TÉCNICO LISBOA

# Playing Soccer with Unknown Teammates

**João Francisco Lopes Pirralha**

Thesis to obtain the Master of Science Degree in

**Information Systems and Computer Engineering**

Supervisor: Prof. José Alberto Rodrigues Pereira Sardinha

**January 2021**

# Acknowledgments

I would like to thank my supervisor, Professor Alberto Sardinha, for guiding me through this thesis. His guidance and counseling was invaluable. I would also like to thank my family for supporting, encouraging and providing me this opportunity. I feel very lucky to have been able to get here. I thank my friends for helping me keep motivated and sane. Without them I feel I would have had a breakdown. I also thank my colleagues for their help throughout the degree. They helped me understand and do many things. Finally, I would like to thank all the Professors and teachers I had throughout my life. They helped shaping me in who I am today.

# Abstract

Robotic soccer allows researchers to attempt to solve many challenges in the field of artificial intelligence. One such challenge is collaboration with unknown teammates, without any sort of pre-coordination, which is known as ad hoc teamwork. Advances in ad hoc teamwork enable collaboration in multi-agent systems to be more robust and versatile compared to traditional coordination mechanisms, as it addresses situations such as collaboration with agents developed by different people, with legacy agents that cannot be modified and even with humans. Some current work in the literature attempts to address this challenge by reusing experience with past teammates to adapt to new ones, for example by acting using previously learned policies. This thesis extends the state-of-the-art approach in order to also deal with unknown teammates that might be significantly different from past teammates, while still leveraging what was previously learned. To achieve this, a current unidentified team is detected either as being a known team or unknown, by observing if the team's behavior is consistently similar to the past behavior of a known team. If it is detected as unknown, the agent selects the previously learned policy whose team it considers to be most similar to the unknown team, which is then improved online, as a source for parameter sharing transfer learning.

# Keywords

Ad hoc teamwork; Artificial intelligence; Autonomous agents and multiagent systems; Reinforcement learning.

# Resumo

O futebol robótico permite a investigadores tentar resolver muitos desafios no campo da inteligência artificial. Um desses desafios é a colaboração com colegas de equipa desconhecidos, sem qualquer tipo de pré-coordenação, o que é conhecido por *ad hoc teamwork*. Avanços em *ad hoc teamwork* possibilitam que a colaboração em sistemas multi-agente seja mais robusta e versátil em comparação a mecanismos de coordenação tradicionais, pois aborda situações como colaboração entre agentes desenvolvidos por pessoas diferentes, com agentes descontinuados que já não podem ser alterados e mesmo com humanos. Algum do atual trabalho na literatura tenta resolver este desafio reutilizando experiência com colegas de equipa antigos para se adaptar a colegas atuais, por exemplo usando políticas previamente aprendidas para agir. Esta dissertação estende a abordagem estado-da-arte de modo a também lidar com colegas desconhecidos que podem ser significativamente diferentes de colegas antigos, aproveitando o que foi previamente aprendido. Para o fazer, uma equipa atual não identificada é detetada como sendo uma equipa conhecida ou desconhecida, observando se o seu comportamento é consistentemente semelhante ao comportamento passado de uma equipa conhecida. Se for detetada como desconhecida, o agente seleciona a política previamente aprendida cuja equipa considera ser mais semelhante à equipa desconhecida, que é então melhorada *online*, como uma fonte para *parameter sharing transfer learning*.

# Palavras Chave

Agentes autónomos e sistemas multi-agente; Aprendizagem por reforço; Coordenação ad hoc; Inteligência artificial.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Acronyms

| | |
|---|---|
| **CNN** | Convolutional Neural Network |
| **CPD** | Change Point Detection |
| **DQN** | Deep Q-Network |
| **EWF** | Exponentially Weighted Forecaster |
| **FQI** | Fitted Q Iteration |
| **HFO** | Half Field Offense |
| **MCTS** | Monte Carlo Tree Search |
| **MDP** | Markov Decision Process |
| **MLP** | Multi-Layer Perceptron |
| **MSE** | Mean Squared Error |
| **OPAT** | Online Planning for Ad Hoc Agent Teams |
| **PLASTIC** | Planning and Learning to Adapt Swiftly to Teammates to Improve Cooperation |
| **PWA** | Polynomial Weights Algorithm |
| **ReLU** | Rectified Linear Unit |
| **SGD** | Stochastic Gradient Descent |
| **UCT** | Upper Confidence Bounds applied to Trees |

# 1 Introduction

**Contents**

## 1.1 Motivation

As autonomous agents increase in number, robotic or virtual, more situations arise where multiple agents need to cooperate as a team in order to achieve their objectives. However, it is not always possible to know in advance the teammates which an agent will end up with. As such, it is desirable to have a dynamic and versatile mechanism for collaboration that does not rely on pre-coordination.

One such situation in the human world is pick-up soccer, where players, that do not belong to a fixed team, gather to play a match. Players often play with others whom they had limited or no contact, so they need to quickly observe each other's capabilities and adapt. This situation could also happen in robotic soccer. Robotic soccer teams usually have communication and coordination mechanisms, but we can consider a challenge where multiple independent robots are gathered to play a soccer match with random teammates, lacking common mechanisms, e.g. the RoboCup drop-in challenge [1]. Advances in such a challenge could be relevant for other domains, such as rescue robots from different sources that need to work together during a catastrophe, which is a prominent example in the literature [2]. Such advances could also be useful for having robust systems where agents can collaborate with older and limited agents that cannot be modified, or even collaborate with humans.

These situations are framed within the problem of ad hoc teamwork [2], where agents need to cooperate with unknown teammates without relying on previously developed mechanisms for coordination or communication. These agents, often called ad hoc agents, must be able to quickly find an adequate strategy to act efficiently. Unlike multiagent learning, ad hoc agents are developed independently from their teammates.

Some of the current work in the literature [3] attempts to address the ad hoc teamwork problem by leveraging prior knowledge about past teammates. However, new teammates may be arbitrarily different from past ones; thus, using previously learned behavior may be ineffective. We believe that the ad hoc teamwork approach is still useful in this situation, as it allows to reuse knowledge about the dynamics of the environment and teammates, enabling to learn to coordinate faster than just a pure reinforcement learning [4] approach.

## 1.2 Problem

This thesis addresses a research problem where an ad hoc agent may encounter unidentified teammates which may be known or unknown, when performing a task. The task involves playing a game of Half Field Offense (HFO) [5], described in Section 2.4. In this environment, one team of agents attempts to score a goal and another defends. The possible teams are: `AUT MasterMinds`, `Base`, `Cyrus`, `Gliders`, `HELIOS` and `YuShan`. These teams are those used in [3], with one missing exception (`Axiom`) as there were technical difficulties in successfully executing it. The ad hoc agent joins an unidentified teammate belonging to one of those teams in the offense and both play against two defensive HELIOS players, one of which is a goalkeeper.

## 1.3 Contributions

The contributions of this thesis are the following:

1. Two methods for identifying known teams: one based on comparing predictions of the next state according to historical data of each of the known teams to the real next state (similar to the presented in [3], but using a model based on a Multi-Layer Perceptron (MLP)); and a second based on discriminating each team directly also according to historical data.

2. An extension to the methods of identifying known teams to also detect unknown teams, where the agent observes if the unidentified team is consistently identified as being the same known team across multiple observations – if not, then the team probably is unknown;

3. The use of parameter sharing transfer learning techniques in order to adapt a policy learned from one team to a different one, online. As far as we are aware, this is a novel use of transfer learning within the field of ad hoc teamwork.

## 1.4 Document Overview

This document is organized as follows. Background knowledge, about relevant algorithms for the solution and the environment, is presented in Chapter 2. Existing work related to the problem of this thesis is described in Chapter 3. The solution is presented in Chapter 4. The obtained results are presented in Chapter 5. Finally, Chapter 6 contains a brief summary of this thesis and presents its limitations and possibilities for future work.

# 2 Background

**Contents**

This section presents background knowledge required for the solution developed in this thesis: in Section 2.1 are presented Markov Decision Processes (MDPs), a formalization of sequential decision making problems, such as the problem of this thesis; Section 2.2 presents neural networks, which are a widely used method for mapping inputs to outputs; and Section 2.3 presents the Deep Q-Network (DQN) algorithm, which uses neural networks to solve MDPs. Additionally, Section 2.4 presents the environment which is used throughout this thesis.

## 2.1 Markov Decision Processes

In a Markov Decision Process (MDP) [4], an agent interacts with an environment in a sequence of discrete time steps, making sequential decisions (Figure 2.1). At each time step, the agent receives a state from the environment and acts based on it. In the next time step, the environment reacts to the agent's previous action, which receives a reward (or cost) and a new state. An MDP is thus comprised of a state space $\mathcal{S}$, which is the set of all possible states; an action space $\mathcal{A}(s)$, which is the set of all possible actions for any given state $s$; environment's dynamics $P(s'|s, a)$, which govern the transition to a new state $s'$ given a state $s$ and an action $a$; and a reward function $r(s, a)$, which provides a reward given a state $s$ and an action $a$. States must verify the Markov property: a new state must only depend on the state and action in the previous time step.

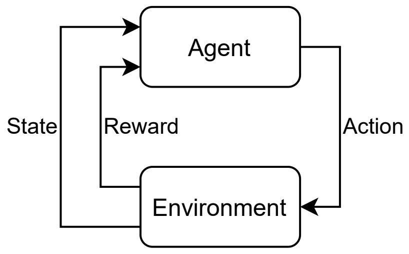

**Figure 2.1:** The interaction between an agent and the environment in an MDP.

The objective when solving an MDP is to maximize the cumulative rewards (or minimize the cumulative costs) that the agent receives during its interaction with the environment. Some states may be terminal. When the agent reaches a terminal state, its interaction with the environment ends, ending an episode, and the agent starts over in the next time step, which is completely independent. However, when solving an MDP it is usually assumed that the agent interacts with the environment forever. This would result in a non-converging infinite sum of rewards, which would not allow to solve it. To make this sum converge to a finite value, a discount factor $\gamma$ is used, with $0 < \gamma < 1$. This parameter makes the reward $r$ at time step $t + k$ to be valued as $r\gamma^k$ when the agent is at time step $t$. A larger $\gamma$ makes the agent give more value to distant rewards.

Solving an MDP consists in determining a policy (a function that, given a state, gives the probability of selecting each action) that attempts to maximize the cumulative reward. This often involves calculating or estimating the discounted sum of rewards (value) for a given state or state-action pair. The policy is then given by selecting the action(s) that maximize(s) the expected value, which assumes the agent follows this policy in subsequent time steps. The value of a state $s$ under policy $\pi$ is given by (2.1) [4], where $\mathbb{E}_\pi$ denotes the expected reward if the agent follows policy $\pi$. The value of a state-action pair $\langle s, a \rangle$ under policy $\pi$, known as the Q-value, is given by (2.2) [4].

$$V_\pi(s) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| S_t = s \right], \forall s \in \mathcal{S} \tag{2.1}$$

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| S_t = s, A_t = a \right] \tag{2.2}$$

If the environment's dynamics and reward function are known, dynamic programming algorithms such as value iteration and policy iteration [4] can be used to solve the MDP. If they are unknown, as in the problem of this thesis, reinforcement learning [4] methods can be used instead. One approach is to model the environment's dynamics and reward function using transition samples (with the state, the agent's action, the reward and the next state – $\langle s_t, a_t, r_t, s_{t+1} \rangle$) collected while the agent interacts with the environment. These models can then be used with a dynamic programming algorithm, interleaving model updates with it. Another approach is to estimate the Q-values given a state, using collected samples, which is known as model-free reinforcement learning.

Q-learning [6] is a model-free reinforcement learning algorithm that uses the update rule in (2.3) in order to learn $Q_\pi(s, a)$, where $0 < \alpha \leq 1$ is the learning rate. If every state-action pair is visited infinitely, Q-learning converges to $Q_*(s, a)$, resulting in the optimal policy $\pi_*$, which is better or equal than all other policies ($Q_*$ denotes the Q-values of $\pi_*$). The policy used to collect the samples may be different from its target policy $\pi_*$, so Q-learning is classified as an off-policy method. In order to visit every state-action pair, it is necessary to also explore new pairs instead of always following (exploiting) the policy learned so far. $\epsilon$-greedy is an exploration strategy where the agent explores a random action with a probability of $\epsilon$ and exploits the policy (selects the action with the highest Q-value) with a probability of $(1 - \epsilon)$.

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha \left[ r(s_t, a_t) + \gamma \max_{a' \in \mathcal{A}(s_{t+1})} Q_\pi(s_{t+1}, a') - Q_\pi(s_t, a_t) \right] \tag{2.3}$$

## 2.2 Neural Networks

Supervised learning [7] is a branch of machine learning where a model learns to map inputs to outputs, given labeled samples. Neural networks are models that accomplish this by passing data through a graph of interconnected nodes, called units. Each unit has multiple inputs (real numbers) and a single output. To obtain the output, it calculates the weighted sum of its inputs and usually applies a function, called activation function. The feed-forward neural network (also known as multi-layer perceptron) is the most common type of neural network, where the units are organized in successive layers that are processed in sequence. The first layer, known as input layer, simply passes the inputs to the second layer. The last layer is known as the output layer and the layers in between are known as hidden layers. Fully connected layers are layers where every unit's inputs are all of the outputs of the previous layer. Other types of layers exist, such as convolutional layers, which use a regularized structure to exploit features in data organized dimensionally (e.g. image-based data). Figure 2.2 presents an example of a feed-forward neural network.

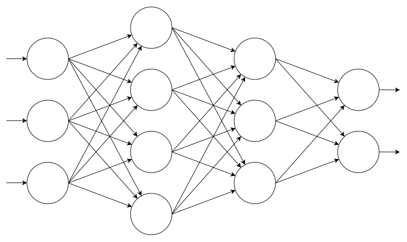

**Figure 2.2:** A feed-forward neural network consisting of fully-connected layers. From left to right: the input layer has three units that simply pass their inputs forward, the first hidden layer has four units, the second hidden layer has three units and the output layer has two units.

Training a neural network involves iteratively optimizing a metric, given by a loss function which quantifies the difference or error rate between a ground truth $Y$ and a prediction $\hat{Y}$. Typical loss functions include the binary cross-entropy (2.4) for classification tasks and the Mean Squared Error (MSE) (2.5) for regression tasks.

$$H(Y,\hat{Y}) = -\frac{1}{n}\sum_{i=1}^{n}\left[Y_i log(\hat{Y}_i) + (1-Y_i)log(1-\hat{Y}_i)\right] \quad (2.4) \qquad MSE(Y,\hat{Y}) = \frac{1}{n}\sum_{i=1}^{n}(Y_i-\hat{Y}_i)^2 \quad (2.5)$$

The backpropagation [8] algorithm is usually used to obtain the gradient of the loss function for every weight, starting with the output layer and going backwards. These gradients can then be used in an optimization algorithm such as Stochastic Gradient Descent (SGD) and variants such as Adam [9] to update the weights of the network. These algorithms usually operate in batches of samples instead of the whole dataset at once. One important parameter of these optimization algorithms is the learning rate, which adjusts the magnitude of the weights' update in one training step. If it is too small the network will learn slowly, while if it is too large the training will be unstable and the network will not converge to a solution.

The activation function allows the network to model non-linearities. Some common activation functions include the logistic function (also known as sigmoid) (2.6), the hyperbolic tangent (2.7), the Rectified Linear Unit (ReLU) (2.8) and softmax (2.9) for multi-class classification output layers (this activation operates on multiple units at once instead of a single one). The ReLU is one of the most widely used activations for hidden layers.

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.6}$$

$$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{2.7}$$

$$f(x) = max(0, x) \tag{2.8}$$

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.9}$$

The weights of the network need to be carefully initialized in order for the network to learn optimally. A naive method is to use random weights, but some methods have been devised for specific types of activation functions: Glorot [10] initialization for layers using activations such as logistic and hyperbolic tangent and He [11] initialization for layers using ReLU and derivatives.

Neural networks trained for one dataset can be adapted to a different dataset via transfer learning techniques. A common technique is to share the parameters of a source network with a target network [12], which often involves freezing the shallower layers[1] (not allowing their weights to be updated). The network is then fine-tuned on the new dataset, progressively unfreezing the layers. For this technique the inputs must be compatible between the datasets, but the output layer may be replaced with a new one.

## 2.3 Deep Q-Networks

MDPs often have such large or infinite state spaces that using algorithms such as Q-learning becomes impractical, being necessary to use function approximation [4] methods, which attempt to construct an approximate function (such as $Q(s, a)$) from limited examples. One such method is the Deep Q-Network (DQN) [13], which employs neural networks in order to perform Q-learning. It is suitable for problems where there are only discrete actions (without parameters). DQN was studied in a video game setting where the input (state) is an image, so it first uses convolutional layers to extract features. They are then followed by a fully connected layer consisting of 512 ReLU and a fully connected linear layer for the output (Q-values for each possible action).

Essentially, given a transition sample $\langle s_t, a_t, r_t, s_{t+1}\rangle$ collected in time step $t$, it updates the network targeting the values given by (2.10), using the loss in (2.11), where $Q(s, a)$ is the predicted value for the state-action pair $\langle s, a\rangle$ by the online network $Q$ and $L$ is a loss function such as the MSE (for now assume $\hat{Q} = Q$). However, this would have difficulty converging due to training instability. To address this, the authors make use of several mechanisms.

$$y_t = \begin{cases} r_t & \text{if last time step of episode} \\ r_t + \gamma \max_{a' \in \mathcal{A}(s_{t+1})} \hat{Q}(s_{t+1}, a') & \text{otherwise} \end{cases} \tag{2.10}$$

$$loss_t = L(y_t, Q(s_t, a_t)) \tag{2.11}$$

[1] Layers that are close to the input layer.

