Juliana do Nascimento Damurie da Silva

Evaluation of Time Limits in Reinforcement Learning Applied to Robot Soccer Simulation

Federal University of Pernambuco
graduacao@cin.ufpe.br
www.cin.ufpe.br/~secgrad

Recife
2021

Juliana do Nascimento Damurie da Silva

Evaluation of Time Limits in Reinforcement Learning Applied to Robot Soccer Simulation

A B.Sc. Dissertation presented to the Center of Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Bachelor in Computer Engineering.

Concentration Area: Reinforcement Learning and Robotics
Advisor: Hansenclever de França Bassani

Recife
2021

AGRADECIMENTOS

Eu quero agradecer ao professor Hansenclever Bassani, pela ajuda para a resolução de problemas e acompanhamento no desenvolvimento do presente trabalho, e também por todo o acompanhamento no RobôCIn.

Também quero agradecer a minha família por todo apoio e base, que me fez chegar até esse ponto. Pelas mulheres fortes que constituem essa família, em especial a minha avó que conseguiu ser uma base sólida dessa família. Ao meu Pai que sempre me incentivou a estudar e sempre me desafiou a ser melhor. A minha mãe que sempre tentou e me deu as melhores oportunidades na vida, e sempre lutou muito por mim.

Agradeço ao Evaristo Luís, por todos os conselhos e suporte durante esse último ano, obrigada por me passar segurança nos momentos mais inseguros.

Aos meus amigos de curso: Eduardo, Letícia, Luana, Matheus, Marcos, Lucas Gabriel, Lucas Pontes, Adrion, Sergio, Thyago, Douglas, Lucas Augusto e Miguel, que me ajudaram e deram suporte na trajetória deste curso, vocês são muito importantes para mim.

Ao RobôCIn, em especial a professora Edna Barros que sempre ofereceu suporte para nos desenvolvermos os nossos conhecimentos e sonhos. Aos meus amigos de equipe: Renato Sousa, Roberto Fernandes, Walber Macedo, Cecília Vírginia, Lucas Cavalcanti, Felipe Martins, Mateus Gonçalves, Raphael Britto, João Gabriel, Thiago Araújo, José Victor, José Douglas, João Amaro, Carlos Caloete e Victor Sabino. Agradeço também a Pedro Braga por todos os conselhos e inspirações.

E por fim, um agradecimento especial a equipe de reinforcement learning do RobôCIn que me deram todo o apoio durante a produção deste trabalho.

"Nothing in life is to be feared; it is only to be understood."

–Marie Curie

ABSTRACT

The advances in robotics and artificial intelligence research are fundamental to the development of new technologies, and one of the drivers of these advances are the robotics competitions. As an example of advancement, we have simulators in robotics and machine learning that allow us to study the behavior of agents in complex scenarios in a controlled way. Reinforcement Learning (RL) is already applied in several robotic simulators to learn tasks in more optimized ways than the classical algorithms. In RL, the goal of the agent is to optimize the reward accumulated over an episode. However, it is usual to set a time for the agent to interact with that environment. This time can often be, in fact, a limiting factor for that task or just a way to diversify experiences. Then, we can divide tasks into episodic tasks and continuous tasks. This characteristic impacts how the agent should accumulate these rewards, how it terminal states, and how the value function is computed. In episodic tasks, time is often a critical factor in solving the problem. So the question arises as to when we should add the notion of time into the observations and when this time limit changes the agent's behavior in the environment. Therefore, we aim to analyze the impact of agent modeling from the remaining time of tasks in the robot soccer simulation, thus understanding how important it is to model the time in a simulated robot soccer environment.The evaluation was done by training two different agents in the same environment, one with time awareness and one without, using the reinforcement learning algorithm Deep Deterministic Policy Gradient (DDPG), performing tests with different parameters, and testing after training in different static scenarios. The experimental results showed that both agents can have good results depending on the scenario in which they are inserted. The time-aware agent can perform better when we have a critical time limit to perform the task.

Keywords: Reinforcement Learning . Time Limits. Robotics Simulation. Robot Soccer.

RESUMO

Os avanços nas pesquisas de robótica e inteligência artificial são fundamentais para o desenvolvimento de novas tecnologias, e um dos propulsores desses avanços são as competições de robótica. Como exemplo temos os simuladores na robótica e a aprendizagem de máquina que nos permitem testar comportamentos complexos de uma forma controlada. A aprendizagem por reforço já é aplicada em diversos simuladores robóticos como forma de aprender tarefas de formas mais otimizadas que os algoritmos clássicos. Na aprendizagem por reforço, o objetivo do agente é otimizar a recompensa acumulada ao longo de um episódio, porém é comum fixar um tempo para o agente interagir com aquele ambiente, esse tempo muitas vezes pode ser de fato um limitante para aquela tarefa ou apenas uma forma de diversificar experiências. Então, podemos dividir as tarefas em tarefas episódicas e tarefas contínuas. Essas características impactam na forma em que devemos acumular essas recompensas, como devemos observar os estados terminais e como devemos calcular a função de valor. Nas tarefas episódicas, muitas vezes o tempo é um fator crítico para resolução do problema e então surge o questionamento de quando devemos adicionar a noção do tempo nas observações, e quando esse limite de tempo de fato muda o comportamento do agente no ambiente. Portanto, o objetivo deste trabalho é realizar uma análise sobre o impacto da modelagem do agente a partir das características de tempo das tarefas aplicadas na simulação de futebol de robôs, assim extraindo informações de quais são as melhores modelagens para o ambiente de simulação de robôs. A avaliação foi feita treinando dois tipos de agentes em um mesmo ambiente, um com noção do tempo e outro sem. Com o algoritmo de reinforcement learning *Deep Deterministic Policy Gradient (DDPG)* e realizando teste com diferentes parâmetros e também testando após o treinamento em diferentes cenários estáticos. Os resultados experimentais demonstraram que ambos os agentes conseguem ter bons resultados a depender do cenário em que estão inseridos, e que o agente com noção do tempo consegue ter um melhor desempenho quando temos um limite de tempo crítico para realizar a tarefa.

Palavras-chave: Aprendizagem de máquina. Limites de Tempo. Simulação Robótica. Futebol de Robôs.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

DDPG	Deep Deterministic Policy Gradient
DP	Dynamic Programming
MDP	Markov Decision Process
MSE	Mean Squared Error
RL	Reinforcement Learning
SARSA	State-Action-Reward-State-Action
SSL	Small Size League

LIST OF ALGORITHMS

CONTENTS

1

INTRODUCTION

Robotics and artificial intelligence competitions have significantly boosted research in the respective areas in recent years, thus defining annual challenges in different categories, bringing advances and promoting the growth of the scientific community in these fields. In Robot Soccer, we can mention the main competition at the world level, the RoboCup.

