level action MOVE_TO, described as "Moves to the specified target point using the max dash speed. Valid values for target $x, y \in [-1, 1]$.". In terms of implementation: The agent executes the action MOVE_TO 4 times to the coordinates (0.5, 0). Although, if the agent is very close to that position the action stops.

Move towards the nearest teammate: Currently we are using the action mid level action MOVE_TO, described as "Moves to the specified target point using the max dash speed. Valid values for target $x, y \in [-1, 1]$." In terms of implementation: The agent executes the action MOVE_TO 4 times to the teammate coordinates (0.5, 0). Although, if the agent is very close to that position the action stops.

Move away from the nearest teammate: Currently we are using the action mid level action MOVE_TO, described as "Moves to the specified target point using the max dash speed. Valid values for target $x, y \in [-1, 1]$.". In terms of implementation: The agent calculates the opposite vector between itself and the teammate, and executes the action MOVE_TO 4 times to in that direction. Although, if the agent is very close to that position the action stops.

Move towards the nearest opponent: Currently we are using the action mid level action MOVE_TO, described as "Moves to the specified target point using the max dash speed. Valid values for target $x, y \in [-1, 1]$". In terms of implementation: The agent executes the action MOVE_TO 4 times to the nearest opponent coordinates. Although, if the agent is very close to that position the action stops.

Move away from nearest opponent: Currently we are using the action mid level action MOVE_TO, described as ”Moves to the specified target point using the max dash speed. Valid values for target $x, y \in [-1, 1]$”. In terms of implementation: The agent calculates the opposite vector between itself and the opponent, and executes the action MOVE_TO 4 times to in that direction. Although, if the agent is very close to that position the action stops.

7.4 Learning Policy Model

In our work we used neural networks to learn the policies to cooperate with the teams. These neural networks were implemented using Keras, in Python 3.6. The Neural networks are a sequential neural networks. The Neural network is defined as follow:

- Input: Number of features (depending on the HFO version);
- Hidden Layer: 512 Dense layer;
- Activation: Relu function;
- Hidden Layer: 512 Dense layer;
- Activation: Relu function;
- Hidden Layer: 512 Dense layer;
- Activation: Relu function;
- Output: Number of actions (depending on the HFO version);

The algorithm used was Deep Q-learning (Mnih et al., 2015). So we have two neural networks. The first one is used to predict the actions, and it is updated every 500 steps. The second one is updated every step. So the agent learns off-policy, and the actions are selected using an e-greedy approach. The hyper-parameters:

- Learning rate: 0.00025;
- Epsilon: [0.8 - 0.05];
- Discount factor: 0.995;