The first mechanism is the experience replay memory (or buffer). At each time step, the agent stores the observed transition in memory, instead of training right away with it. When training (done every certain number of time steps), it draws a randomly sampled batch from the replay memory and updates the network. This allows for greater data efficiency (using each transition more than once); allows to reduce the correlation between samples decreasing the variance of the updates; and allows to stop prejudicial feedback loops where one action could influence the following transitions to always favor the same action.

In order to further stabilize DQN, a second network $\hat{Q}$, called the target network, is used to obtain the targets in (2.10). This network is periodically synchronized with the online network $Q$. Without it, the authors state DQN is more susceptible to oscillations and divergence.

Finally, the authors clip the errors in the loss to $[-1, 1]$. They use the squared error for errors between $[-1, 1]$ and the absolute error for errors outside it. It increases the training stability as outliers have a limited impact. The Huber loss [14] (2.12) with $\delta = 1$ can be used for this purpose.

$$L(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| \leq \delta \\ \delta|y - \hat{y}| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases} \tag{2.12}$$

Several extensions attempting to enhance DQN's performance have been proposed. Double DQN [15] adapts Double Q-learning [16], a technique to prevent overestimating the Q-value in Q-learning, to DQN. It modifies the targets (2.10) to use the action that has the highest value according to the online network, but with the value of the target network (2.13). Prioritized experience replay [17] modifies replay in order to replay more frequently transitions that are deemed more important according to the last temporal-difference error ($|y - \hat{y}|$). Dueling networks [18] is a modification to the DQN network architecture where the fully connected layers split in two separate streams, where one estimates the value ($V$) of the state and another the advantage ($A$) (2.14) of each action relatively to the value of the state. These streams are then merged using (2.15). The authors state this allows for a better evaluation when there are many actions with similar values. Rainbow [19] combines these and other improvements (multi-step learning, distributional reinforcement learning [20] and noisy networks [21]) to achieve the state-of-the-art DQN. Lanctot *et al.* [22] present how to handle illegal actions (that the agent cannot perform given a state) in Q-learning, by forcing the probability of illegal actions to be zero when acting and ignoring them when determining $\max_{a' \in \mathcal{A}(s_{t+1})}$ during the update. This can be extended to DQN.

$$y_t = \begin{cases} r_t & \text{if last time step of episode} \\ r_t + \gamma \hat{Q}(s_{t+1}, \operatorname{argmax}_{a' \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a')) & \text{otherwise} \end{cases} \tag{2.13}$$

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \tag{2.14}$$

$$Q(s_t, a_t) = V(s_t) + A(s_t, a_t) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A(s_t, a') \tag{2.15}$$

## 2.4 Half Field Offense

RoboCup [23] is a soccer competition played by autonomous agents that has several leagues. It has several environments for researchers to empirically test and validate ideas, such as algorithms and agent architectures. While it is best known for its robotic leagues, it also provides simulated ones, namely the 2D subleague of the RoboCup Simulation League.

To provide a more tractable problem, Stone *et al.* proposed a subtask of the 2D Simulation League called Keepaway [24]. In this task, one team attempts to keep the ball from the other team, which attempts to take it, within a small region. This task is episodic – the episodes end if the taking team takes the ball or if it leaves the region. In addition to the smaller state space than the full league, it also has less sub-objectives, making it more suitable for directly comparing methods.

In order to test the scalability of algorithms that have good performance in Keepaway to more complex environments, without going directly to the full league, Kalyanakrishnan *et al.* proposed the Half Field Offense (HFO) [5]. Similarly to Keepaway, which it extends, HFO is also a subtask of the 2D Simulation League. In this task, one offense team attempts to score a goal, while a defense team attempts to prevent it. It is played within one half of the soccer field and it is an episodic task – the episodes end if the offense scores a goal or if the ball gets captured by the defense or leaves the playing area. HFO can be viewed as problem for both the offense and the defense, but the authors only focus on the former.

Hausknecht *et al.* [25] released an open source[2] implementation of the HFO. Previous work on the HFO (such as Kalyanakrishnan *et al.* [5]) did not release source code, making it inaccessible for many potential users. It is built on top of the RoboCup 2D simulation platform and it is the environment used throughout this thesis. The authors proposed a primary evaluation metric, Goal Percentage, which is defined as the percentage of trials with a goal scored. They also proposed a secondary one, Time to Goal (TTG), which is defined as the number of time steps required to score a goal, for the trials that end with one scored. This release allows to develop a partial team and already includes two teams that can be used for automated teammates: `HELIOS` (specifically the 2013 release [26]) and `HELIOS Base` [27] (also known as `Agent2D`). An instant of a match in this HFO implementation can be observed in Figure 2.3.

This HFO release provides access to both low (Figure 2.4) and high-level (Figure 2.5) state spaces. The low-level state space is an egocentric viewpoint based on `HELIOS`' world model. It contains features related to the agent such as its position and orientation, as well as other features pertaining to the ball

[2] `https://github.com/LARG/HFO`

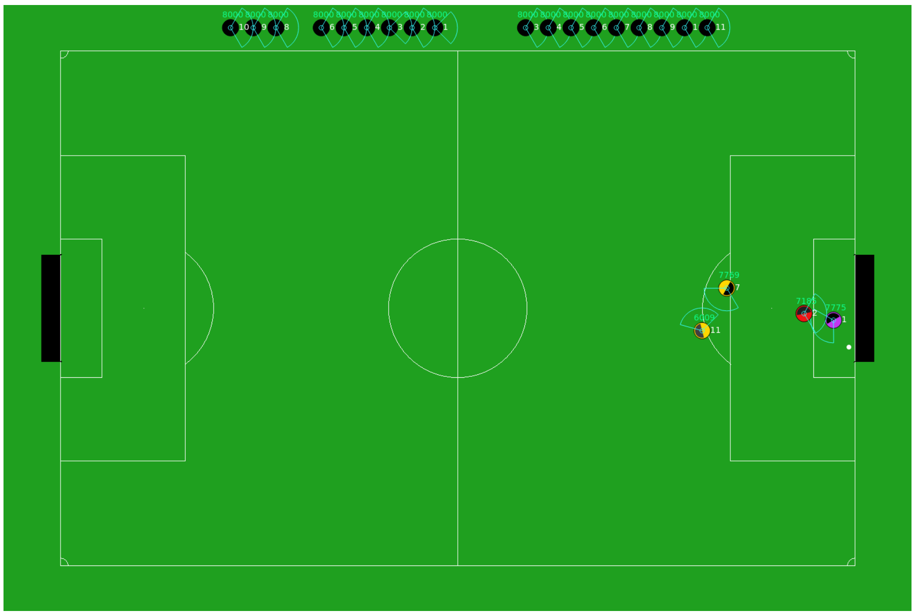
**Figure 2.3:** A match of HFO played by two offensive players (in yellow) against two defensive players (including a goalkeeper in purple).

and other agents. For a 2 VS 2 match, it has 86 features. The high-level state space is a more compact representation and may allow the agent to generalize easier. For a 2 VS 2 match, it has 24 features. When using a DQN with this environment, it should only have fully-connected layers, as the state is not based on raw data such as an image. Agents cannot directly observe each others' actions, as there are no such features in any of the state spaces.

The actions are also provided in different levels of abstraction: low, medium and high. There are four low-level actions (`dash`, `turn`, `tackle` and `kick`) and they are parameterized by power and angle. The medium-level actions (`kick to`, `move to`, `dribble to` and `intercept`) are still parameterized (by coordinates and speed). The high-level actions (`move`, `shoot`, `pass`, `dribble`, `catch`, `reduce angle to goal`, `defend goal`, `go to ball` and `mark player`) are discrete and thus suitable for an algorithm such as DQN (`pass` and `mark player` are parameterized but by a discrete argument identifying a player). They are composed by low level actions, parameterized following `Agent2D`'s strategy. According to the release's manual, the applicable high level actions for the offense team are `shoot`, `pass` and `dribble` when the agent has the ball; and `move`, `go to ball` and `reorient` when it does not.

```
0: Self Pos Valid
1: Self Vel Valid
2-3: Self Vel Ang
4: Self Vel Mag
5-6: Self Ang
7: Stamina
8: Frozen
9: Colliding with ball
10: Colliding with player
11: Colliding with post
12: Kickable
13-15: Goal Center
16-18: Goal Post Top
19-21: Goal Post Bot
22-24: Penalty Box Center
25-27: Penalty Box Top
28-30: Penalty Box Bot
31-33: Center Field
34-36: Corner Top Left
37-39: Corner Top Right
40-42: Corner Bot Right
43-45: Corner Bot Left
46: OOB Left Dist
47: OOB Right Dist
48: OOB Top Dist
49: OOB Bot Dist
50: Ball Pos Valid
51-52: Ball Angle
53: Ball Dist
54: Ball Vel Valid
55: Ball Vel Mag
56-57: Ball Vel Ang
8T: Teammate Features
8O: Opponent Features
T: Teammate Uniform Nums
O: Opponent Uniform Nums
+1: Last Action Success Possible
```

**Figure 2.4:** Low level HFO state feature set according to its manual (`T`: number of teammates; `O`: number of opponents).

```
0: X position
1: Y position
2: Orientation
3: Ball X
4: Ball Y
5: Able to Kick
6: Goal Center Proximity
7: Goal Center Angle
8: Goal Opening Angle
9: Proximity to Opponent
T: Teammate's Goal Opening Angle
T: Proximity from Teammate i to Opponent
T: Pass Opening Angle
3T: X, Y, and Uniform Number of Teammates
3O: X, Y, and Uniform Number of Opponents
+1 Last Action Success Possible
+1 Stamina
```

**Figure 2.5:** High level HFO state feature set according to its manual (`T`: number of teammates; `O`: number of opponents).

# 3 Related Work

## Contents

This chapter presents related work in the area of ad hoc teamwork. The predecessor of the ad hoc teamwork problem is presented in Subsection 3.1. Subsection 3.2 presents the work that established this problem. Subsection 3.3 describes in detail the ad hoc teamwork approach that is the base for the solution of this thesis. Subsection 3.4 presents other approaches in the literature, along with a justification of why they are not directly appropriate for the problem of this thesis. Subsection 3.5 presents work focusing on sample efficiency (that is, learning effectively with fewer samples), an important issue when trying to learn a policy as quickly as possible, such as this thesis when adapting a policy online.

## 3.1 Impromptu Teams

Bowling and McCracken [28] introduced the problem of coordination in impromptu teams (or pickup teams), where a team of agents (unknown to each other) must coordinate without any pre-established coordination, such as strategies or explicit communication protocols. These agents may be developed independently from each other, by different developers, and may have different capabilities.

In their work, the authors focused on the sub-problem of a single agent, called the pickup player, joining an unknown but already established team, called the core team. The core team is unaware that the pickup player is different, so they assign a role to it and act as if it was a regular team member. They explored this sub-problem in the context of the (simulated) RoboCup Small Size League, where each team has five players. Given a playbook (a library of team plans) for the pickup player, its task consists in selecting a play and determining its role in it (each role corresponds to a sequence of actions). The core team may have a different playbook or none at all.

To address play selection, the authors proposed two approaches: one predictive and another adaptive. The adaptive approach is based on increasing the probability of selecting plays which had good results for the pickup team. The predictive approach is based on estimating a score of how well the teammates match the roles of a play, for each play, by observing them. The pickup player then executes the play with the highest score. They also use these same approaches for role selection.

The authors also proposed three baselines. The first baseline has an unresponsive pickup player that does not move, on which the core team relies (e.g. by passing the ball). The second baseline has no pickup player, so the team is only composed by the core team (missing a player). Since there is no pickup player to falsely rely on, it is not as detrimental as the first baseline. The third baseline provides an upper-bound and corresponds to the full original team.

Both approaches had similar results, being close to the full team baseline, and generally had better results than the non-upper-bound baselines. The baseline with the immobile and unresponsive pickup player performed the worst. The baseline with the absent pickup player performed well in five player games, only performing worse with less players.

As their work relies on handcrafted playbooks, this presents an important limitation, which makes it unsuitable for more complex problems such as the setting of this thesis. However, their formulation of impromptu teams remains relevant.

Dias *et al.* [29] studied this problem in the context of human-robot pickup teams. They focused on teams formed by humans and heterogeneous robots in a treasure hunt domain. However, they relaxed one restriction of this problem: they use a pre-established protocol for communication. In their setting, there are tasks which correspond to multiple plays that must be performed. As in [28], each play has multiple roles, each composed by multiple actions. Tasks are selected via auctions in a market (each agent, including the humans, has an interface to the market), and the same happens for role assignment. The presence of a pre-established and explicit communication protocol increases the possible mechanisms for coordination. However, that is not the case in most settings where impromptu teams/ad hoc teamwork is relevant, such as the setting of this thesis. Additionally, the reliance on such communication protocols negates the versatility that ad hoc teamwork solutions attempt to offer.

## 3.2 Ad Hoc Teamwork

Stone and Kraus [30] reintroduced the fundamentally same problem as the concept of ad hoc teamwork (in the context of autonomous agents), being the designation that was adopted in later work. While this is a concept about teamwork, it is only regarding the behavior of a single agent.

In their work, the authors studied the ad hoc teamwork problem in a cooperative multi-armed bandit [4] setting, where there is a teacher agent and a learner agent. The teacher is an ad hoc agent that knows the payoff distributions of each arm, while the learner is a legacy agent that does not (additionally, it cannot pull the arm with the highest expected payoff). The dilemma of the ad hoc agent is then to decide if it should pull the arm with the highest expected payoff or to pull an arm with a smaller expected payoff, while increasing the learner's information and enabling a possibly higher total payoff. The authors present several theoretical results for this setting, as well as an algorithm based on dynamic programming. Their work is limited to the multi-armed bandit setting, so it is not directly applicable to more complex settings, such as this thesis.

In [2], Stone *et al.* expanded the concept of ad hoc teamwork in several directions. They consider an ad hoc agent to be competent if it belongs to a subset of all available agents such that subset is able to obtain or surpass a minimal performance criterion for all tasks in the domain. They also proposed an evaluation procedure to compare two ad hoc agents.

In the same work, the authors identify three challenges that are necessary to address in order to create an effective ad hoc agent: to find all the situations that it must handle; to determine appropriate behavior for each situation; and to be able to detect the current type of situation. The authors also present three dimensions that characterize ad hoc teamwork situations: teammate characteristics (such as their abilities and behavior), team characteristics (such as the composition of the team and inter-observability) and task characteristics (such as the objective and the number of time steps per episode). They state that by limiting the ad hoc problem along these dimensions, it can be made more accessible for studying.

## 3.3 PLASTIC

Barrett *et al.* [3], based on Barrett's PhD thesis [31], presents an approach to ad hoc teamwork where the ad hoc agent reuses experience with previous teammates when encountering new ones. For this purpose, they proposed a general-purpose algorithm called Planning and Learning to Adapt Swiftly to Teammates to Improve Cooperation (PLASTIC). This algorithm assumes that there are similarities between past teammates and new ones (while this may not always be true, the authors consider that often there are some). Given enough time, PLASTIC enables the ad hoc agent (but not necessarily the team) to act optimally if its past experience correctly predicts the new teammates and their behavior is fixed.

The authors focus on a limited version of the ad hoc teamwork problem where they make the following assumptions:

- the ad hoc agent has previous experience with past teammates;
- all agents in the team share a common objective;
- there are no explicit communication protocols;
- the teammates do not learn and their behavior does not change.

PLASTIC is fundamentally based on:

- acquiring knowledge of past teammates either by learning about them or receiving hand-coded knowledge;
- forming beliefs about the current teammates (about which acquired knowledge best fits them);
- using those beliefs and the knowledge of past teammates for planning;
- updating the beliefs by observing the current teammates' behavior.

This general-purpose algorithm has two realizations, PLASTIC-Model and PLASTIC-Policy. As suggested by their names, PLASTIC-Model is a model-based approach that learns models of past teammates (offline, via supervised learning [7] methods), while PLASTIC-Policy is a model-free approach that learns policies to cooperate with past teams. PLASTIC-Model then selects the models that best predict the current teammates for planning, while PLASTIC-Policy similarly selects the policy that best reflects the current team for acting.

The authors argue that PLASTIC-Policy is more adequate for complex problems with large state spaces, as it can use function approximation techniques - they state that it is more effective in HFO as planning in PLASTIC-Model needs many samples and has problems with inaccuracies in the environment modeling. As such, the solution of this thesis is based on PLASTIC-Policy – it extends it to also consider the possibility of an unknown team and to use a special policy improved online for that case. However, they state that PLASTIC-Model, by being able to use a model for each teammate (instead of each team), would be more appropriate for heterogeneous teams (as PLASTIC-Policy selects a policy that best reflects the entire team, not considering individual teammates). Additionally, they also suggest it would be better suited for teams that change behavior (not explored in their work).

### 3.3.1 PLASTIC-Policy

PLASTIC-Policy (Algorithm 3.1) directly collects samples from the environment and uses them to learn a policy for a team, using existing reinforcement learning algorithms. The authors suggest using Fitted Q Iteration (FQI) [32] for learning the policies. This thesis uses DQN to learn the policies as it is more efficient than FQI.