The RoboCup emerged in 1997 to start a world league with real robots and aim to have until 2050 a complete team of autonomous humanoid robots to win a match of the most recent FIFA World Cup champion [Weitzenfeld *et al.*, 2015].

The Small Size League (SSL), also known as F-180, is one of Robocup's main robot soccer categories. In the SLL, two teams (one blue, one yellow) with a number of 6 to 11 robots depending on the team's division, each of these robots can be a maximum of 180mm in diameter and 150mm in height, facing each other in a soccer match. The ball for the game is a standard golf ball, approximately 43mm in diameter. In Figure 1, we have a picture of a robot from this category.

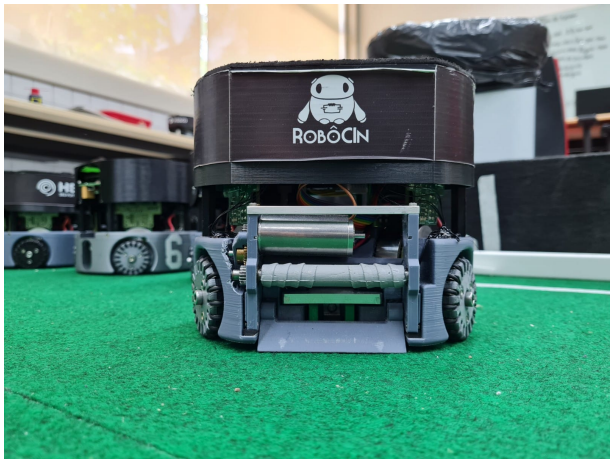


Figure 1: Robot model from the Small Size League

The objects on the field (robots and the ball) are identified by a unified vision system that sends information such as robot positions and ball position to the computer. Then each team uses artificial intelligence to control these robots without human intervention. The way the vision system works can be seen in Figure 2.

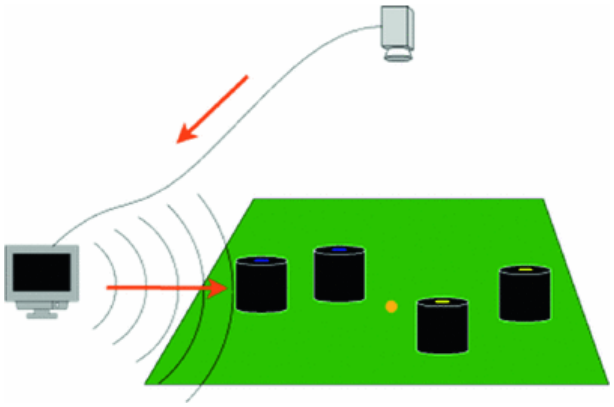


Figure 2: Structure of SSL competition. Source: [Weitzenfeld *et al.*, 2015]

The use of simulation in robotics has an important function since it permits the development and validation inexpensively and safely [Afzal *et al.*, 2021][Ibarz *et al.*, 2021]. The execution of exhaustive tests is also something that in simulation is easier and often avoids a stress of the mechanical and electronic part of the robot. Although there is usually a reality gap between the policies learned in simulation and the policies applied to the real robot, simulations in machine learning are essential for the evolution of such applications in robotics Barrett *et al.* [2010].

There are already fully simulated categories in RoboCup, like the "Simulation 2D league" and the "Simulation 3D League". There are also simulators of the competitions that require a physical robot to offer the team the opportunity to test their algorithms quickly and at a low cost because many teams cannot have the physical structure of the competition in their labs.

Reinforcement Learning (RL) is a subcategory of machine learning in which learning happens from interactions between the agent and the environment, and the agent learns how to map observations from the environment into actions [Sutton & Barto, 2018]. We can compare RL to training a dog by giving a treat as a reward when it does a correct action and not giving when it does something incorrectly [Ravichandiran, 2018]. Therefore, as in the dog training, we do not teach the agent what it should do, but it learns by discovering which actions return good rewards.

We can explain the flow of learning in RL as, at each interaction t, the agent sends an action A_t to the environment, the environment performs the action and then returns to the agent a new state S_{t+1} and also a reinforcement signal the reward R_{t+1} that represents the impact of previous actions on the environment, which can have been positive, negative or neutral.

In the last years, reinforcement learning methods have been in evidence in the literature for their expressive results in the ability to learn complex activities and performing better than humans, such as games like Chess [Campbell *et al.*, 2002], Go [Silver *et al.*, 2017], Atari [Mnih *et al.*, 2013], Starcraft 2 [Vinyals *et al.*, 2017], Dota [Berner *et al.*, 2019].

In robot soccer simulation, reinforcement learning has been present in RoboCup for years, as we can see in Stone & Sutton [2001] application. It is still a challenge to apply these

methods in real robots, so simulators and frameworks that can provide environments closer to the real world are frequently employed. This development of simulators and frameworks is important for applying machine learning methods to real robots. As an example we have the Very Small Size Soccer competition, for which an agent trained in simulation was successfully transferred to a real world competition [Bassani *et al.*, 2020].

Therefore, as in most traditional machine learning algorithms, the choice of hyper-parameters and the architecture design directly impact the results obtained, in reinforcement learning, it is no different. Therefore, defining and characterizing the task that the agent needs to do and the architecture is essential to obtain an optimal policy. Sutton & Barto [2018] defines two types of tasks: those that must be optimized in a fixed period of time and those that must be optimized over an indefinite period of time and the time limit is only used during training to diversify the experiments.

Besides the division of tasks into episodic and continuous is an old definition, this topic is not much explored in the literature. We have articles like Harada [1997] makes a study on the addition of the notion of time in finite horizon problems. Also, Pardo *et al.* [2018] explores the limitation of time in a more general way for both episodic and continuous tasks and the most recent Reinke [2020] introduces a specific algorithm for solving problems with time restrictions.

To exemplify the importance of time in some tasks, we have the example of the last-moment problem in Pardo *et al.* [2018]. Imagine that we have a map with two positions: position A and position B, the agent always starts at position A. If he stays in position A, he receives 0 reward. If he jumps to position B, he gets a +1 reward, but from position B, he only has the option to stay in position B, and each step that he stays in position B receives a reward of -1. Imagine also that the episode ends after T steps. The optimal solution for this agent is to jump to position B only when it is at step T and stay T-1 steps at position A. However if the agent is not time aware he would not be able to solve the problem and the best option for him would be to stay T steps at position A.

It is easy to find situations where time is critical to decide what action to take in the sports environment. We can observe some behaviors, such as in the final of a football match when there are only a few minutes to finish the match and the whole team, including the goalkeeper, sometimes move to the attacking area in an attempt to score that last goal. Alternatively, in basketball, where the last seconds of the game the players try to attack as much as possible and there is also the rule of the 'buzzer beater. Which is a ball that can count as a shot after the end of the game, provided it has been thrown before the final buzzer.