PLASTIC-Policy uses the Polynomial Weights Algorithm (PWA) [33] (3.1) to update the belief of each known team given the current team's behavior in the observed transition. The loss for PWA is given by 1 minus the probability of the observed behavior, for each team. However, it is not possible to directly obtain this probability, as it is not given by the policies. To address this, the authors use instead a model to estimate it, in particular via a nearest neighbor based approach. For each state $s$ and its next state $s'$ that the agent observes, it finds the past sample $\langle \hat{s}, \hat{a}, \hat{r}, \hat{s}' \rangle$ whose state $\hat{s}$ is most similar to $s$ and then uses the difference between $s'$ and $\hat{s}'$ to estimate a probability. In contrast, this thesis uses models based on neural networks to estimate this probability without searching all of the stored previous samples at each inference. This possibly also allows for better models.

$$belief \leftarrow \frac{belief \cdot (1 - \eta \cdot loss)}{\sum belief \cdot (1 - \eta \cdot loss)}, \eta \leq 0.5 \tag{3.1}$$

### 3.3.2 Application of PLASTIC-Policy to HFO

The authors used 7 different teams in their HFO experiments. In order to help the agent generalize and learn faster, they used high level states and actions. They focused on the offense of HFO, using a reward of 1000 for when the offense scored and -1000 otherwise, with an additional discount of -1 for each time step (equation 3.2). For each type of teammate, the ad hoc agent trained for 100,000 episodes.

$$r(s,a) = \begin{cases} 1000 & \text{if the offense team scores a goal} \\ -1000 & \text{if the defense team scores a goal} \\ -1 & \text{otherwise} \end{cases} \tag{3.2}$$

**Algorithm 3.1:** The PLASTIC-Policy algorithm as presented in Barrett *et al.* [3].

```
1:  function LearnAboutPriorTeammate:
        inputs:
            $t$                                   ▷ the prior teammate
        outputs:
            $\pi$                                 ▷ policy for cooperating with teammate $t$
            $m$                                   ▷ nearest neighbors model of the teammate's behavior
        params:
            Q-learning parameters: $\alpha$, $\lambda$, and the function approximation
2:      Data = $\emptyset$
3:      repeat
4:          Collect $s, a, r, s'$ for $t$
5:          Data = Data $\cup \{(s, a, r, s')\}$
6:      Learn a policy $\pi$ for Data using Q-Learning
7:      Learn a nearest neighbors model $m$ of $t$ using Data
8:      return $(\pi, m)$

9:  function UpdateBeliefs:
        inputs:
            BehaviorDistr                         ▷ probability distr. over possible teammate behaviors
            $s$                                   ▷ the previous environment state
            $a$                                   ▷ previously chosen action
        outputs:
            BehaviorDistr                         ▷ updated probability distr.
        params:
            $\eta$                                ▷ bounds the maximum allowed loss
10:     for $(\pi, m) \in$ BehaviorDistr do
11:         loss = $1 - P(a|m, s)$
12:         BehaviorDistr$(m) *= (1 - \eta$loss$)$
13:     Normalize BehaviorDistr
14:     return BehaviorDistr

15: function SelectAction:
        inputs:
            BehaviorDistr                         ▷ probability distr. over possible teammate behaviors
            $s$                                   ▷ the current environment state
        outputs:
            $a$                                   ▷ the best action for the agent to take
16:     $(\pi, m) = \text{argmax}$ BehaviorDistr  ▷ select most likely policy
17:     $a = \pi(s)$
18:     return $a$
```

The authors tested PLASTIC-Policy in two variations of the HFO task: a limited version with 2 attackers versus 2 defenders, including a goalkeeper; and a full version with 4 attackers versus 5 defenders. This thesis only focuses on the limited task. They averaged their results over 1,000 trials, with each trial having multiple episodes (matches). They measured their performance according to the fraction of trials where the offense team scored a goal in the same episode (analogous to the Goal Percentage metric in Hausknecht *et al.* [25]). They used three baselines: as an upper-bound, the agent knowing the correct policy for the team; as a lower-bound, randomly selecting a policy; and in order to compare to a single-agent approach, a combined policy based on all past experience. Besides testing PLASTIC-Policy as described previously, the authors also tested a variation that uses a multi-armed bandit approach for policy selection, which improves its estimates over each episode (instead of each time step).

The results were similar for both the limited and full variations of the task. Knowing the correct policy yielded the best results and selecting randomly the worst. More importantly, knowing the correct policy had a significant increase in the fraction of goals scored when compared to the combined policy, showing that an ad hoc teamwork approach performs better than a single-agent one. The bandit approach converged slowly to the correct policy, needing about 10,000 episodes to do so for the limited variation. It only reached a similar performance to the combined policy baseline at about 1,750 episodes. In contrast, PLASTIC-Policy achieved results near the correct policy almost instantaneously. This is due to the fact that PLASTIC-Policy can better estimate the policies' probabilities and can update them more frequently. Even if the correct policy does not have a very high probability of matching the team, it quickly can become the policy with the highest probability.

## 3.4 Other Ad Hoc Teamwork Approaches

Stone *et al.* [34] studied the problem of an ad hoc agent leading a best-response agent, which adapts to the ad hoc agent's behavior and has no means of communication, to act in such a way that maximizes the joint utility of both agents in a game theoretic setting. The best-response agent selects its best action depending on which action it believes the ad hoc agent will choose. Its belief is based on some amount of the last past actions of the ad hoc agent (hence, the best-response agent is a bounded-memory agent). The ad hoc agent thus has to lead the best-response agent to the best joint action, incurring short term losses. The authors propose an algorithm that uses a dynamic programming approach for finding sequences of actions that minimize these losses. This algorithm has polynomial complexity in time, and it is also polynomial in memory if the best-response agent only looks at the last action of the ad hoc agent, but exponential if it looks at more than one action. To mitigate this issue, the authors proposed some alterations to their algorithm, but it remained a significant limitation. They also addressed the situation where the teammate may not be deterministic, proposing further alterations. The setting of their work (leading a best-response agent) does not fit this thesis.

Agmon and Stone [35] further extended the work by Stone *et al.* [34] for multiple teammates. The authors state that in settings with multiple teammates it may not be possible to lead them to the optimal joint action, only to a reachable joint action or cycle that provides the maximum payoff between those that are reachable. They use a graph-based approach where they find the best reachable cycle of joint actions and then the shortest path to reach it. Their algorithm also has polynomial complexity if the teammates only look at the last past action of the ad hoc agent. Additionally, they prove that this problem is NP-hard if the teammates look at more than one past action, even if it is just one teammate. They also extend their approach to address leading a team of multiple teammates by a team of multiple ad hoc agents and leading when the ad hoc agent is uncertain about the behaviors of the teammates. Similarly to [34], the setting of their work does not fit this thesis.

Chakraborty and Stone [36] studied the problem of cooperating with a single Markovian ad hoc teammate in a game theoretic setting. A Markovian agent's policy verifies the Markov property: it is based on features such that their values in a certain time step only depend on their values and the joint action in the previous time step. These features are discrete and computed based on the joint play history. A memory-bounded teammate, such as in [35], is a specific case of this teammate. To this end, the authors proposed an algorithm called Learning to Cooperate with a Markovian teammate. The authors assume that the Markovian teammate's policy's features belong to a set of possible features known to the ad hoc agent. They also assume that the set of possible features is ordered and that the teammate only uses the first $K$ features, with the ad hoc agent not knowing $K$ but knowing an upper-bound. As the ad hoc agent does not know $K$, it keeps a model of its teammate's policy for every possible $K$ value, updating them and selecting the one which better predicts the teammate every $T$ time steps (any model which uses more features than $K$ should be able to predict the teammate's behavior, so they focus on the one that can do so with the smallest $K$). This approach is not appropriate for this thesis as these assumptions do not hold.

Albrecht and Ramamoorthy [37] worked on an approach where they model the ad hoc teamwork problem as a stochastic Bayesian game, allowing to tackle this problem formally. They assume that the ad hoc agent knows the possible types its teammates may be (which are pre-defined by the user). To calculate the probability of a teammate belonging to a type, they proposed an algorithm called Harsanyi-Bellman Ad Hoc Coordination. The probability distributions for each agent are then used for planning. They also extended this approach to address the case where there are types with changing behavior and to learn completely new types. Their work introduces the *level-based foraging* domain, used for evaluating ad hoc teamwork approaches, which is a grid-based world where players need to collaborate in order to eat food. The authors later [38] performed more theoretical analysis, namely analyzing the convergence properties. This approach relies on expert-provided types, which is not feasible in this thesis. Learned or generated types were not explored and as such may be unsuitable.

Albrecht and Stone [39] studied a version of the ad hoc teamwork problem where the ad hoc agent models each teammate individually, by matching it to a known type of agent. Their work is novel in the sense that it explores the possibility of parameterized types (with bounded continuous parameters). They proposed a method which estimates the relative probabilities of each type and then estimates their parameters as the agent acquires more observations of the teammates. They successfully reduced the computational costs by managing to only update the parameter estimates of a single type per time step, which had similar results to updating all of them. This approach is not appropriate for this thesis as modeling each teammate individually is hard to scale to more complex tasks such as HFO (they tested their approach in the foraging domain [37]). The same authors later performed an extensive survey [40] on existing ad hoc teamwork methods. In particular, they categorized methods in several categories, such as *type-based reasoning* (which includes [39] and PLASTIC-Model) and *group modelling* (which includes PLASTIC-Policy).

Ravula *et al.* [41] explored the situation where the behavior of teammates may change, which is assumed to not happen in most of the previous work. They use a Change Point Detection (CPD) approach to detect sudden changes in the behavior (each teammate is modeled individually). They expanded the method by Albrecht and Stone [39], by adding a CPD step where the agent stops considering previous observations if a change is detected. In order to perform the CPD, they proposed an algorithm based on a Convolutional Neural Network (CNN) [7], as they found existing CPD algorithms to be unsuitable for this task. Being based on [39], this approach is also unsuitable for this thesis.

Wu *et al.* [42] proposed an online learning algorithm called Online Planning for Ad Hoc Agent Teams (OPAT). OPAT is based on estimating the utility of each action via Monte Carlo Tree Search (MCTS) [43], where smaller virtual games are solved using teammate actions estimated from the past interaction history (if there is no history, it simulates that the other agents act rationally). In their work, the authors simplify the problem of ad hoc teamwork with some assumptions, namely that the state is fully observable and that the ad hoc agent can observe the joint actions. Furthermore, the ad hoc agent knows the possible actions the teammates may take. They successfully experimented with their algorithm with two types of agents, one that follows a pre-defined sequence of actions and another that plays optimally according to its beliefs. OPAT is unsuitable for this thesis due to these observability assumptions.

Melo and Sardinha [44] focused on the sub-problem of task identification, where the ad hoc agent is not aware of the objective nor the rewards. In their work, the authors propose that an ad hoc teamwork problem is subdivided in three steps: task identification, teammate identification and planning. They argue that most existing approaches ignore task identification, assuming it is given, so their contributions focus on it. In their setting, the ad hoc agent attempts to identify the task (from a set of possible ones) by observing its teammates' behavior. As such, the rewards are modeled by the agent and not observed directly in the environment. The authors propose two approaches: one based on online learning and another based on decision theory. Their work is not suitable for this thesis as the rewards are known.

## 3.5 Sample efficiency

This section presents some approaches addressing sample efficiency (learning effectively with fewer samples), which is relevant in the context of this thesis for improving a policy online, especially regarding the performance of the policy during the first time steps.

Hester and Stone [45] introduced TEXPLORE, a model-based method for real time reinforcement learning. It is able to learn with few samples, in continuous state spaces and is computationally efficient. In addition, it can handle unknown sensor and actuator delays, which is especially relevant for robotics. It uses decision trees in the form of random forests to model the MDP using the collected samples so far. Considering a state $s$, its next state $s'$, an action $a$ and a reward $r$, this is a supervised learning problem with $\langle s, a \rangle$ as the input and $s' - s$ and $r$ as the outputs. It uses those random forests to attempt to explore the state-action pairs that seem to be more likely to receive a reward. Given the model of the MDP, it uses Upper Confidence Bounds applied to Trees (UCT) [46] (an MCTS method) with some modifications to plan, which can return its results at any time and as such is suitable for real time control. One of the empirical tests they performed was a task where an autonomous vehicle had to adjust its velocity, whose reward was based on the difference between the current and the desired velocities. As shown in Barrett *et al.* [3], learning a policy instead of modeling the environment seems to be a better approach for HFO.

Nagabandi *et al.* [47] proposed a hybrid approach, where they initially use a model-based method which is then used to initialize a model-free method. Their approach is based on the reasoning that a model-based approach is more sample efficient and thus is more suitable when there are few samples, but a model-free approach is able to obtain better policies and thus higher rewards at the expense of more samples. In the model-based method, they learn a model for the difference between future states and their previous states, which is then used together with a reward function for evaluating the possible state-action pairs the agent may choose. In the model-free method, they start by learning a policy via a neural network, using collected trajectories following the model-based method. Afterwards, they optimize this initial policy via Trust Region Policy Optimization [48] as more samples are observed. They tested their approach in a physics simulator, where agents had to learn how to walk. Similarly to Hester and Stone [45], learning a policy directly seems to be a better approach for HFO. However, this approach could be studied further for future work.

Yang Yu [49] presented several aspects of sample efficiency in reinforcement learning that are challenging. The first aspect is limitations in the exploration strategies used by agents, which often tend to rely on finding new trajectories by introducing noise in the actions (action space noise). The author presented better alternatives such as parameter space noise, where noise is introduced in the parameters of the policy, and curiosity-driven exploration, where the agent is encouraged to visit unknown states. Another aspect is environment modeling, where model-based methods were presented as being more sample efficient than model-free, but harder to use in more complex environments. As such, approaches that combine model-based methods with model-free were suggested, where an inaccurate model of the environment is used to generate augmented features for the model-free method. The author also studied different transfer reinforcement learning approaches, each adequate for a subset of reinforcement learning problems. It was noted that there is no general strategy for transfer reinforcement learning. Finn *et al.* [50], one of these approaches, presented a meta-learning approach to create a model that can adapt to new learning tasks with few samples, which could be used for future work. This thesis, however, uses parameter sharing transfer learning [12] (in terms of supervised learning), in order to reuse previous knowledge in existing policies. Finally, the author also mentions several optimization and abstraction techniques. Dynamic environments and large action spaces are also identified as challenges for reinforcement learning.

Barrett *et al.* presented *TwoStageTransfer* [51], a transfer learning algorithm that weights multiple data sources via cross-validation (e.g. one for each team in the library of known teams in the setting of this thesis). This thesis focuses on parameter sharing transfer learning instead of experience transfer, so this approach was not explored. Lazaric *et al.* [52] also presented a method for experience transfer, first based on the similarity between source tasks and the target task, and then on the similarity between source instances (samples) and target instances. It was also not explored in this thesis for the same reason.

# 4

# Learning, identifying and adapting to teammates

**Contents**

The problem (Section 1.2) is addressed with an architecture (Figure 4.1) based on Barrett's PLASTIC-Policy [3], comprised of a training and an execution phase. In the training phase, the agent builds a library of policies, with one policy for each of the known teams. The agent can collect and train on as much experience as needed. In the execution phase, the agent needs to identify the current team either as one of the known teams or as an unknown team. If the team is known, the agent will ideally use the policy previously trained with it, else it will use a policy that is improved online. A team is considered to be unknown if the agent has determined it is not one of the teams it knows. It is considered unidentified if the agent has not yet determined if it is one of the known teams or unknown.

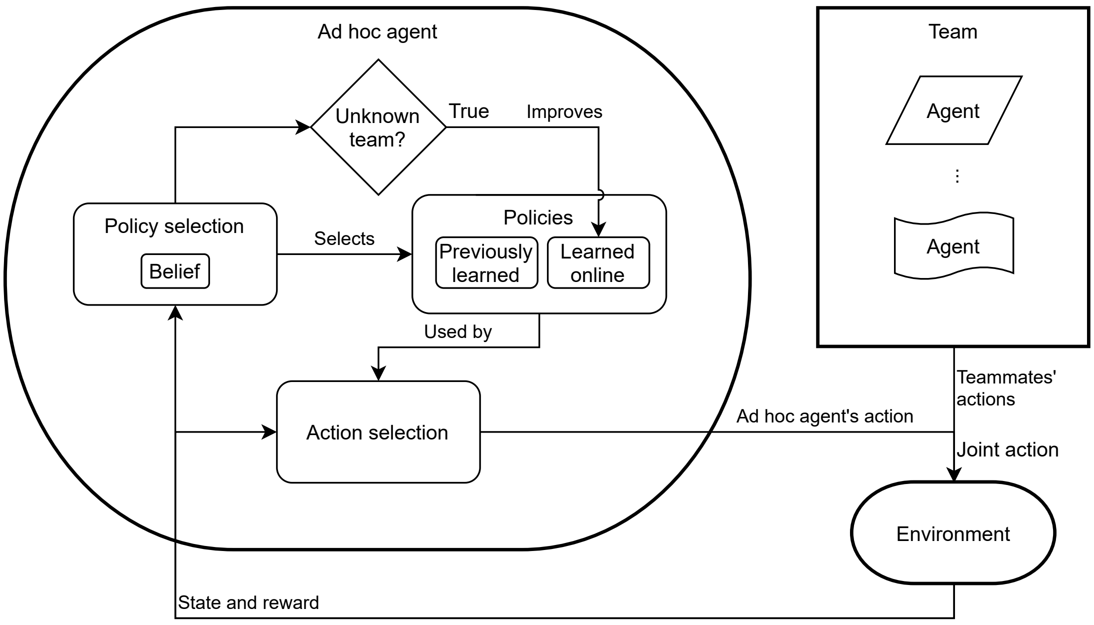

**Figure 4.1:** Overview of the architecture (execution phase), inspired by PLASTIC-Policy [3].

## 4.1 Policies for known teams

Policies for known teams are learned in the training phase using DQNs. It is appropriate for this problem as it approximates Q-values for discrete actions. This section describes how the policies are modeled and how the hyper-parameters are adjusted through a non-extensive search.