Therefore, performing a study on time limits and applying them in the context of robot soccer is important because, in robot soccer, we also have a time limit during the match. In challenges and tasks performed in these competitions, there is also a time limit.

As mentioned before Pardo *et al.* [2018], as a way to solve these tasks, the authors define two types of agent: time-aware and partial episode bootstrapping, one to solve the problems presented in the time-limited tasks and the other for time-unlimited tasks.

This work has as a general objective of applying the methods developed by Pardo *et al.* [2018], specifically related to the time limit and variations of this method in the context of robot soccer to create different types of agents and conduct a study on their performance. Furthermore, contribute to the advancement of further studies of applications of these methods of time limits in robot soccer.

As specific objectives, this work aims to:

- Test the methods of time limits in a classical environment, gridWorld;
- Analyze the impact of the use of different parameters in the robot soccer environment combined with the insertion of the notion of time in the agent;
- Analyze agents' behavior with and without a notion of time in static scenarios in a robot soccer environment.

This work was separated into 5 chapters. In Chapter 2, we introduce the main concepts used in this work, then we will present the concepts of RL and concepts about the time limit. In Chapter 3, we explain how the contributions of this work were implemented and how the tests we propose to do were performed. In Chapter 4, we will present the results obtained in this work using the methods presented in Chapter 3. In Chapter 5, we present the final considerations and conclusions obtained in this work.

2

THEORETICAL BACKGROUND

2.1 REINFORCEMENT LEARNING

RL is a branch of machine learning that aims to learn how to map observations to actions in an environment to maximize a numerical reward. RL is accomplished through interactions between the environment and the agent. It is not classified like the other machine learning paradigms, usually divided in supervised, unsupervised, and semisupervised learning.

To develop an RL model, we have the following essential elements: the agent, the environment model, the reward signal, the policy, the value functions, and the return. The agent learns which actions will lead to a better reward in the future from environmental observations and actions taken by the agent. The environment model represents the real environment in which the agent will apply its actions. The reward signal is the signal provided by the environment from an action that was performed on it and can be positive, neutral, or negative. The policy defines the agent's behavior in the environment, in other words, the policy takes an observation of the environment as input and provides the action to be taken as output. The value functions represent the estimated reward that will be accumulated by the agent from a certain state $v_\pi(s)$ or from a state-action pair $q_\pi(s,a)$ to the end of the episode.

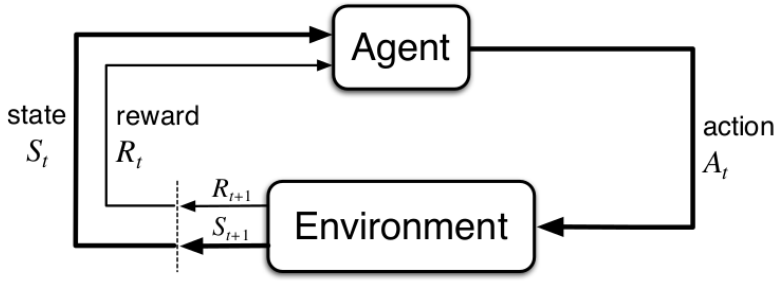


Figure 3: Basic model of the interaction between agent and environment in Reinforcement Learning. Source: Sutton & Barto [2018].

In Figure 3, we can see how the Agent-Environment interface works. The agent receives the observation state as a representation of the environment, and based on the policy, it defines the following action, and this action is returned to the environment. The agent sends the new

action that causes a state change in the environment, and then the environment returns a new observation state and a reward for that new state [Sutton & Barto, 2018].

Most reinforcement learning problems can be modeled as a Markov Decision Process (MDP) [Graesser & Keng, 2019]. An MDP is represented by 4-tuple $M =< S, A, P(.), R(.) >$, where:

- $\mathbb{S}$ is a set of states;
- $\mathbb{A}$ is a set of actions;
- $\mathbb{P}(.)$ is the probability of transition;
- $\mathbb{R}(.)$ is the immediate reward after a transition.

In RL, we try to obtain an optimal policy even when the perfect model is unavailable. Therefore, the only way for the agent to access information about the transition function and the reward function is to perform experiments in the environment [Graesser & Keng, 2019].

The agent's purpose is not to maximize the immediate reward but to maximize the total reward accumulated over a series of actions. This function is called return, denoted by G_t, which is defined in the simplest case as:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T, \tag{2.1}$$

where T is the last step. We can call these tasks episodic tasks. It is important to distinguish between tasks that are episodic and have a terminal state and tasks that have a time limit. Episodic tasks with no time limit end up with T, termination time, as a random variable that depends only on the terminal states. Time-limited tasks are episodic, but we have a terminal state that is the time limit, and in an episode, the upper bound of T is defined by the time limit.

However, there are cases where a task is not supposed to finish in a time limit, such as an agent that must learn how to walk without a time limitation. So we can call these tasks continuous tasks because our final step becomes $T = \infty$. Since in this type of task we have $T = \infty$, we cannot keep the return G_t as previously defined, we need to add a discount factor, so that the return to be maximized is not infinite, then we have the discounted return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.2}$$

where γ is a parameter called the discount factor and must be set between 0 and 1. Although the discount factor is unnecessary in episodic tasks, we can also add this discount factor with a value less than one, making the agent prefer short-term rewards.

The policy represented as π describes how the agent defines its actions, mapping the probability of performing an action a from a state s. We can classify policies into deterministic or stochastic.

The value function is another component in an RL agent. This function estimates the agent's rewards starting from a state s and following a policy π. We can formally describe it as:

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right]. \tag{2.3}$$

The action-value function or Q-value can be defined as the expected return from being in state s and performing an action a and follow a policy π. We can formally describe it as:

$$q_\pi(s,a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right]. \tag{2.4}$$

The two value functions presented previously have the objective of evaluating a policy. So, we can compare two policies and define if we have a policy π better or equal to a policy π' based on the expected return.

We can consider at least one optimal policy, and when we have more than one, they share the same optimal state-value function and optimal action-value function. Moreover, this optimal value function $v^*(s)$ is defined as:

$$v^*(s) = \max_\pi (v_\pi(s)). \tag{2.5}$$

And the optimal action-value function is defined as:

$$q^*(s,a) = \max_\pi (q_\pi(s,a)). \tag{2.6}$$

We can say that the basis of RL is the Bellman equation [Sutton & Barto, 2018, Chapter 3], which helps us to find the optimal policies and their respective value functions. The Bellman equation for v_π is defined as:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')], \text{for all} s \in \mathcal{S}, \tag{2.7}$$

where a is the action, s the current state, s' the next state and r the reward.

Knowing how the optimal value functions are defined we can define a relationship between the optimal state-value function and the optimal action-value function:

$$v^*(s) = \max_a (q^*(s,a)). \tag{2.8}$$

So with the Bellman equation, we can recursively define $v(s)$ and $q(s,a)$ based on the current and future state and action. As shown in the following equations:

$$v^*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v^*(s')] \tag{2.9}$$

$$q^*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q^*(s',a')] \tag{2.10}$$

2.2 DYNAMIC PROGRAMMING

Dynamic Programming (DP) refers to the technique for solving a set of complex problems divided into sub-problems based on Bellman's optimality [Sutton & Barto, 2018, Chapter 3]. DP aims to find the optimal policy whenever there is a perfect modeling of an environment such as an MDP. Although in RL , we often do not have a perfect modeling of the environment, DP provides an optimal solution that allows us to evaluate other methods and concepts. We can mention here two powerful algorithms for finding optimal policies: Policy iteration and Value iteration.

2.2.1 Policy iteration

Policy iteration consists in searching for an optimal policy using two processes: evaluating the current policy by obtaining a value function (policy evaluation) and then improving this policy from the obtained value function (policy improvement). The algorithm works with the following steps:

1. We initialize with a random policy;
2. Policy evaluation: we obtain the value function for the current policy by running it on the environment multiple times and computing the obtained rewards accumulated from each state;
3. Policy improvement: we update the policy to take the best actions according to the current value function;
4. If the value function is optimal (stopped changing), we terminate the algorithm, otherwise we repeat from step 2.

2.2.2 Value iteration

The value iteration algorithm also aims to find the optimal policy faster than the policy iteration algorithm because it uses a single step of policy evaluation. The value iteration uses the Bellman optimality equation, thus iteratively calculating the optimal value function and then obtaining the optimal policy. The value iteration algorithm works as follows:

1. In the value iteration method, we initialize the value-state function with a random value for all states except the terminal states, which must be zero.

2. Then we compute the value-state function for all states by updating it according to Equation 2.10.

3. We repeat these steps until the change in the state-value function is small.

4. Then, from this value function we estimate the optimal policy.

2.3 TIME LIMITS

As mentioned in Section 2.1, relative to time, we can classify tasks in RL into two types episodic and continuous tasks. The concept of episodic tasks refers to tasks that are naturally limited by time or whether they have terminal states, so the agent has a specific time to solve that problem. In continuous tasks, on the other hand, the agent does not have a specific time limit to solve the problem. However, in many cases, it is important to limit the agent's training to diversify the agent's experiences.

The paper by Pardo *et al.* [2018] makes a study of the implications of the time limit on learning agents in RL, both in episodic and continuous tasks. We describe this analysis in this chapter.

Primarily, when we analyze a standard agent as defined in most algorithms, the value function is defined as:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(s_{t+1})|S_t = s], \end{aligned} \tag{2.11}$$

where, for one-step temporal-difference (TD), we have that the expected return is:

$$v_\pi(s) = \begin{cases} r & \text{at all termination (including time limits)} \\ r + \gamma \hat{v}_\pi(s') & \text{otherwise} \end{cases} \tag{2.12}$$

The solution proposed by Pardo *et al.* [2018] for episodic tasks is to add the notion of time for the agent. In addition to the observation space of the environment, add the notion of the time remaining to perform the task. Moreover, thus redefine the value function as:

$$
\begin{aligned}
v_\pi(s, T-t) &= \mathbb{E}_\pi[G_{t:T}|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1:T}|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(s_{t+1}, T-t-1)|S_t = s],
\end{aligned}
\tag{2.13}
$$

where for one-step temporal-difference (TD), we have that the expected return is:

$$
v_\pi(s) = \begin{cases} r & \text{at all termination (including time limits)} \\ r + \gamma \hat{v}_\pi(s', T-t-1) & \text{otherwise} \end{cases}
\tag{2.14}
$$

For continuous tasks, Pardo *et al.* [2018] also proposes a solution based on the default agent not bootstrapping when in training is artificially limited by time and also defines a different way of writing the value function to solve this problem:

$$
v_\pi(s) = \mathbb{E}_\pi[G_{t:T} + \gamma^{t-T} v_\pi(s_t)|S_t = s],
\tag{2.15}
$$

where for one-step temporal-difference (TD), we have that the expected return is:

$$
v_\pi(s) = \begin{cases} r & \text{at environmental terminations} \\ r + \gamma \hat{v}_\pi(s', T-t-1) & \text{otherwise (including time limits)} \end{cases}
\tag{2.16}
$$

2.4 Q-LEARNING

Q-learning is a classic RL Watkins & Dayan [1992] algorithm. It is considered off-policy, model-free, and its optimization is done based on values. Its algorithm can be viewed at Algorithm 1. The function Q(s,a) is calculated as follows:

$$
\underbrace{Q_{t+1}(s_t, a_t)}_{\text{New Q-Value}} = Q_t(s_t, a_t) + \underbrace{\alpha}_{\text{Learning rate}} [\underbrace{r_{t+1}}_{\text{Reward}} + \underbrace{\gamma}_{\text{Discount factor}} \overbrace{\max_a Q_t(s_{t+1}, a)}^{\substack{\text{Maximum predicted reward, given} \\ \text{new state and all possible actions}}} - Q_t(s_t, a_t)]
$$

where α is the learning rate and γ is the discount factor. Q-learning is an RL method that has strong convergence evidence [Jaakkola *et al.*, 1994].

Algorithm 1: Q-learning

Initialize $Q(s,a)$ arbitrarily;
for $episode \leftarrow 0$ **to** $episode_{max}$ **do**
    Initialize s;
    for $step \leftarrow 0$ **to** $step_{max}$ **do**
        Choose a from s using policy derived from Q (e.g, $\varepsilon - greedy$);
        Take action a, observe r, s';
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$;
        $s \leftarrow s'$;
        if s *is terminal* **then**
            break;

2.5 STATE–ACTION–REWARD–STATE–ACTION - SARSA

State-Action-Reward-State-Action (SARSA) is also a classic RL algorithm proposed by [Rummery & Niranjan, 1994]. It is a value-based algorithm, these algorithms evaluate the state-action pair by learning one of its value functions and using these evaluations to select actions [Graesser & Keng, 2019]. It is an on-policy algorithm, that is, it evaluates the policy being used to make decisions. Different from Q-learning, it does not use maximization of actions. The algorithm can be visualized in Algorithm 2. The function $Q(s,a)$ in SARSA is calculated as follows:

$$Q_{t+1}(s_t,a_t) = Q_t(s,a) + \alpha[r_{t+1} + \gamma Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)]. \tag{2.18}$$

Algorithm 2: Sarsa

Initialize $Q(s,a)$ arbitrarily;
for $episode \leftarrow 0$ **to** $episode_{max}$ **do**
    Initialize s;
    Choose a from s using policy derived from Q (e.g, $\varepsilon - greedy$);
    for $step \leftarrow 0$ **to** $step_{max}$ **do**
        Take action a, observe r, s';
        Choose a' from s' using policy derived from Q (e.g, $\varepsilon - greedy$);
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$;
        $s \leftarrow s'$;
        $a \leftarrow a'$;
        if s *is terminal* **then**
            break;

2.6 DEEP DETERMINISTIC POLICY GRADIENT (DDPG)

The Deep Deterministic Policy Gradient (DDPG) algorithm was proposed by Lillicrap *et al.* [2015]. It is an off-policy algorithm and uses the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy. It is model-free, and is commonly used in continuous action problems. The algorithm can be visualized in Algorithm 3.