This adjustment is performed using an `HELIOS` teammate as a representative. This teammate was selected as it is one of those with the fastest execution times. While potentially biased, this simplification is performed in order to reduce the number of policies needed to be learned for this purpose. In the case of subjectively similar results, the value closest to the initial hyper-parameters (according to literature) is chosen.

### 4.1.1 State and action spaces

The HFO environment provides both low level and high level state feature sets. Both feature sets are experimented. As usual when employing DQN, the most current states are concatenated for its input. The 4 most recent states will be used, as often used in the literature (e.g. Mnih *et al.* [13]). This concatenation may allow the agent to extract more complex features, such as more insights in the behavior of other agents.

The high level action set provided by the HFO environment is used, as the lower level action sets are parameterized by continuous values and thus unsuitable for DQN. When it does not have the ball, the agent can move (according to team `Base`'s [27] strategy), go to the ball and reorient (the agent stops and recovers stamina). When it has the ball, it can dribble, shoot and pass it. To model these conditions for the actions, the DQN algorithm is modified according to Lanctot *et al.* [22] such that the DQN does not consider illegal actions, by forcing the probability of illegal actions to be zero when acting and ignoring them when determining $\max_{a' \in \mathcal{A}(s_{t+1})}$ during the update.

### 4.1.2 Reward function

The reward function (4.1) is used. It positively rewards the agent when the offense team scores a goal and is neutral otherwise. There is no reward shaping, so the agent will not learn behavior that exploits the reward function in unintended ways. Barrett's [3] reward function (3.2) presented some issues in terms of the agent attempting to play too defensively until the maximum time step limit, even if scaled to $[-1, 1]$.

$$r(s,a) = \begin{cases} 1 & \text{if the offense team scores a goal} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

### 4.1.3 Initial hyper-parameters

The hyper-parameters/modeling from Table 4.1 are used as a starting point. They are based on the work by Mnih *et al.* [13], with some adjustments to the target network update frequency and final exploration rate by Hasselt *et al.* [15] which are also used in later extensions of DQN. The optimizer is also changed to Adam, as used in Hessel *et al.* [19].

### 4.1.4 Hyper-parameter and modeling adjustment

Some of the hyper-parameters and modeling approaches from Table 4.1 are adjusted through a non extensive search. The results are presented in Section 5.2.

1. Using the low level feature set, the first hyper-parameter to be adjusted is the train frequency. A higher train frequency (that is, lower number of time steps between each training) may allow to train

| Hyper-parameter/modeling | Value | Reference/explanation |
|---|---|---|
| DQN extensions | None | Mnih *et al.* [13] |
| Hidden layers (fully connected) | One layer with 512 hidden units with ReLU activation | Mnih *et al.* |
| Minibatch size | 32 samples | Mnih *et al.* |
| Replay memory size | 1 million samples | Mnih *et al.* |
| Agent history length | 4 states | Mnih *et al.* |
| Target network update frequency | 7500 updates | Hasselt *et al.* [15] (30000 frames) |
| Discount factor | 0.99 | Mnih *et al.* |
| Action repeat | None | HFO is sensible to each individual action. |
| Update frequency | 4 time steps | Mnih *et al.* |
| Optimizer | Adam | Hessel *et al.* [19] |
| Learning rate | 0.0000625 | Hessel *et al.* |
| Initial exploration ($\epsilon$-greedy) | 1 | Mnih *et al.* |
| Final exploration | 0.01 | Hasselt *et al.*, Hessel *et al.* |
| Final exploration time step | 1 million time steps | Mnih *et al.* |
| Replay start size | 50,000 samples | Mnih *et al.* |

**Table 4.1:** Initial hyper-parameters based on literature. Refer to Mnih *et al.* [13] for a detailed description.

the policies in less wall clock time, which would be saved during the other hyper-parameter/modeling searches. A train frequency of 1 is compared to the train frequency of 4 from Table 4.1.

2. The low level feature set is compared to the high level feature set, to determine which modeling approach provides better performance.

3. A deeper topology of the Q-network is experimented, consisting of two hidden layers of 256 and 64 ReLU respectively. This topology is not extensively fine-tuned. It may ease transfer learning in Section 4.3 by not updating the weights of the first or even both layers.

4. Some extensions to the DQN algorithm are evaluated, in terms of modeling: Double DQN [15] and Dueling DQN [18]. All fully connected layers are doubled for the Dueling DQN. However, neither of these extensions showed to be beneficial in this problem in Section 5.2.

### 4.1.5 Policy training and effectiveness

Policies are trained for every known team and each trained policy is then evaluated with all known teams (Section 1.2). Ideally, for each agent the best policy should be the one that was trained with it. With such result, it will be beneficial to identify the team the agent is playing with, in order to select the most optimal policy.

## 4.2 Policy selection

PLASTIC-Policy needs to identify the team in order to select the policy to be used during the execution phase. This is performed using belief losses that are calculated for each observed transition, which are then used to update the belief. The following subsections present how these beliefs are calculated.

### 4.2.1 Identifying known teammates

PLASTIC-Policy's approach involves scanning known transitions for each team in order to find the past transition whose initial state is most similar to the initial state of the observed transition. Then it uses the difference between the next state of the known transition and the observed new state as a loss. This scanning procedure may have a significant impact in execution performance, so this thesis focuses on creating models for the belief loss during the training phase, which are then used in the execution phase. The samples contained in the DQN replay buffers are used to create these models.

#### 4.2.1.A Predicting the next state

One alternative is to predict the next state, which includes changes dependent on the team. For each known team, an MLP is trained (hyper-parameters in Table 4.2): given an observation consisting of 4 concatenated states and an action (one-hot encoded), the output is the difference between the new state $s'$ and the previous state $s$ ($s' - s$ is predicted instead of $s'$ since, as pointed out in [45], the relative transitions may be similar for many state-action pairs). This is an adaptation of the method employed in PLASTIC-Policy, where an MLP is used to obtain $s'$ instead of scanning past transitions directly.

During execution, for each known team:

1. The difference is predicted and added to the previous state;
2. The resulting predicted new state is clipped to the maximum and minimum values of the features;
3. A loss is measured between the real new state and the predicted one using the Huber loss.

This loss vector is then normalized.

#### 4.2.1.B Discriminating teams

Another alternative is to discriminate between the known teams. This is modeled as a multi-class classification problem. An MLP is trained (hyper-parameters in Table 4.3), where the inputs correspond to the previous 4 concatenated states, an action (one-hot encoded) and the new state. The outputs are one-hot encoded vector representations of the known teams, where the first position denotes the transition was drawn from the first team's replay buffer and so on. The training data is naturally balanced,

| Hyper-parameter | Value |
|---|---|
| Hidden layers (fully connected) | Four layers with 256, 192, 128 and 96 units respectively with ReLU activation and He initialization |
| Output layer | Linear (no activation) with the size of the used state feature set, using uniform He initialization |
| Loss | Huber |
| Optimizer | Adam |
| Learning rate | $10^{-2}$ for 750 iterations, $10^{-3}$ for 150 iterations, $10^{-4}$ for 75 iterations and $10^{-5}$ for 25 iterations, for a total of 1000 training iterations |
| Batch size | $2^{14}$ (adjusted for training speed) |
| Regularization | L1 and L2 of $10^{-6}$ |

**Table 4.2:** Hyper-parameters for the predictive approach (not extensively adjusted).

| Hyper-parameter | Value |
|---|---|
| Hidden layers (fully connected) | Four layers with 256, 128, 64, 32, 16 and 8 units respectively with ReLU activation and He initialization |
| Output layer | Softmax activation with as many outputs as there are known teams, using uniform Glorot initialization |
| Loss | Categorical cross-entropy |
| Optimizer | Adam |
| Learning rate | $10^{-2}$ for 350 iterations, $10^{-3}$ for 75 iterations, $10^{-4}$ for 50 iterations and $10^{-5}$ for 25 iterations, for a total of 500 training iterations |
| Batch size | $2^{14}$ (adjusted for training speed) |
| Regularization | L1 and L2 of $10^{-6}$ |

**Table 4.3:** Hyper-parameters for the discriminative approach (not extensively adjusted).

as each team's policy has an associated replay buffer of equal size. The output of this MLP can then be used to obtain a loss vector as follows: $loss\ vector = 1 - output\ vector$.

#### 4.2.1.C Advantages and disadvantages of each alternative

The alternative based on discriminating teammate behavior is an easier problem and as such should result in a better loss. However, every time a new team is added to the library, it needs to be retrained. For development and testing purposes of this thesis, it means one MLP needs to be trained for each set of possible known teams that are explored. The alternative based on predicting teammate behavior is more versatile, being able to just add a new MLP for a new team. However, it is slower, as multiple MLPs need to be inferred during execution. The discriminative approach is shown as having better results in Section 5.4.

### 4.2.2 Belief updating

To update the agent's belief during execution, algorithms such as the Polynomial Weights Algorithm (PWA) [33] (4.2) (used in Barrett's work) or the update rule used in Exponentially Weighted Forecasters

(EWFs) [53] (4.3) can be used. Higher values of $\eta$ make the belief converge more quickly. EWF seemed to initially have better results than PWA in Section 5.4. A Bayesian update is not used for the discriminative approach (which gives probabilities) since, as pointed out in [3], a single update may set a belief to zero.

$$belief \leftarrow \frac{belief \cdot (1 - \eta \cdot loss)}{\sum belief \cdot (1 - \eta \cdot loss)}, \eta \leq 0.5 \quad (4.2)$$

$$belief \leftarrow \frac{belief \cdot e^{-\eta \cdot loss}}{\sum belief \cdot e^{-\eta \cdot loss}}, \eta > 0 \quad (4.3)$$

### 4.2.3 Belief loss for unknown teams

It is not feasible to train the described MLP's during execution with samples acquired from the current team, as they are unlabeled data. As such, the belief loss for an unknown team must be obtained by other means.

Considering that the vector of belief losses for known teams sums to 1, if there is a team whose loss for the observed transition is smaller than 0.5 (in other words, more than half of the probability) then the agent probably is playing with that team. Otherwise, if there is no such team, then the agent probably is playing with an unknown team. As such, a simple heuristic approach is to use 0.5 as the loss for the unknown team: if there is a team where the model is confident across multiple transitions it should be identified, otherwise the agent should recognize the team as unknown. These results are smoothed across multiple transitions by the algorithms used for the belief update.

If the vector of losses of known teams summed to another value, the loss for an unknown team would thus be half of that value. This approach needs at least two known teams: with only one known team it would always have a loss of zero.

## 4.3 Policy for an unknown team

This section presents our approach to learning a policy for unknown teams via transfer learning. In face of an unknown team, the agent can start by acting with a policy learned within its library. Such policy will probably be better than acting randomly or starting to learn a new policy from scratch. One heuristic is to use the policy whose team the agent considers to be most similar to the unknown team. To do that, the agent can keep a second belief vector where the loss of an unknown team is not considered (to avoid numerical issues in the main belief, such as individual beliefs possibly being zero). It can then act with the policy whose team has the highest value in that second belief vector. This is equivalent to ignoring the possibility of an unknown team.

Assuming the agent has better performance while playing with a teammate using a policy trained for it than a policy trained for another teammate, this strategy will produce non-optimal behavior. However, training a policy from scratch is also not desirable, as it would produce worse performance for an extended period of time. Ideally the agent would leverage its past training while improving by observing the team's behavior.

### 4.3.1 Adapting a selected pre-trained policy

One approach is to select a pre-trained policy and adapt it to the unknown team, using it as a source for parameter sharing transfer learning [12]. This selection can be done by using the second belief vector and selecting the team whose belief is highest after a certain amount of time steps (e.g. 1000). This policy is then used dynamically like the remaining.

However, training this policy online may result in some instability as the outputs of the network might change abruptly with just a few updates. As such, the following possible improvements to this transfer learning approach are also tested, in order to possibly improve the stability:

1. Use larger batch sizes, which may result in more similar gradients between updates;
2. Freeze the weights of the first hidden layer of the DQN, resulting in less weights to update; this is usual practice when employing parameter sharing transfer learning [12];
3. Reduce the learning rate, resulting in smaller updates;
4. Use samples from the selected known team, possibly avoiding overfitting in the initial replay buffer.

In order to perform exploration to learn more about the unknown teammate, the agent acts with the same 0.01 $\epsilon$-greedy exploration presented in Table 4.1 when using this policy for acting, during PLASTIC-Policy's execution.

# 5 Results

## Contents

This Chapter presents results that were used to validate the solution described in Chapter 4. Section 5.1 presents how these results are evaluated. Section 5.2 presents results for the hyper-parameter and modeling adjustment described in Section 4.1. Section 5.3 presents the results of policies trained for all possible teams. Section 5.4 presents results for the policy selection mechanisms presented in Section 4.2. Section 5.5 presents results for adapting a policy to an unknown team, presented in Section 4.3.

## 5.1 Evaluation procedure

This section presents the tasks and metrics used to evaluate the solution.

### 5.1.1 Task

The main task consists in repeatedly playing HFO episodes, where an ad hoc agent plays with a teammate belonging to one of the six possible teams (Section 1.2) in the offensive against two defensive `HELIOS` players, including a goalkeeper. This task varies in the teammate's team and in the ad hoc agent's behavior:

1. For training a policy where the ad hoc agent knows the teammate, it plays for up to 400 thousand episodes and creates a policy using the observed transitions (10 runs).

2. For testing the policy selection mechanisms, scenarios involving different subsets known teammates are evaluated (these subsets are: all teams known; an example with three known teams; an example with five known teams) (100 runs).

3. For testing the policy improved online for an unknown team, the agent is forced to treat the unidentified team as unknown and to adapt from a pre-determined policy (10 runs). The resulting policy is evaluated over 50 thousand episodes.

To obtain the policies that are used in further sections after the policy hyper-parameter and modeling adjustment (Section 5.2), for each team the policy that scores the highest among the last 5 evaluations of each run is selected[1]. Each run uses a random seed.

### 5.1.2 Metrics

The fraction of episodes ending in goal is used to evaluate the performance of policies. It is equivalent to the goal percentage metric in Hausknecht *et al.* [25]. In order to evaluate our approach during the training phase, the agent is tested every 5000 episodes with the policy learned until that point (that is, with static

[1] Due to time constraints, the policies were selected only among the first 4 runs.

DQN weights and no exploration) during 500 episodes. Each policy is trained for 10 independent runs (ideally more runs would have been performed).

The fraction of trials where the correct team has maximum belief (among all teams in the belief vector) is used for evaluating the policy selection. It is evaluated over the first 1000 to 10,000 time steps after meeting an unidentified teammate. This evaluation is performed for 100 trials.

The online policy improvement also uses the fraction of episodes ending in goal as a metric. It is evaluated without exploration every 1000 episodes, for 1000 episodes with static DQN weights. Each experiment is performed for 10 independent runs.

The plots used to present the results also show 95% confidence intervals, obtained using the bootstrap method [54] with 1000 resamplings[2].

## 5.2 Policy hyper-parameter and modeling adjustment

In this section, the learning of policies for known teams is adjusted according to Section 4.1. An `HELIOS` teammate is used as a representative to adjust hyper-parameters and the modeling.

Increasing the train frequency from every 4 time steps (Figure 5.1) to every time step (Figure 5.2) (less time steps between updates results in more frequent training) has no negative impact in the performance, but allows to train a policy in a smaller amount of episodes (400 thousand for the former and 200 thousand for the later). This also may result in some wall clock time savings, depending on the number of episodes used (e.g. around 33 hours instead of 36-38 for each run of Figure 5.2 and Figure 5.1, respectively).

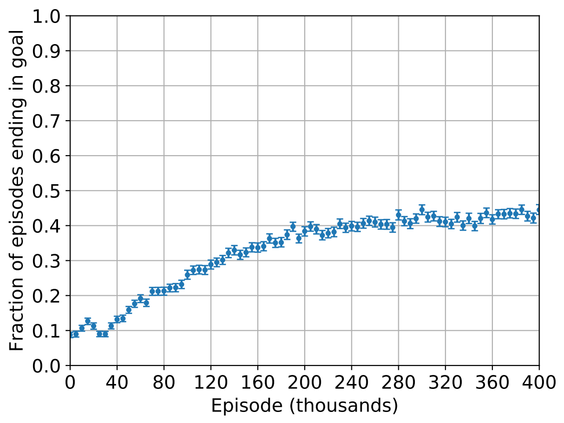

**Figure 5.1:** Train frequency of 4 time steps (low level state feature set; 512 ReLU).

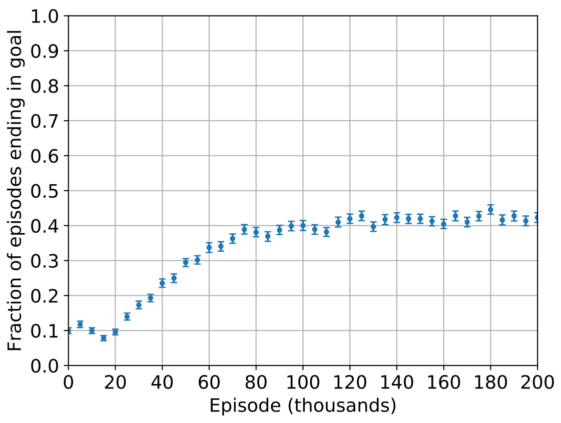

**Figure 5.2:** Train frequency of 1 time step (low level state feature set; 512 ReLU).

The high level HFO state feature set (Figure 5.3) has significantly lower performance than the low level set (Figure 5.2). This may be caused by insufficient or inadequate features for DQN. One hypothesis is

[2]As there are few runs, the results for a run are not averaged first – this results in smaller plotted errors than expected.

the angles: in the low level feature set they are encoded as a sinus-cosinus pair, avoiding discontinuities, while in the high level feature set they are a single scalar.