Algorithm 3: DDPG algorithm. Source: Lillicrap *et al.* [2015]

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights$\theta^{Q\prime} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for *episode=1, M* **do**
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state s_1
    for *t=1,T* **do**
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
        Execute action a_t and observe reward r_t and observe new state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in R
        Sample a random minibatch of $\mathcal{N}$ transitions (s_i, a_i, r_i, s_{i+1}) from R
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
        Update the actor policy using the sampled policy gradient:
        $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
        Update the target networks:
        $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
        $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$

In this Chapter, some basic definitions of reinforcement learning were presented to perform the analyses that will be necessary for the following chapters, as well as emphasize the difference between continuous and episodic tasks. It also presented some classic algorithms in the RL area, such as Dynamic Programming, Q-learning, and SARSA, that will be used in an experiment to validate the concept presented by Pardo *et al.* [2018], and these concepts were also presented here. Finally, the DDPG was explained because it was with this algorithm that we performed our final experiments based on the choice of the authors Martins *et al.* [2021a] that used to solve problems of the same environment.

3

METHODOLOGY

This chapter describes the experiments that were performed in this work using the knowledge presented above as a background. The algorithms and hyperparameters for experiments and tests performed are presented in detail as well as the respective justification for the choices made. These experiments have as a general goal of applying the methods developed by Pardo *et al.* [2018], specifically related to the time limit and variations of this method in the context of robot soccer to create different types of agents and conduct a study on their performance.

3.1 GRIDWORLD PROBLEM

In the present work, we conducted a study applying the time-aware agent introduced by Pardo *et al.* [2018] applied to robot soccer. Initially, we applied the concept to a deterministic task in which we had a perfect modeling of the environment, the gridWorld [Knyszewski, 2018]. In the experiment, we defined a limit of steps for the agent to find the gridWorld solution and compare the results of the standard agent and the agent with the notion of time with the optimal policy solution.

To provide a context, the gridWorld environment that we use has a dimension 10×10, an action space of size 5, as the agent has the following options for actions: down, right, left, up, and stay. The rewards are given as follows: the agent can fall into a hole and receive a reward of -20, move around and get a -1 reward every step, stay at the same position and get a 0 reward, or arrive at the treasure and get a +50 reward, as shown in equation 3.1. The terminal states of this environment are when the agent falls into the hole, reaches the treasure, or exceeds the limit of 5 steps to solve the problem.

$$R = \begin{cases} 50 & \text{if the state position equals to treasure position} \\ -20 & \text{if the state position equals to hole position} \\ 0 & \text{if the action equals to stay} \\ -1 & \text{otherwise} \end{cases} \tag{3.1}$$

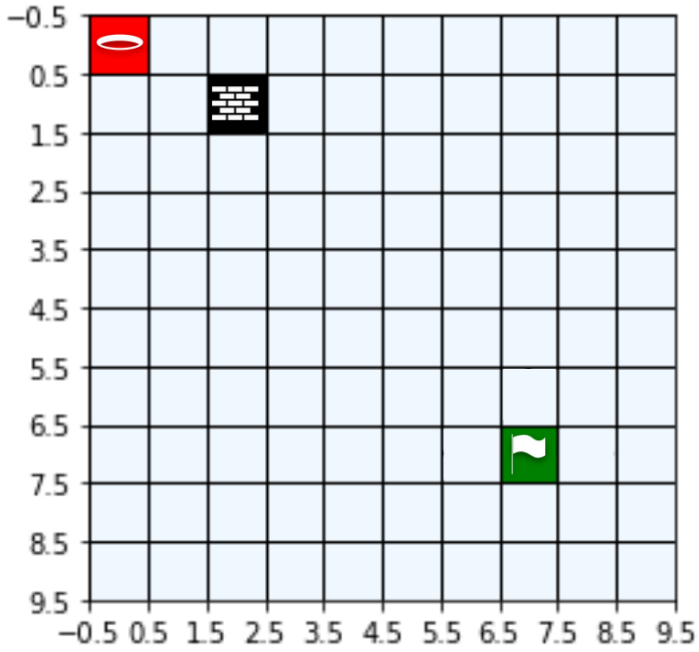


Figure 4: GridWorld environment used in the present work. The red block, position (0,0), represents a hole. The black block, position (1,2), represents a wall. The wall acts as an obstacle for the agent and offers no rewards. The green block, position (7,7), represents the treasure.

To find the optimal policy solution we employ the value iteration algorithm, Section 2.2 , of dynamic programming, for which we model the probability transition matrix of the problem. And for the standard agent and the time-aware agent we used the Q-learning, Section 2.4 and SARSA, Section 2.5, algorithms [Sutton & Barto, 2018], with the parameters shown in Table 1. The choice of parameters was made based on the parameters used by Pardo *et al.* [2018] in the Two-Goal-Gridworld Problem experiment.

Parameter	Value
ε-greedy	0.1
Learning Rate α	0.5
Discount Rate γ	0.99
Steps	25000

Table 1: Parameters for the Q-learning and SARSA algorithms.