The deeper topology with two hidden layers of 256 and 64 ReLU respectively (Figure 5.4) had improved results over a single layer of 512 ReLU (Figure 5.2) – the latter reached a performance of 0.4 to 0.45 starting at around episode 100 thousand, while the former barely went over 0.4. In general, deeper networks allow to learn a representation that is composed by simpler representations [55], which may present an advantage.

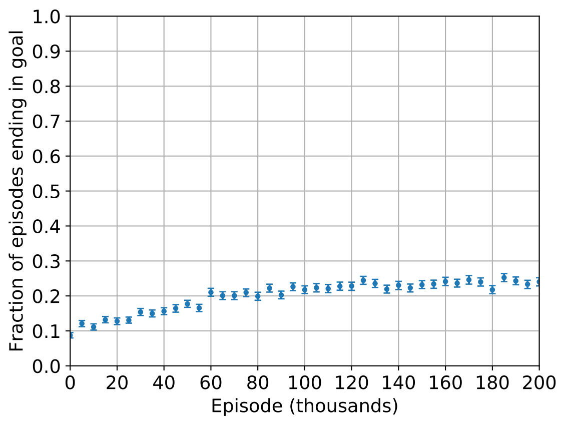

**Figure 5.3:** High level feature set (train frequency of 1 time step; 512 ReLU).

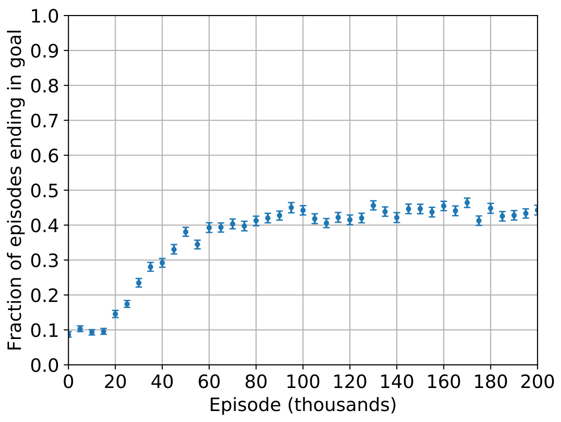

**Figure 5.4:** Hidden layers with 256 and 64 ReLU (low level state feature set; train frequency of 1 time step).

Regarding extensions to the original DQN [13], neither Double DQN [15] (Figure 5.5), Dueling DQN [18] (Figure 5.6) nor their combination (Figure 5.7) improved the performance in this environment. For instance, compare the performance at around 60 episodes (with a fraction of episodes ending in goal of around 0.3 or lower) to Figure 5.4 (around 0.4). Therefore, these extensions are not used in the following sections.

In summary, relatively to Table 4.1 are used: two fully connected hidden layers of 256 and 64 ReLU respectively; and an update frequency of 1 time step.

## 5.3 Learning and evaluating policies

Team `Base` and `Gliders` reached an average fraction of episodes ending in goal of around 0.3 to 0.4 using the adjusted hyper-parameters and modeling, `AUT MasterMinds`, `Cyrus` and `HELIOS` a fraction of around 0.4 to 0.5 and `YuShan` around 0.5. The respective plots can be observed in Appendix A.1.

The policy that is used for each team in the following sections was selected among the last 5 evaluations of each of the first 4 runs, as described in Subsection 5.1.1. Each policy was then compared

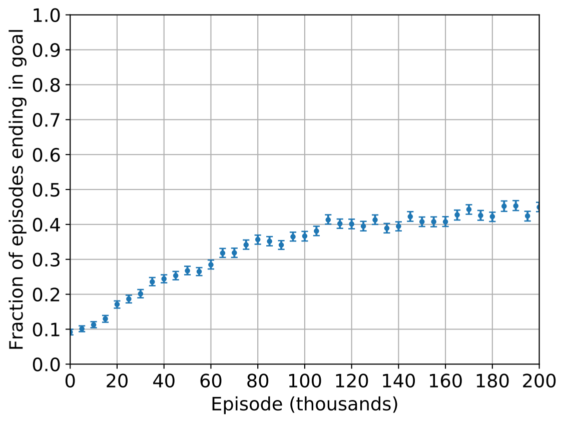

**Figure 5.5:** Double Q-learning (low level state feature set; train frequency of 1 time step; 256 and 64 ReLU).

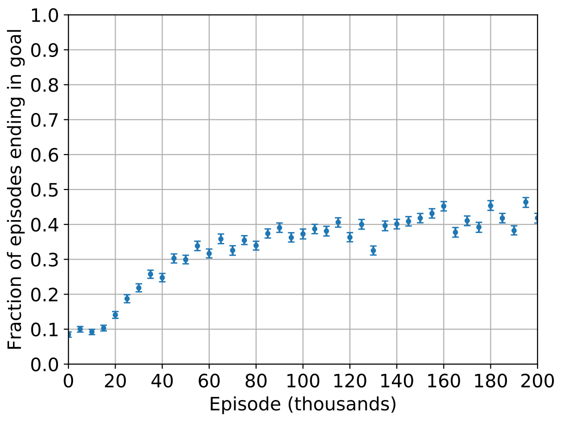

**Figure 5.6:** Dueling networks (low level state feature set; train frequency of 1 time step; 256 and 64 ReLU).

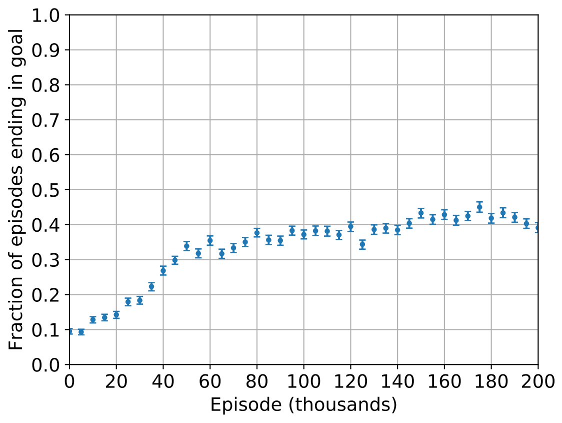

**Figure 5.7:** Double Q-learning and dueling networks (low level state feature set; train frequency of 1 time step; 256 and 64 ReLU).

against all teams for 1000 episodes for each combination. The results are present in Table 5.1. As can be observed, the most effective policy for each teammate is often the policy it was trained with, with the exception of `HELIOS`. This result may suggest that some teams do not allow to explore certain plays as effectively as others.

## 5.4 Evaluating the policy selection within the ad hoc agent

This section presents the results of the policy selection mechanisms described in Section 4.2.

| | | Policy (the team it was trained with) | | | | | |
|---|---|---|---|---|---|---|---|
| | | AUT MasterMinds | Base | Cyrus | Gliders | HELIOS | YuShan |
| Team | AUT MasterMinds | **0.481** | 0.334 | 0.359 | 0.375 | 0.408 | 0.330 |
| | Base | 0.336 | **0.392** | 0.367 | 0.363 | 0.298 | 0.389 |
| | Cyrus | 0.451 | 0.403 | **0.481** | 0.395 | 0.432 | 0.432 |
| | Gliders | 0.351 | 0.380 | 0.375 | **0.391** | 0.365 | 0.390 |
| | HELIOS | **0.467** | 0.411 | 0.375 | 0.411 | 0.456 | 0.356 |
| | YuShan | 0.343 | 0.399 | 0.384 | 0.402 | 0.361 | **0.530** |

**Table 5.1:** Policy effectiveness (given by fraction of episodes ending in goal) when playing with all possible teams. The most effective policy for a given team is highlighted in bold. Each combination played for 1000 episodes.

### 5.4.1 Only known teams

Once again, an HELIOS teammate is used as a representative to adjust the belief updating, between PWA and EWF (Subsection 4.2.2). For this purpose, the discriminative approach (Subsubsection 4.2.1.B) is used, as it is more computationally efficient. The results with PWA can be observed in Figure 5.8. The results with the update rule used in EWFs can be observed in Figure 5.9 for $\eta = 1$ and Figure 5.10 for $\eta = 10$. The update rule used in EWFs with $\eta = 1$ seemed to perform the best up to the first 300 time steps, while PWA seemed to have slightly better results up to 700 time steps. As the first time steps are of higher concern in ad hoc teamwork, the EWF with $\eta = 1$ approach was chosen.

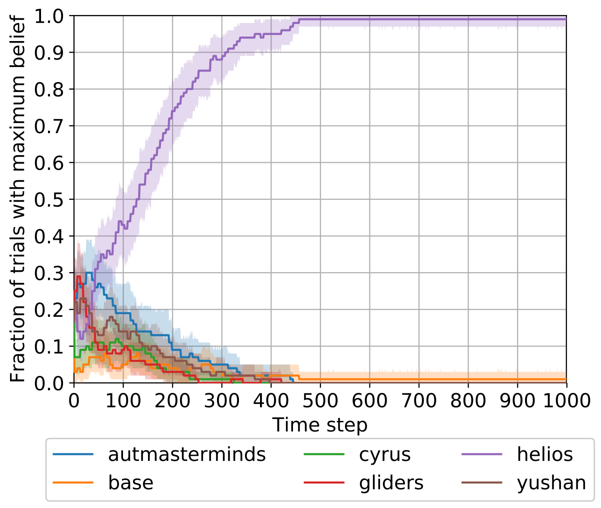

**Figure 5.8:** Discriminative approach, using PWA with $\eta = 0.5$.

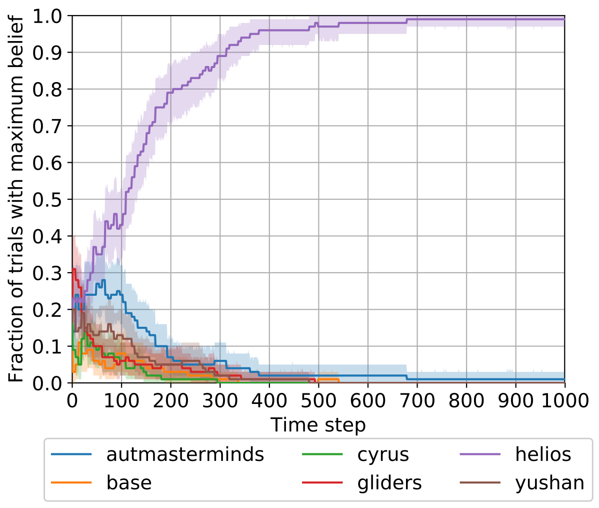

**Figure 5.9:** Discriminative approach, using EWF with $\eta = 1$.

The loss approach is then decided between the predictive approach (Subsubsection 4.2.1.A – Figure 5.11) and discriminative approach (Subsubsection 4.2.1.B – Figure 5.9), again using HELIOS as a representative. As can be observed, the discriminative approach identifies the correct team significantly faster (the predictive approach is plotted for 10 thousand time steps). This may be due to it being easier

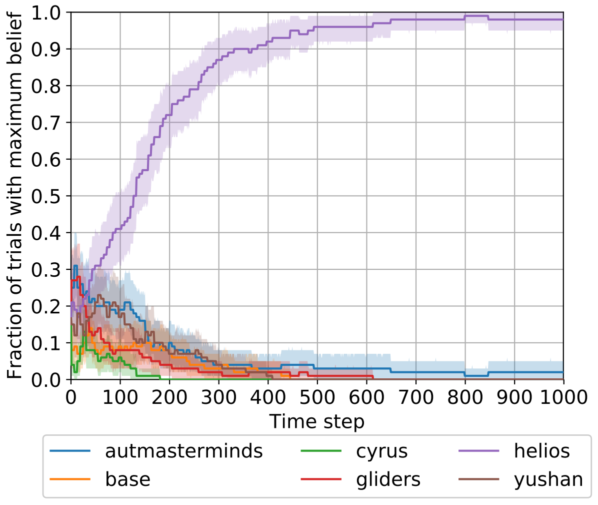

**Figure 5.10:** Discriminative approach, using EWF with $\eta = 10$.

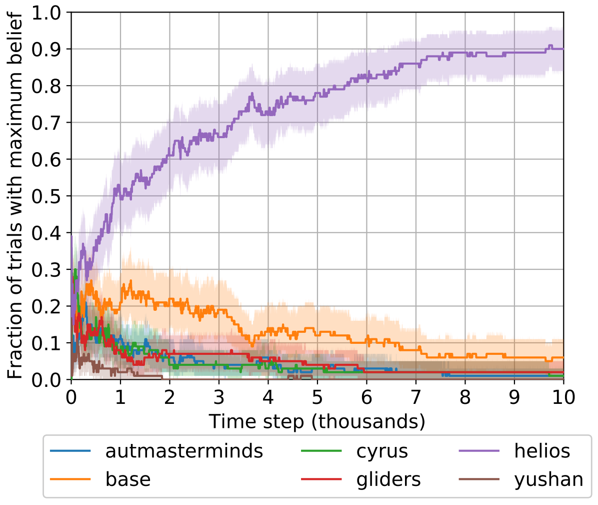

**Figure 5.11:** Predictive approach, using EWF with $\eta = 1$.

to learn compared to the predictive approach.

For the remaining teams, by around 400 time steps the agent correctly identified the team at least for 90% of the trials, using the update rule used in EWFs with $\eta = 1$ and the discriminative approach (Appendix A.2).

### 5.4.2 Considering the possibility of an unknown team

Considering an example of three known teams (the first three by alphabetical order - `AUT MasterMinds`, `Base` and `Cyrus`) and three unknown teams (`Gliders`, `HELIOS` and `YuShan` – identifiable as ”unknown”), the agent can correctly identify all teams (tested for all six possible teams individually) for the majority of trials after 1000 time steps, except for `Gliders`, which it misidentifies as `Base` (Figure 5.12). One possibility is that the discriminative model used for the belief loss failed to create strict enough boundaries, such that most `Gliders` transitions fall within the boundaries of `Base` in the model and are classified as such. The agent correctly identified every known team at least for 90% of the trials by around 200 time steps, but the unknown teams it correctly identified were for only around 50% to 65% of the trials by around 1000 time steps (remaining plots in Appendix A.3).

Considering another example where five teams are known (all except for `Gliders`), there is an increase in identification performance for unknown teams (Figure 5.13). The agent correctly identified every team for 80% to 90% of the trials by around 500 time steps (remaining plots in Appendix A.4). Compared to three known teams, the agent believes a known team is unknown for more steps in the beginning (Figure 5.14 for three known teams and Figure 5.15 for five known teams). This may be due to having more possible teams, so the model gives more varied predictions, causing it to take longer to converge to a known team and having a greater bias for the unknown possibility. The results for the remaining known teams are similar to Figure 5.15.

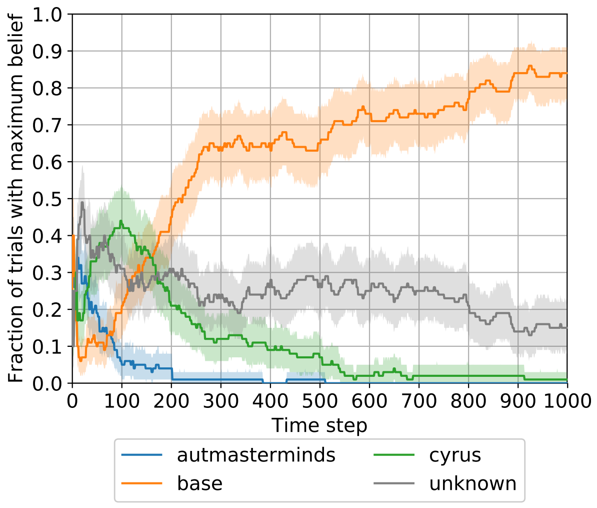

**Figure 5.12:** Three known teams (team `Gliders`).

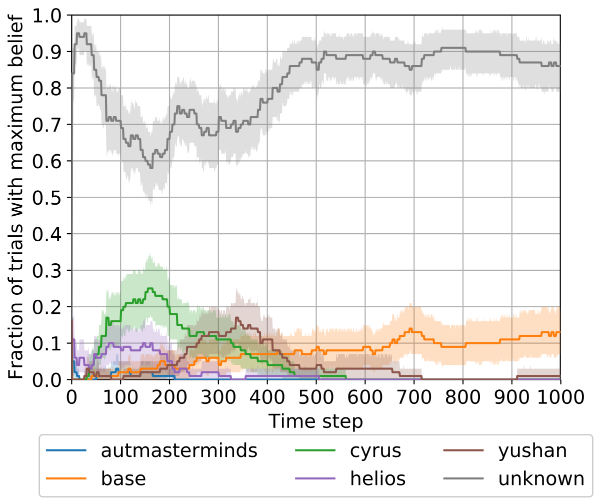

**Figure 5.13:** Five known teams (team `Gliders`).

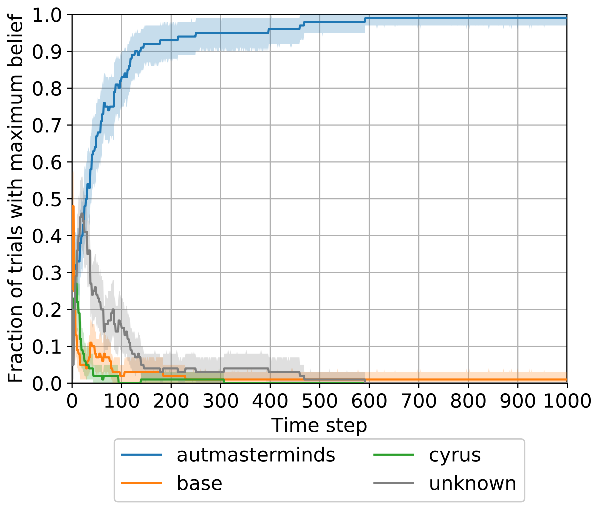

**Figure 5.14:** Three known teams (team `AUT MasterMinds`).

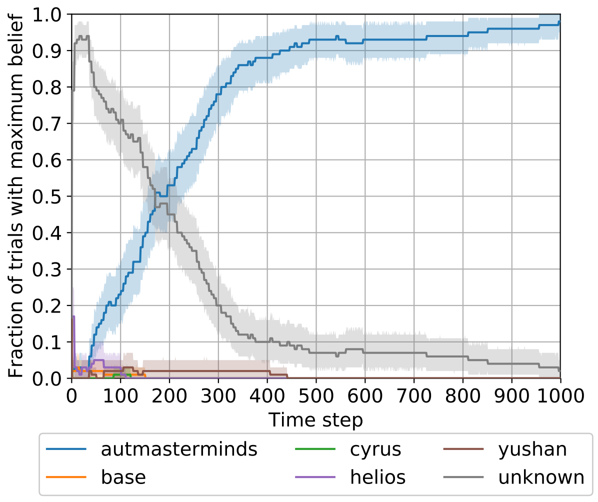

**Figure 5.15:** Five known teams (team `AUT MasterMinds`).

## 5.5 Policy for an unknown team

In this section are presented the results of the approach for adapting a policy to an unknown team, based on parameter sharing transfer learning [12]. A case where the agent is playing with a `YuShan` teammate and only knows `AUT MasterMinds` is used for illustrative purposes (this is the case with the biggest possible growth according to Table 5.1). Adapting a pre-trained policy often results in achieving better performance faster (Figure 5.16) when compared to training a new policy from scratch (Figure 5.17), particularly in the initial episodes.

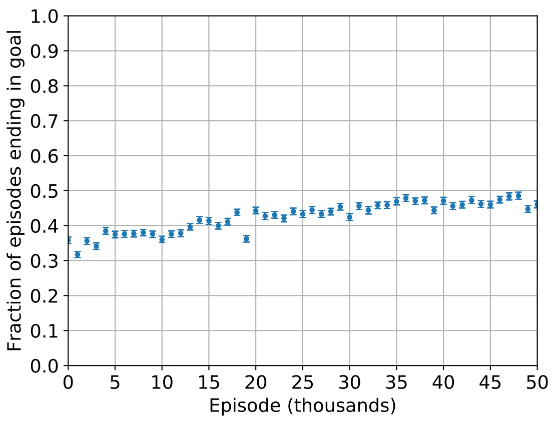

**Figure 5.16:** Adapting a policy from `AUT MasterMinds` to `YuShan`.

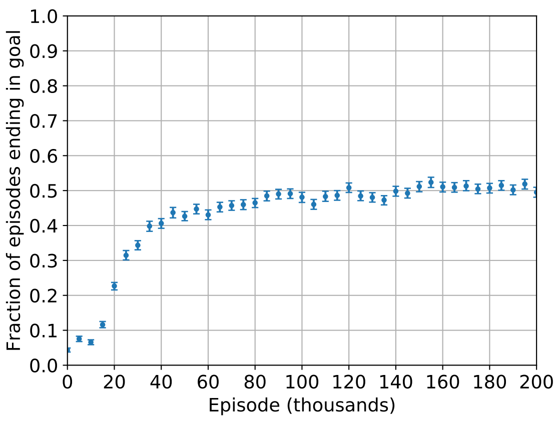

**Figure 5.17:** Training a policy for `YuShan` from scratch.

Some training instability can be observed when performing straightforward parameter sharing transfer learning (Figure 5.18 illustrates a particularly bad run of Figure 5.16). The following possible improvements to this approach are thus tested using the same `YuShan` case:

1. Using a larger batch size of 256 instead of 32. However, in order to compensate for the added computational cost and possibly reduce overfitting, the DQN updates are only performed every 4 time steps. Figure 5.19 presents a seemingly more steady growth than Figure 5.16, particularly in the first 20 thousand episodes, with only a small performance decrease (possibly related to the more infrequent updates).

2. Freezing the weights of first hidden layer of the DQN prevented the agent from improving (Figure 5.20). This may be related to the particular DQN topology that was used. The layer was kept frozen for the entire duration of the runs.

3. Reducing the learning rate to a fourth ($0.0000625/4$) resulted in smaller updates and thus a slower improvement (Figure 5.21), while still displaying some instability.

4. Initially using samples from the source policy did not help (Figure 5.22).

As such, we conclude that only using a larger batch size may present an advantage in this scenario. Some additional scenarios involving different teams (using the larger batch size) are present in Appendix A.5.

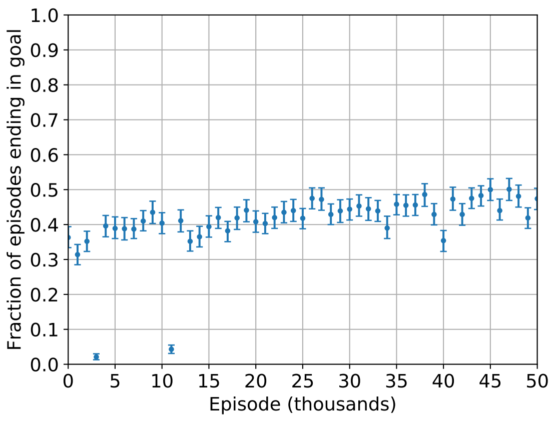

**Figure 5.18:** Adapting a policy from `AUT MasterMinds` to `YuShan`: a single run illustrating training instability.

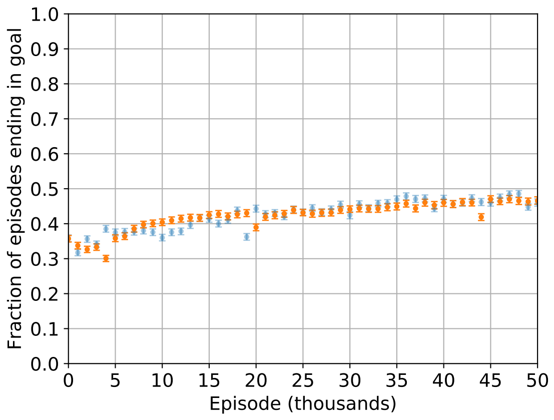

**Figure 5.19:** Adapting a policy from `AUT MasterMinds` to `YuShan`: batch size of 256 and training frequency of 4 time steps (Figure 5.16 repeated in blue).

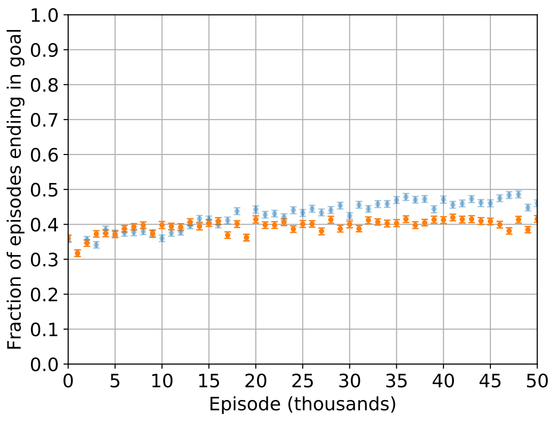

**Figure 5.20:** Adapting a policy from `AUT MasterMinds` to `YuShan`: freezing the weights of the first hidden layer of the DQN (Figure 5.16 repeated in blue).

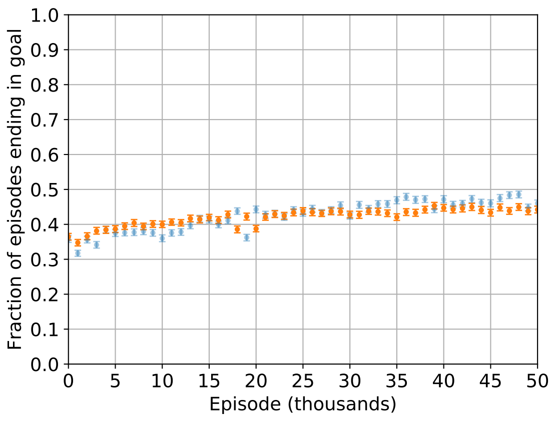

**Figure 5.21:** Adapting a policy from `AUT MasterMinds` to `YuShan`: reducing the learning rate to $0.0000625/4$ (Figure 5.16 repeated in blue).

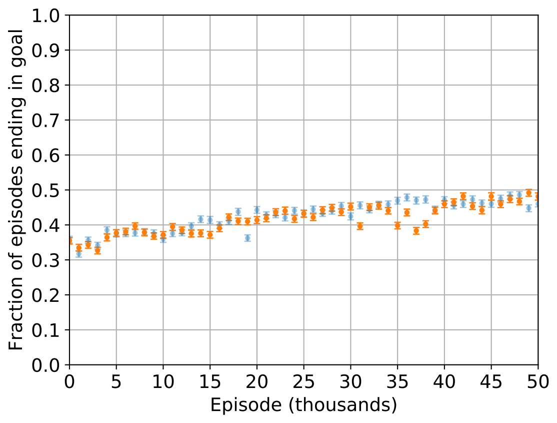

**Figure 5.22:** Adapting a policy from `AUT MasterMinds` to `YuShan`: using the replay buffer of the source policy as a starting point (Figure 5.16 repeated in blue).

# 6 Conclusion

**Contents**

This thesis aimed to address the problem of ad hoc teamwork when teammates may be unknown (never used for training). It was explored in the context of a two versus two match of HFO, where the ad hoc agent plays in the offense along with a teammate. For this purpose, an architecture based on PLASTIC-Policy [3] was developed. In this architecture, policies are learned for known teams via DQNs and added to a library of policies. A policy is then selected during execution with an unidentified team via a belief. Compared to PLASTIC-Policy, this architecture uses DQN instead of FQI for learning the policies, uses different mechanisms for identifying teammates (based on MLPs), extends these mechanisms to also identify if the team is unknown and adapts a policy from a known team to an unknown one.

To obtain a belief loss for a transition, two approaches were proposed: one based on predicting the next state (which depends on the team) and another based on directly discriminating its behavior among the known teams. These approaches are based on building a model using the transitions stored in the replay buffers of the DQNs. The discriminative approach proved to perform better, possibly for being easier to learn. However, it needs to be retrained every time a new team is added to the library. These belief losses are then combined across multiple transitions using an algorithm such as the update rule used in EWFs [53].

However, the original PLASTIC-Policy is unable to identify whether the unidentified team is in the library of known teams or not. To obtain the belief loss of a team being unknown these approaches cannot be used, as the transitions currently being observed are unlabeled (the agent does not know if they belong to previously encountered team or an unknown one). A simple approach based on an heuristic artificial belief was introduced, where the belief loss of the team being unknown is half of the sum of the known teams' belief losses: if a team has less than half of the total loss/more than half of the probability then the model is confident in it; otherwise, if all teams have less than half of the probability, then the model is not confident in any of them and the team probably is unknown. However, this approach is flawed since as more teams are introduced, there is less probability of the model identifying the same team consistently across multiple transitions, creating a bias for unknown.

If the ad hoc agent identifies a team as unknown, it then switches to a special policy improved online. As learning a new policy from scratch would be impractical in this setting, the ad hoc agent uses an existing policy as a source for parameter sharing transfer learning [12]. This source can be selected as the known team the agent considers to be most similar to the unknown team according to its belief (however, in Subsection 5.5 were chosen combinations of teams that allowed to observe the performance growth more easily). Several possible improvements were also tested, where increasing the batch size resulted in a more steady growth but in slightly reduced performance (possibly related to the more infrequent training that was used to compensate for the additional computational cost).

## 6.1 Limitations and future work

The first main limitation of this work is the simplistic unknown team detection. While it appears to work with some success in our experiments, its robustness has to be thoroughly tested. When there are many known teams it is expected that the model will have more varied predictions across multiple transitions, biasing the agent for the team being unknown. Future work on this area could involve applying more robust and well-studied techniques, namely in the field of anomaly detection [56]. Another possibility could be approaches based on the entropy of the transitions' belief losses.

The second main limitation of this work is the online policy adaptation. While the policy can remain reasonably stable, there was no meaningful observed improvement in the first moments (e.g. the first 1000 episodes), which are of greater importance in this setting. Future work in this area could involve studying some of the work mentioned in Yang Yu [49], namely using better exploration techniques [57] [21], using model-based methods for augmented DQN features [58] [59] and exploring meta-learning approaches [50].

# Bibliography

[1] P. MacAlpine, K. Genter, S. Barrett, and P. Stone, “The robocup 2013 drop-in player challenges: Experiments in ad hoc teamwork,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 382–387.

[2] P. Stone, G. A. Kaminka, S. Kraus, and J. S. Rosenschein, “Ad hoc autonomous agent teams: Collaboration without pre-coordination,” in *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence*, July 2010.

[3] S. Barrett, A. Rosenfeld, S. Kraus, and P. Stone, “Making friends on the fly: Cooperating with new teammates,” *Artificial Intelligence*, October 2016.

[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., ser. Adaptive Computation and Machine Learning series. USA: MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[5] S. Kalyanakrishnan, Y. Liu, and P. Stone, “Half field offense in RoboCup soccer: A multiagent reinforcement learning case study,” in *RoboCup-2006: Robot Soccer World Cup X*, ser. Lecture Notes in Artificial Intelligence, G. Lakemeyer, E. Sklar, D. Sorenti, and T. Takahashi, Eds. Berlin: Springer Verlag, 2007, vol. 4434, pp. 72–85.

[6] C. J. C. H. Watkins, “Learning from delayed rewards,” Ph.D. dissertation, King’s College, Cambridge, UK, May 1989. [Online]. Available: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

[7] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[9] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017.

[10] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia

Laguna Resort, Sardinia, Italy: JMLR Workshop and Conference Proceedings, 13–15 May 2010, pp. 249–256. [Online]. Available: http://proceedings.mlr.press/v9/glorot10a.html

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," 2015.

[12] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2021.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[14] P. J. Huber, "Robust estimation of a location parameter," *Ann. Math. Statist.*, vol. 35, no. 1, pp. 73–101, 03 1964. [Online]. Available: https://doi.org/10.1214/aoms/1177703732

[15] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," 2015.

[16] H. V. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 2613–2621. [Online]. Available: http://papers.nips.cc/paper/3964-double-q-learning.pdf

[17] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015.

[18] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," 2015.

[19] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," 2017.

[20] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," 2017.

[21] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg, "Noisy networks for exploration," 2019.

[22] M. Lanctot, E. Lockhart, J.-B. Lespiau, V. Zambaldi, S. Upadhyay, J. Pérolat, S. Srinivasan, F. Timbers, K. Tuyls, S. Omidshafiei, D. Hennes, D. Morrill, P. Muller, T. Ewalds, R. Faulkner, J. Kramár, B. D. Vylder, B. Saeta, J. Bradbury, D. Ding, S. Borgeaud, M. Lai, J. Schrittwieser, T. Anthony, E. Hughes, I. Danihelka, and J. Ryan-Davis, "Openspiel: A framework for reinforcement learning in games," 2020.

[23] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara, “Robocup: A challenge problem for ai,” *AI Magazine*, vol. 18, no. 1, p. 73, Mar. 1997. [Online]. Available: https://www.aaai.org/ojs/index.php/aimagazine/article/view/1276

[24] P. Stone, R. S. Sutton, and G. Kuhlmann, “Reinforcement learning for RoboCup-soccer keepaway,” *Adaptive Behavior*, vol. 13, no. 3, pp. 165–188, 2005.

[25] M. Hausknecht, P. Mupparaju, S. Subramanian, S. Kalyanakrishnan, and P. Stone, “Half field offense: An environment for multiagent learning and ad hoc teamwork,” in *AAMAS Adaptive Learning Agents (ALA) Workshop*, May 2016.

[26] H. Akiyama, T. Nakashima, and K. Yamashita, “Helios2013 team description paper,” *RoboCup*, 2013.

[27] H. Akiyama and T. Nakashima, “Helios base: An open source package for the robocup soccer 2d simulation,” in *RoboCup 2013: Robot World Cup XVII*, S. Behnke, M. Veloso, A. Visser, and R. Xiong, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 528–535.

[28] M. Bowling and P. McCracken, “Coordination and adaptation in impromptu teams,” in *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*, 2005, pp. 53–58.

[29] M. B. Dias, T. K. Harris, B. Browning, E. G. Jones, B. Argall, M. M. Veloso, A. Stentz, and A. I. Rudnicky, “Dynamically formed human-robot teams performing coordinated tasks,” in *AAAI Spring Symposium: To Boldly Go Where No Human-Robot Team Has Gone Before*, 2006.