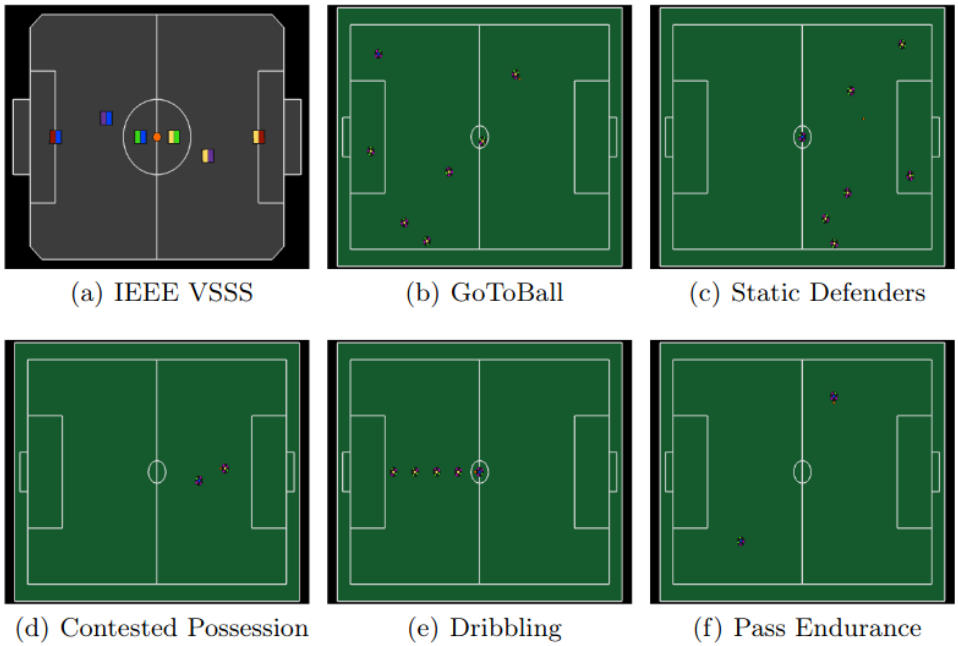
To compare the different algorithms combined with the different agent types, we use the Mean Squared Error (MSE) as a metric, which is defined below:

$$MSE = \frac{1}{n}\sum_{s=1}^{n} v^*(s) - v_\pi(s)^2 \tag{3.2}$$

where $v^*(s)$ represents the optimal value function obtained with the dynamic programming algorithm, $v_\pi(s)$ the value function of the algorithm we are evaluating, s the state we are evaluating and n the number of states.

3.2 SIMULATED ENVIRONMENT FOR ROBOT SOCCER

The simulated environment used in this work was the RSoccer. It is an open-source framework based on OpenAI Gym [Brockman *et al.*, 2016] developed by Martins *et al.* [2021b] for creating RL environments in robot soccer, and that can be divided into three modules: simulator, environment, and render. The simulation module is responsible for providing to the environment the physical simulation of the real environment. The environment module is an intermediate module connected to both the agent and the simulator and is thus responsible for receiving the agent's actions and returning to the agent the observations and rewards. Lastly, the render module is responsible for creating the visualization of the environment. One advantage of the RSoccer is its simulation module, rSim, which can make a higher number of steps per second than other simulators available for robot soccer, bringing the possibility to perform training without the use of graphical rendering and faster than others.



(a) IEEE VSSS (b) GoToBall (c) Static Defenders

(d) Contested Possession (e) Dribbling (f) Pass Endurance

Figure 5: Environments proposed in the RSoccer. Source: [Martins *et al.*, 2021b]

3.2.1 SSL-Go To Ball And Shoot

Among the available environments in the framework, we chose to use in this project, the GoToBallAndShoot, an environment of the Small Size League category. In this environment, we have a robot and a ball, and the task that the robot must develop is to go to the ball and then

kick it to score a goal. As in the Small Size League (SSL) hardware challenge tasks, in this environment time limits are set for the robot to solve the problem, as in soccer matches.

As the observation we have:

- Ball position (x, y) - real values;
- Ball speed (v_x, v_y) - real values;
- Robot position $(x, y, sin(\theta), cos(\theta)))$ - real values;
- Robot speed (v_x, v_y, v_θ) - real values;
- Infrared signal - boolean value;

where (x, y) is the position of the robot in a Cartesian frame, and is the angle of the robot [Kim *et al.*, 2004].

The actions in this environment are continuous and defined by a 5-dimensional command vector, including:

- Robot desired speed (v_x, v_y, v_θ)
- Robot's kick activation
- Robot's dribbler activation

The reward works as follows: if the robot scores one goal, it gets a +5 reward. However, reward shaping is also added to avoid using only sparse rewards that would make the problem harder, so we have the ball_grad and the ball_dist rewards. The ball_grad reward is the variation of the Euclidean distance between the goal position and the last ball position. Therefore, at each step that the ball moves towards the goal, the agent receives a positive reward. If it stays still, it receives no reward, and if it moves away from the goal, a negative reward is given. The ball_dist is the variation of the Euclidean distance between the position of the robot and the position of the ball. Thus if the distance between the ball and the robot decreases, the robot receives a positive reward. If it stays in at the same distance, the reward is zero, and if the distance increases, it receives a negative reward.

It should be noted that in the above terminal states, there is a reward only when the robot scores a goal. According to the category's rules, the robot cannot enter the goalkeeper's area. In all these cases, the done signal is activated, and the agent does not employ bootstrap.

In this environment, we used Deep Deterministic Policy Gradient (DDPG) [Lillicrap *et al.*, 2015] as the agent as it had already been tested in this environment by Bassani *et al.* [2020] and had shown good performance.

We performed tests with two types of agents in this environment: a time-aware agent (TA) and a standard agent (ST) with no information about the time remaining in the environment.

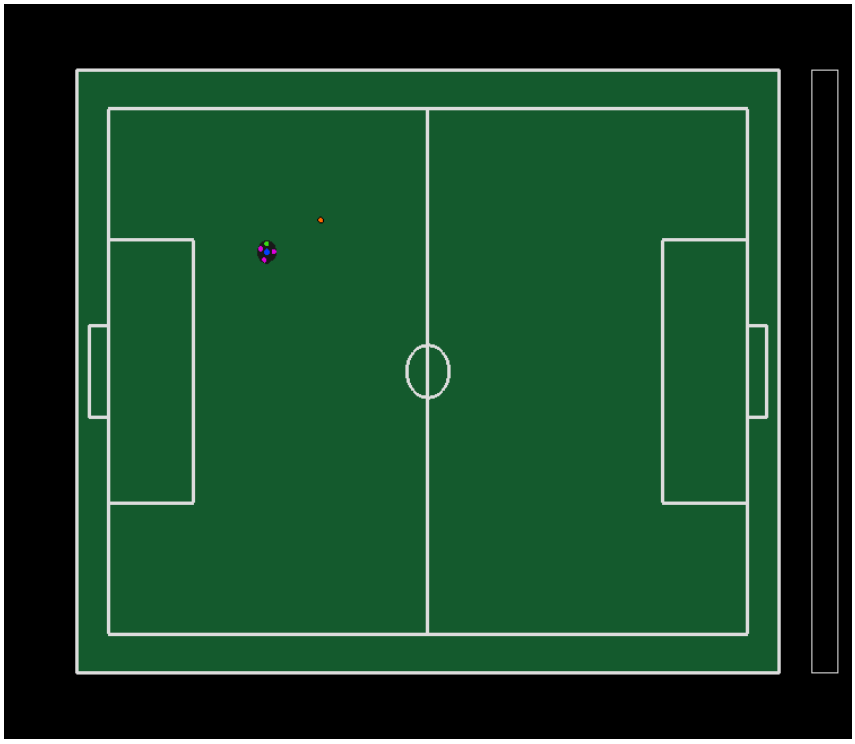


Figure 6: Example of an episode of the SSL-GoToBallAndShoot environment, where the ball and robot positions are randomly initialized.