[30] P. Stone and S. Kraus, “To teach or not to teach? decision making under uncertainty in ad hoc teams,” in *The Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, May 2010.

[31] S. Barrett, “Making friends on the fly: Advances in ad hoc teamwork,” Ph.D. dissertation, The University of Texas at Austin, Austin, Texas, USA, December 2014.

[32] D. Ernst, P. Geurts, and L. Wehenkel, “Tree-based batch mode reinforcement learning,” *J. Mach. Learn. Res.*, vol. 6, pp. 503–556, dec 2005.

[33] A. Blum and Y. Mansour, *Learning, Regret Minimization, and Equilibria*. Cambridge University Press, 2007, p. 79–102.

[34] P. Stone, G. A. Kaminka, and J. S. Rosenschein, “Leading a best-response teammate in an ad hoc team,” in *Agent-Mediated Electronic Commerce: Designing Trading Strategies and Mechanisms for Electronic Markets*, E. David, E. Gerding, D. Sarne, and O. Shehory, Eds. Springer Verlag, November 2010, pp. 132–146.

[35] N. Agmon and P. Stone, “Leading ad hoc agents in joint action settings with multiple teammates,” in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, ser. AAMAS ’12. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 341–348.

[36] D. Chakraborty and P. Stone, “Cooperating with a markovian ad hoc teammate,” in *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2013.

[37] S. V. Albrecht and S. Ramamoorthy, “A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems,” in *AAMAS*, 2013.

[38] ——, “On convergence and optimality of best-response learning with policy types in multiagent systems,” in *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, ser. UAI’14. Arlington, Virginia, United States: AUAI Press, 2014, pp. 12–21.

[39] S. Albrecht and P. Stone, “Reasoning about hypothetical agent behaviours and their parameters,” in *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-17)*, May 2017.

[40] ——, “Autonomous agents modelling other agents: A comprehensive survey and open problems,” *Artificial Intelligence*, vol. 258, pp. 66–95, 2018.

[41] M. Ravula, S. Alkobi, and P. Stone, “Ad hoc teamwork with behavior switching agents,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, August 2019.

[42] F. Wu, S. Zilberstein, and X. Chen, “Online planning for ad hoc autonomous agent teams,” in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume One*, ser. IJCAI’11. AAAI Press, 2011, pp. 439–445. [Online]. Available: http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-081

[43] R. Coulom, “Efficient selectivity and backup operators in monte-carlo tree search,” in *Proceedings of the 5th International Conference on Computers and Games*, ser. CG’06. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 72–83.

[44] F. S. Melo and A. Sardinha, “Ad hoc teamwork by learning teammates’ task,” *Autonomous Agents and Multi-Agent Systems*, vol. 30, no. 2, pp. 175–219, Mar 2016. [Online]. Available: https://doi.org/10.1007/s10458-015-9280-x

[45] T. Hester and P. Stone, “Texplore: real-time sample-efficient reinforcement learning for robots,” *Machine Learning*, vol. 90, no. 3, pp. 385–429, Mar 2013. [Online]. Available: https://doi.org/10.1007/s10994-012-5322-7

[46] L. Kocsis and C. Szepesvári, “Bandit based monte-carlo planning,” in *Proceedings of the 17th European Conference on Machine Learning*, ser. ECML'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 282–293. [Online]. Available: http://dx.doi.org/10.1007/11871842_29

[47] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, “Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 7559–7566.

[48] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 1889–1897. [Online]. Available: http://proceedings.mlr.press/v37/schulman15.html

[49] Y. Yu, “Towards sample efficient reinforcement learning,” in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 7 2018, pp. 5739–5743. [Online]. Available: https://doi.org/10.24963/ijcai.2018/820

[50] C. Finn, P. Abbeel, and S. Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 1126–1135. [Online]. Available: http://proceedings.mlr.press/v70/finn17a.html

[51] S. Barrett, P. Stone, S. Kraus, and A. Rosenfeld, “Teamwork with limited knowledge of teammates,” in *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, July 2013.

[52] A. Lazaric, M. Restelli, and A. Bonarini, “Transfer of samples in batch reinforcement learning,” in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 544–551. [Online]. Available: https://doi.org/10.1145/1390156.1390225

[53] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[54] B. Efron, “Bootstrap methods: Another look at the jackknife,” *Ann. Statist.*, vol. 7, no. 1, pp. 1–26, 01 1979. [Online]. Available: https://doi.org/10.1214/aos/1176344552

[55] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[56] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, Jul. 2009. [Online]. Available: https://doi.org/10.1145/1541880.1541882

[57] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter space noise for exploration," 2018.

[58] V. Pong*, S. Gu*, M. Dalal, and S. Levine, "Temporal difference models: Model-free deep RL for model-based control," in *International Conference on Learning Representations*, 2018.

[59] S. Racanière, T. Weber, D. Reichert, L. Buesing, A. Guez, D. Jimenez Rezende, A. Puigdomènech Badia, O. Vinyals, N. Heess, Y. Li, R. Pascanu, P. Battaglia, D. Hassabis, D. Silver, and D. Wierstra, "Imagination-augmented agents for deep reinforcement learning," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017, pp. 5690–5701. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/9e82757e9a1c12cb710ad680db11f6f1-Paper.pdf

# A

# Extended Results

## Contents

## A.1 Policies

Figures A.1 to A.6 present the fraction of episodes ending in goal for every team, tested every 5000 training episodes for 500 episodes without exploration and without changing the DQN weights.

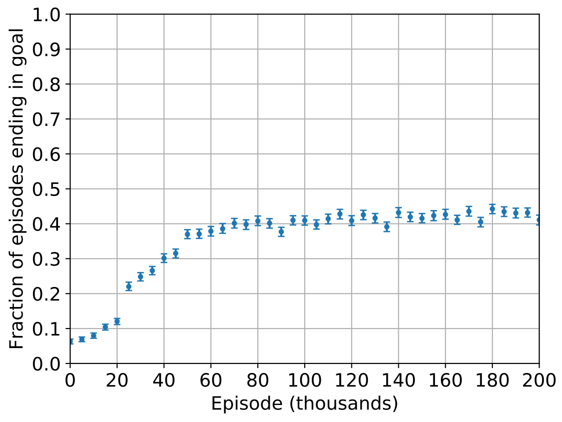

**Figure A.1:** Policy training for `AUT MasterMinds`.

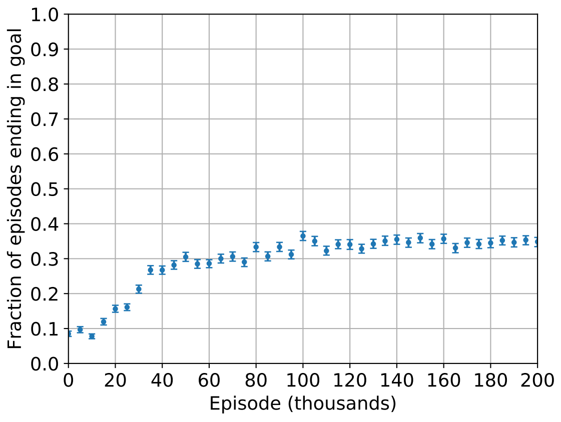

**Figure A.2:** Policy training for `Base`.

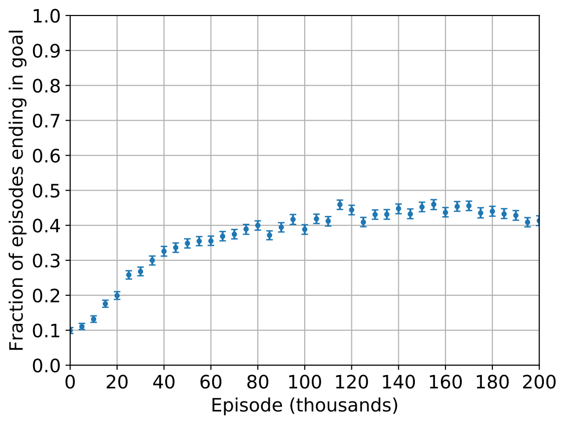

**Figure A.3:** Policy training for `Cyrus`.

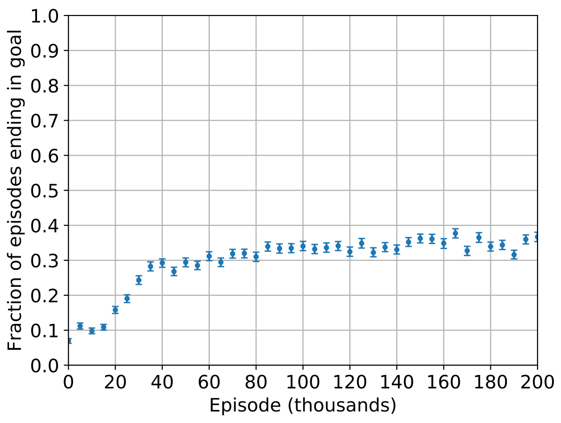

**Figure A.4:** Policy training for `Gliders`.

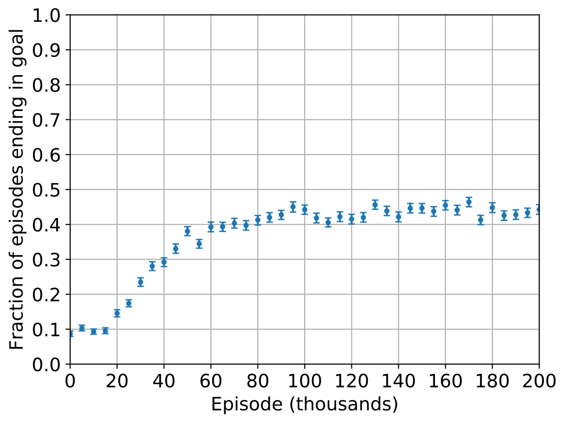

**Figure A.5:** Policy training for `HELIOS`.

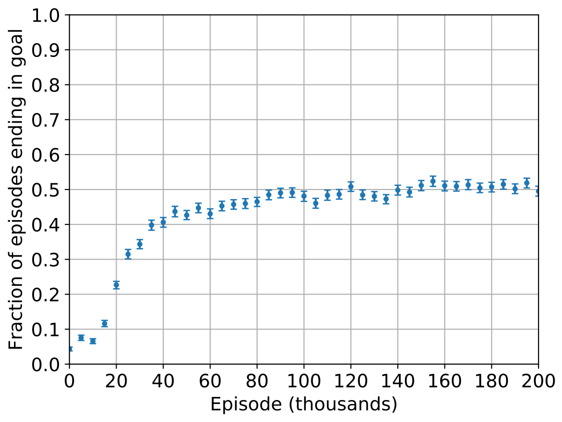

**Figure A.6:** Policy training for `YuShan`.

## A.2 Policy selection when all teams are known

Figures A.7 to A.12 present the fraction of trials with maximum belief (where the agent identifies the team as the one that has maximum belief) for all teams, when all teams are known. The update rule used in EWFs with $\eta = 1$ and the discriminative approach are used.

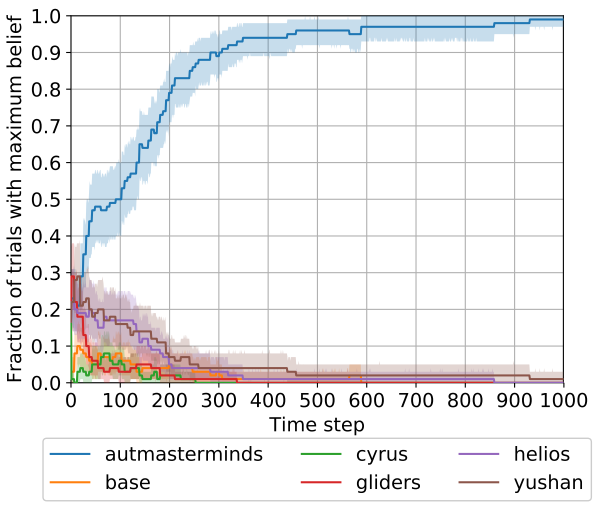

**Figure A.7:** Policy selection for `AUT MasterMinds`.

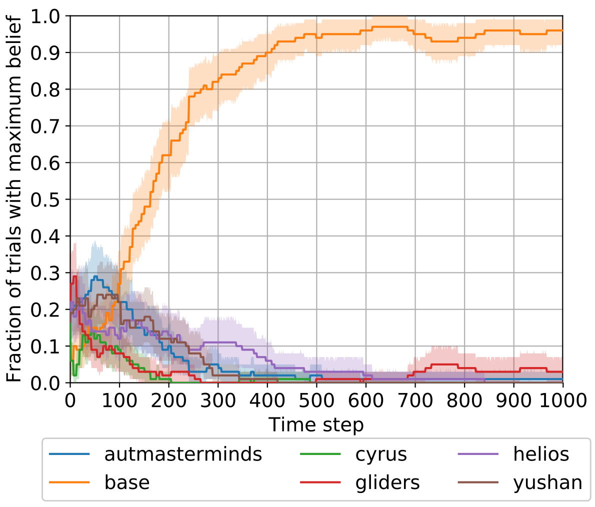

**Figure A.8:** Policy selection for `Base`.

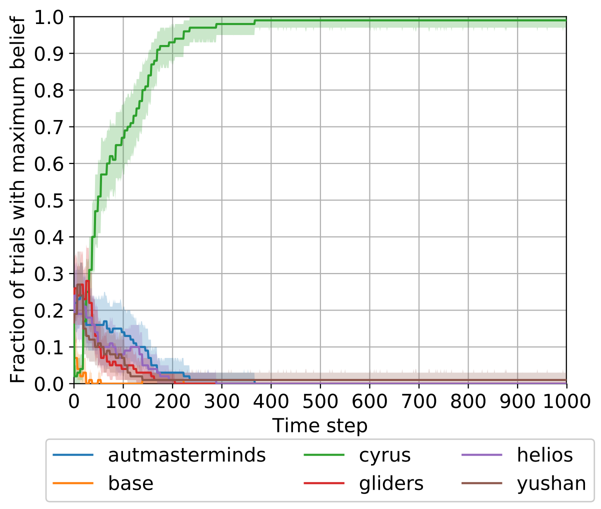

**Figure A.9:** Policy selection for `Cyrus`.

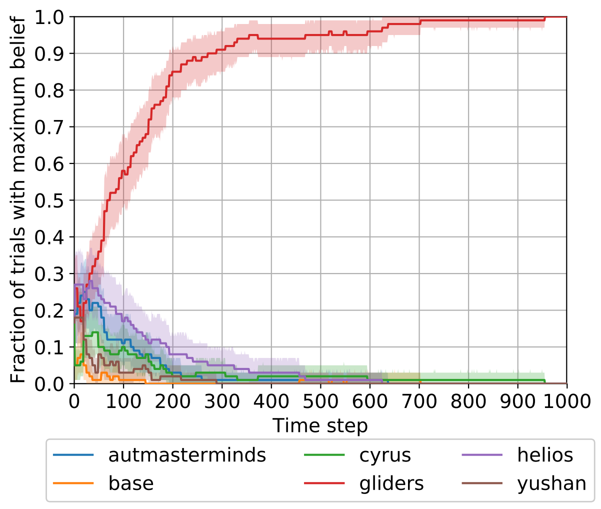

**Figure A.10:** Policy selection for `Gliders`.

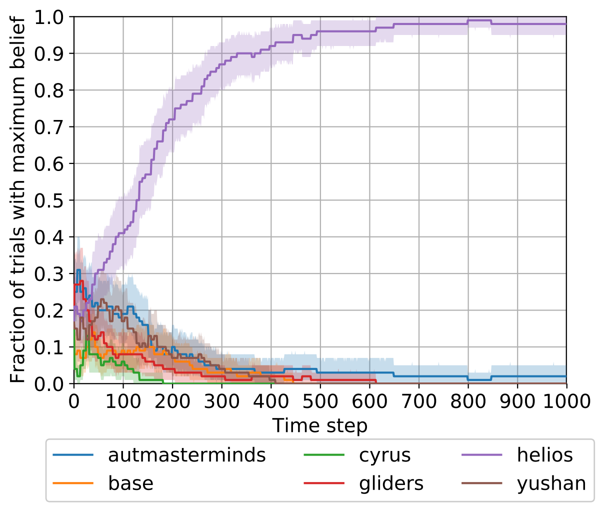

**Figure A.11:** Policy selection for `HELIOS`.

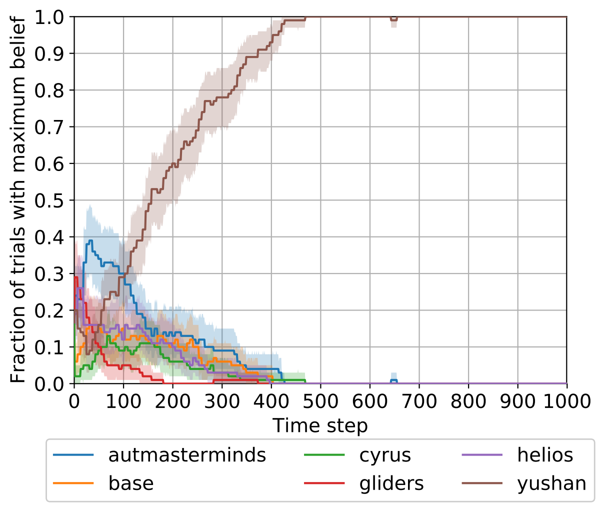

**Figure A.12:** Policy selection for `YuShan`.

## A.3 Policy selection with three known teams

Figures A.13 to A.18 present the fraction of trials with maximum belief (where the agent identifies the team as the one that has maximum belief) for all teams, where `AUT MasterMinds`, `Base` and `Cyrus` are known and `Gliders`, `HELIOS` and `YuShan` are unknown. The update rule used in EWFs with $\eta = 1$ and the discriminative approach are used.

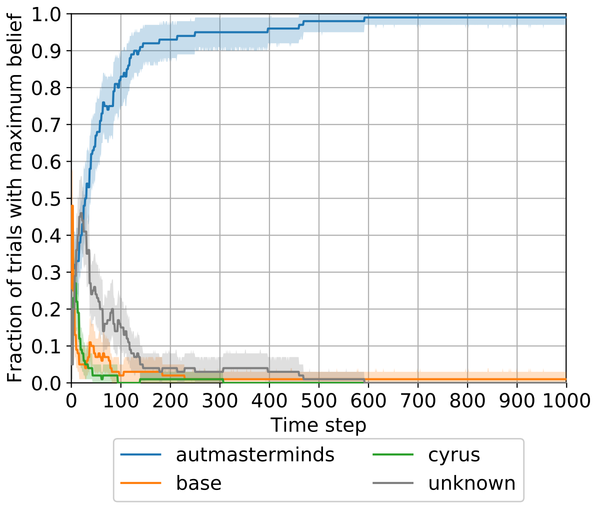

**Figure A.13:** Policy selection for `AUT MasterMinds` (known) with three known teams.

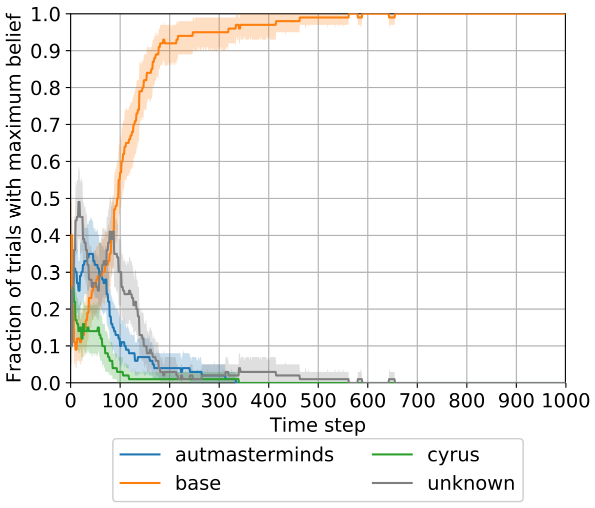

**Figure A.14:** Policy selection for `Base` (known) with three known teams.

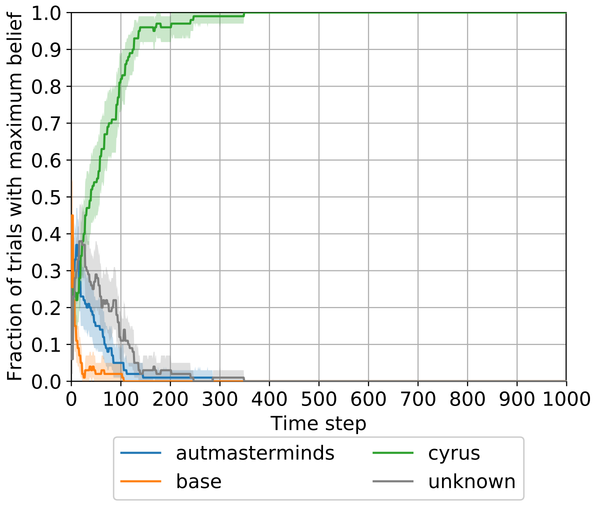

**Figure A.15:** Policy selection for `Cyrus` (known) with three known teams.

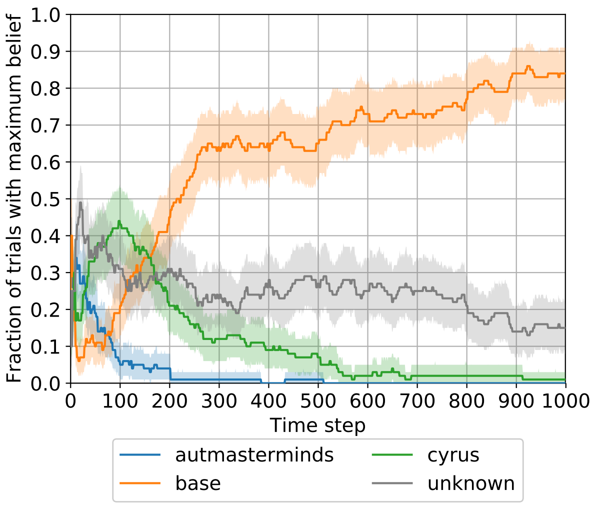

**Figure A.16:** Policy selection for `Gliders` (unknown; misidentified as `Base`) with three known teams.

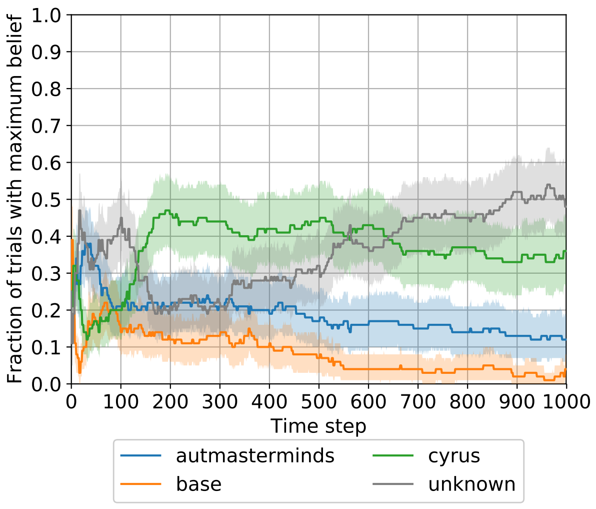

**Figure A.17:** Policy selection for `HELIOS` (unknown) with three known teams.

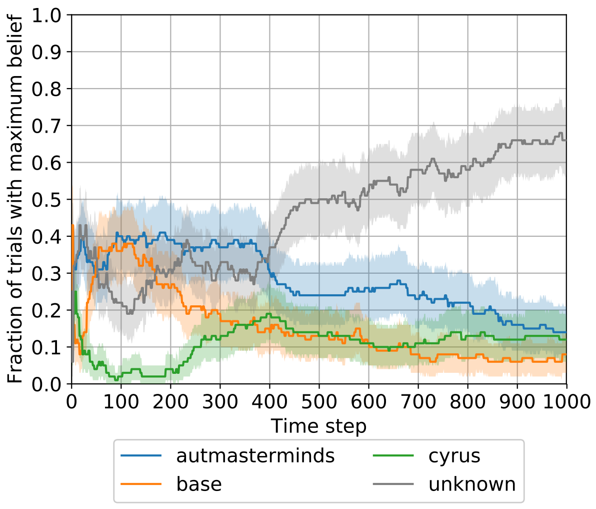

**Figure A.18:** Policy selection for `YuShan` (unknown) with three known teams.

## A.4 Policy selection with five known teams

Figures A.19 to A.24 present the fraction of trials with maximum belief (where the agent identifies the team as the one that has maximum belief) for all teams, where `AUT MasterMinds`, `Base`, `Cyrus`, `HELIOS` and `YuShan` are known and `Gliders` is unknown. The update rule used in EWFs with $\eta = 1$ and the discriminative approach are used.

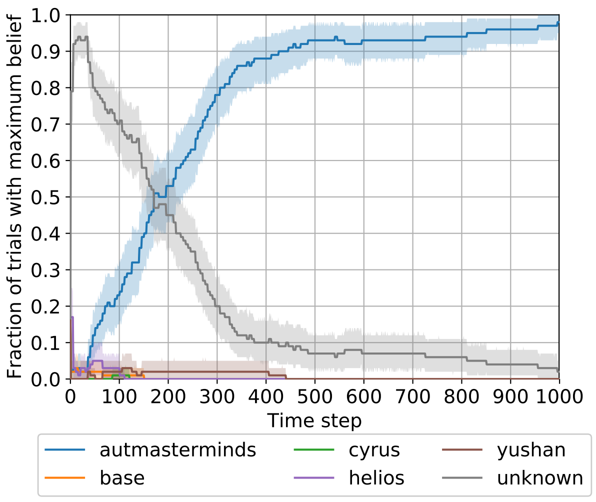

**Figure A.19:** Policy selection for `AUT MasterMinds` (known) with five known teams.

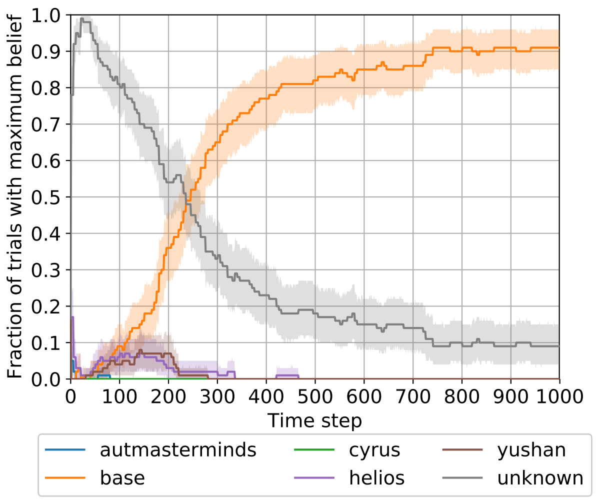

**Figure A.20:** Policy selection for `Base` (known) with five known teams.

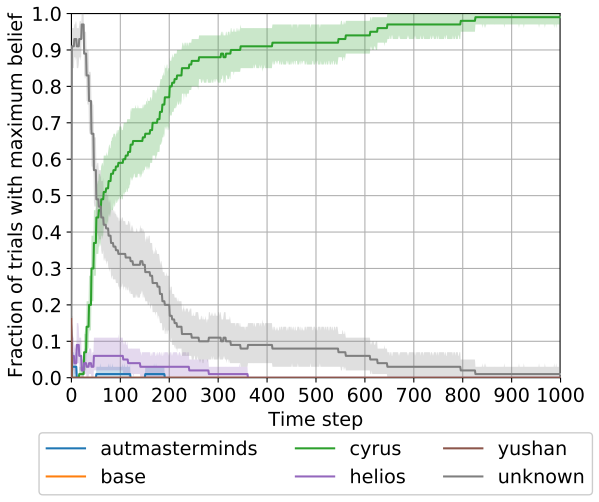

**Figure A.21:** Policy selection for `Cyrus` (known) with five known teams.

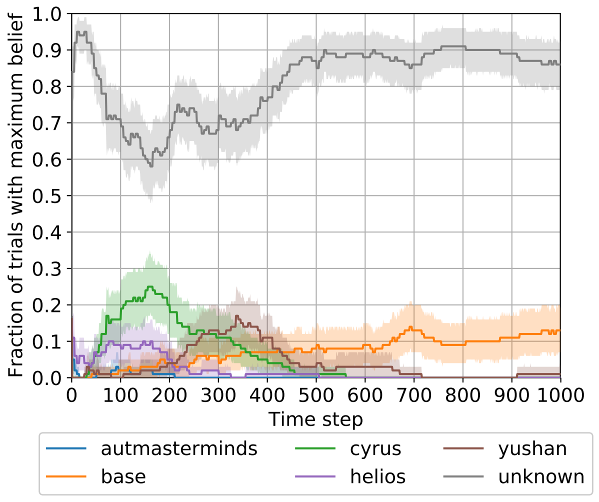

**Figure A.22:** Policy selection for `Gliders` (unknown) with five known teams.

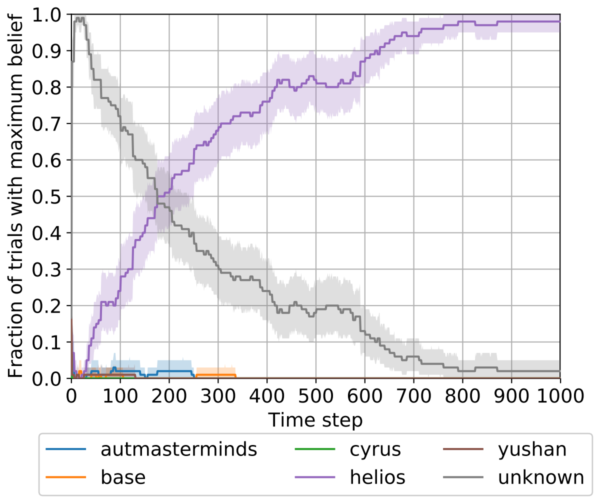

**Figure A.23:** Policy selection for `HELIOS` (known) with five known teams.

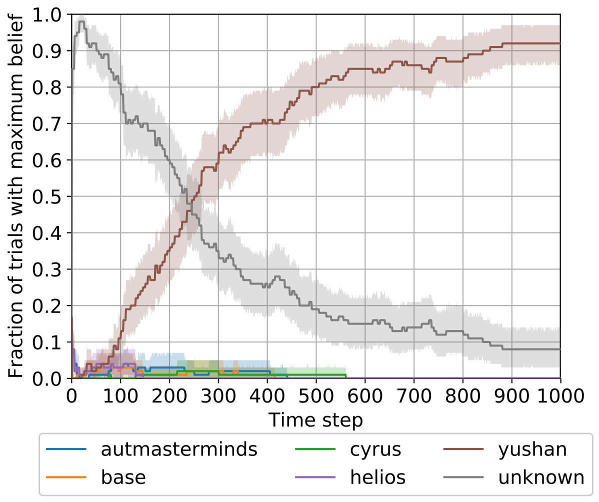

**Figure A.24:** Policy selection for `YuShan` (known) with five known teams.

## A.5 Additional online policy adaptation experiments

Figures A.25 to A.28 present the fraction of episodes ending in goal when the agent adapts a policy from one teammate to a different teammate. The results can be compared to Table 5.1.

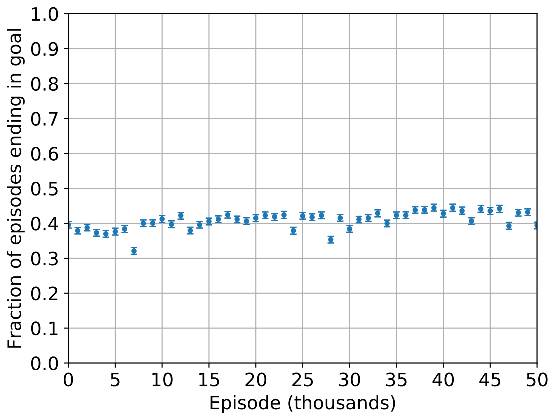

**Figure A.25:** Adapting a policy from `Base` to `HELIOS`: batch size of 256 and training frequency of 4 time steps.

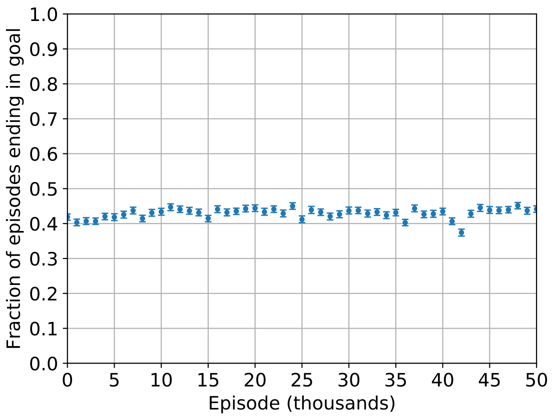

**Figure A.26:** Adapting a policy from `Gliders` to `Cyrus`: batch size of 256 and training frequency of 4 time steps.

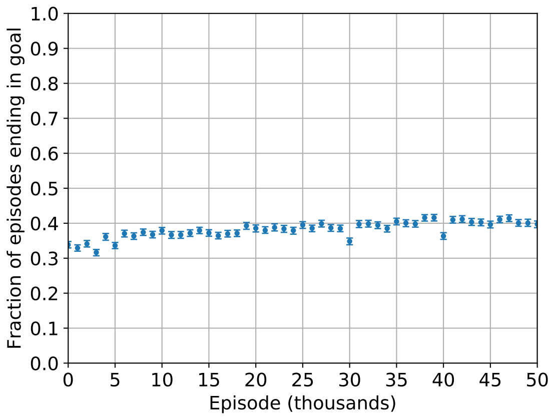

**Figure A.27:** Adapting a policy from `YuShan` to `AUT MasterMinds`: batch size of 256 and training frequency of 4 time steps.

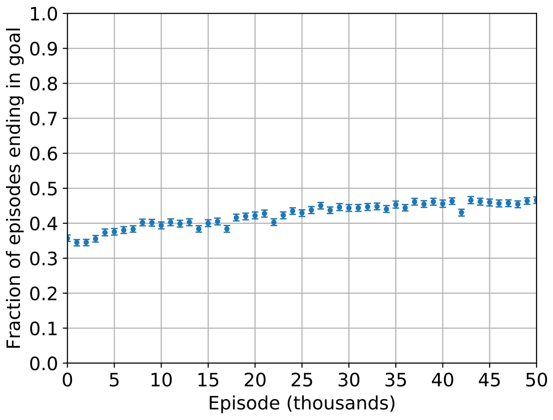

**Figure A.28:** Adapting a policy from `YuShan` to `HELIOS`: batch size of 256 and training frequency of 4 time steps.