We modified some parameters in the environment: the time limit, reward shaping, the area where the ball or the robot is initialized, and the addition of noise in the observation angle. In the agent, we modified the discount factor γ and the noise sigma, which is a parameter that is part of the OU process of DDPG exploration. All this is to test different situations and perform a complete analysis on the implications of using time observation in the agent.

4

RESULTS

In this chapter, we present the results of the experiments performed and analyze these results. This chapter is divided into two sections. The first section portrays the analysis done on the grid world environment using classical DP and RL algorithms. The second section shows the results and analysis of the experiments performed on the simulated robot soccer environment.

4.1 TEST ANALYSIS IN THE DETERMINISTIC ENVIRONMENT

As we described in Section 3, we performed the tests with GridWorld because it is a deterministic environment allowing an initial analysis of how the notion of time can improve the results in an episodic task. Figure 7 shows the mean square error between the ground truth, which is the value function obtained using dynamic programming (Section 2.2), and the value function obtained with other Q-learning (Section 2.4) and SARSA (Section 2.5) algorithms. The agents that have the notion of time manage to have an error value close to 0, while the default agents keep a high error value, it can be explained by the fact that the default agent cannot distinguish from which states it is not possible reach the treasure.

Figure 8 represents the state value function for different algorithms and with the TA agent, and so we verify that the notion of time allows from states that are at a distance greater than T steps the agent expects to gain a reward of 0 that is equivalent to staying in place because if he moves, he would not reach the treasure and would end up with a negative reward. This figure also allows us to visualize that we add a third dimension to the state space when we add the notion of time, since with the standard agent, we had two dimensions, the position in x and y. Therefore, what happens is that the standard agent considers all states of the value function equally for different time steps.

4.2 ANALYSIS OF EXPERIMENTS IN ROBOT SOCCER SIMULATION

Before performing the analysis on the time limits, we analyze the choice of the γ parameter and the choice of the noise parameter. Figures 9 and 10 refer to the experiments done for different values of the discount factor, γ. This parameter was chosen for the analysis because

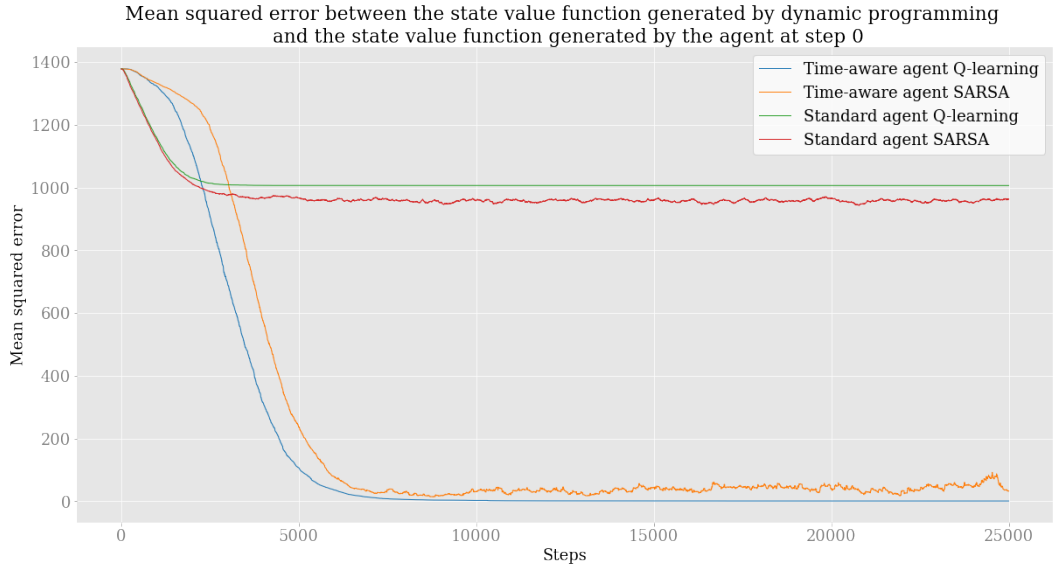


Figure 7: Comparison between the different agents with respect to the ground truth obtained with dynamic programming.

we thought about the hypothesis that it correlates with the way the TA agent behaves, since if we have a small gamma value, the agent will prioritize only the immediate rewards, so it will not take into account the time it has to perform the task, and possibly the time information would not make any difference. During the tests, we verified that if we have a small gamma, it happens that the TA agent and the standard agent have the same behavior. This allows us to define a value to use during our experiments. We can see that the number of goals is higher for agents with the gamma set to 0.99. However, when we look at the metric of time left after kicking the ball, Figure 10, we can see that agents with a gamma of 0.999 finish the task with little time left, probably because they do not prioritize short-term rewards as much as the others.

After an analysis of the behavior of the agents, we noticed that the agents always kick as soon as it reaches the ball, at a great distance from the goal, regardless of the time limit. Then we hypothesized that with perfect observations and actions the agent had no difficulties performing the task. This does not reflect the reality of the SSL game, as we have noisy position estimation and control. Thus, we performed tests to add a Gaussian noise to the robot angle observation to hamper but not prevent learning.

Figures 11 and 12 show the results of the experiments performed to observe the differences from the addition of zero mean noise to the robot's angle observed by the agent. It can be seen that with the noise level with standard deviation 5, the number of goals decreases concerning the other agents. The curve that is farthest from the others is the TA agent that has decreased the number of goals more than the other agents but continues to learn. As for the time left when the kick was executed, the agents with higher noise present kick with less time left, which can be explained by the difficulty for the agent to execute the goal task, requiring the agent to get closer to the goal before kicking.

From this test, we observed that the presence of noise is important for modeling a more

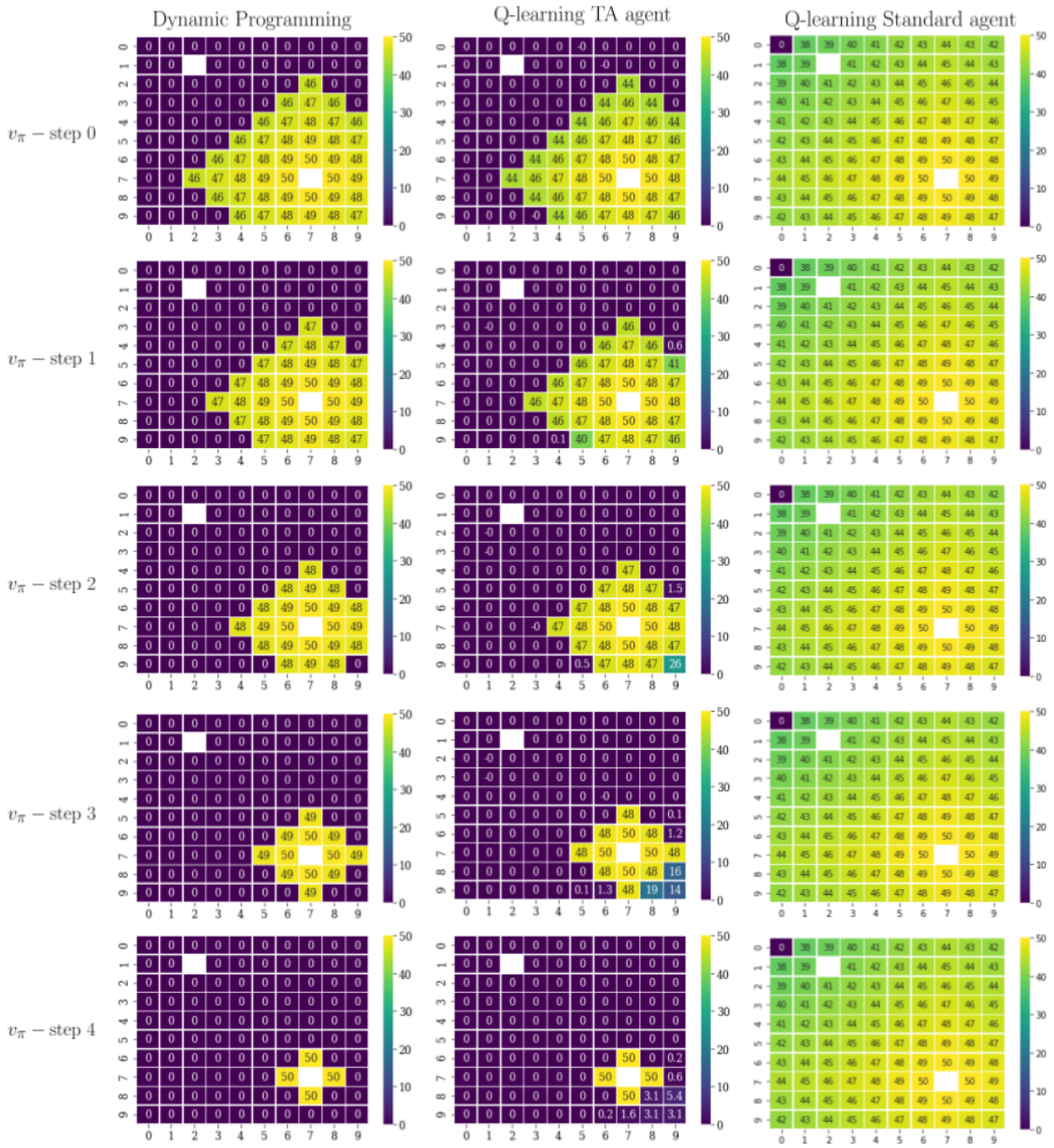


Figure 8: Comparison of the state value function for the different algorithms tested for a TA agent and standard agent. Each line represents one step of the agent within the map to perform five steps at most. The line at step 0 represents the agent's expected to return when it has not yet taken any steps.

realistic situation because in a real SSL game we do not have perfect precision. Thus, the agents try to get around this inaccuracy by getting closer to the goal to perform the kick. We also observed that regardless of the type of agent, they had 15 seconds to solve the problem and they could score in an average time, so that time was not a limiting factor for its performance.

After testing different values for the parameters: γ and noise of the robot angle observation. We chose the parameters to make the tests based on the analysis made earlier. Therefore, in Figure 13, we present a test comparing the standard agent and a TA agent. The difference

Figure 9: The number of goals for agents with and without time awareness and with different discount factor values.

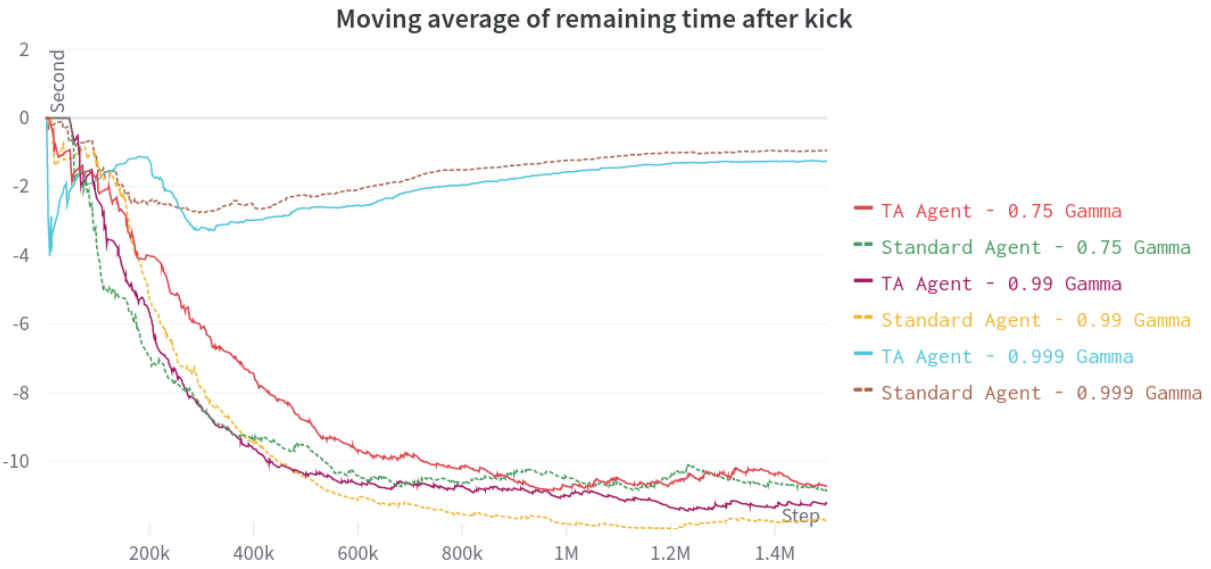


Figure 10: The time remained after the robot kicked for agents with and without time awareness and different discount factor values.

between TA agent and standard agent is that TA agent has one more observation that is the number of steps remaining to finish the episode. . We used the discount factor of 0.99 and the noise in the observation of the robot angle of mean 0 and standard deviation of 5.

Remarkably, the TA agent displays a higher goal average and a higher reward than the standard agent during learning. We can observe by looking at the videos generated during training that, when the time left allows, the agent with time awareness takes the ball closer to the goal than the agent without time awareness. This can also be verified because the TA agent has less time left after the kick in most cases from step 3M.

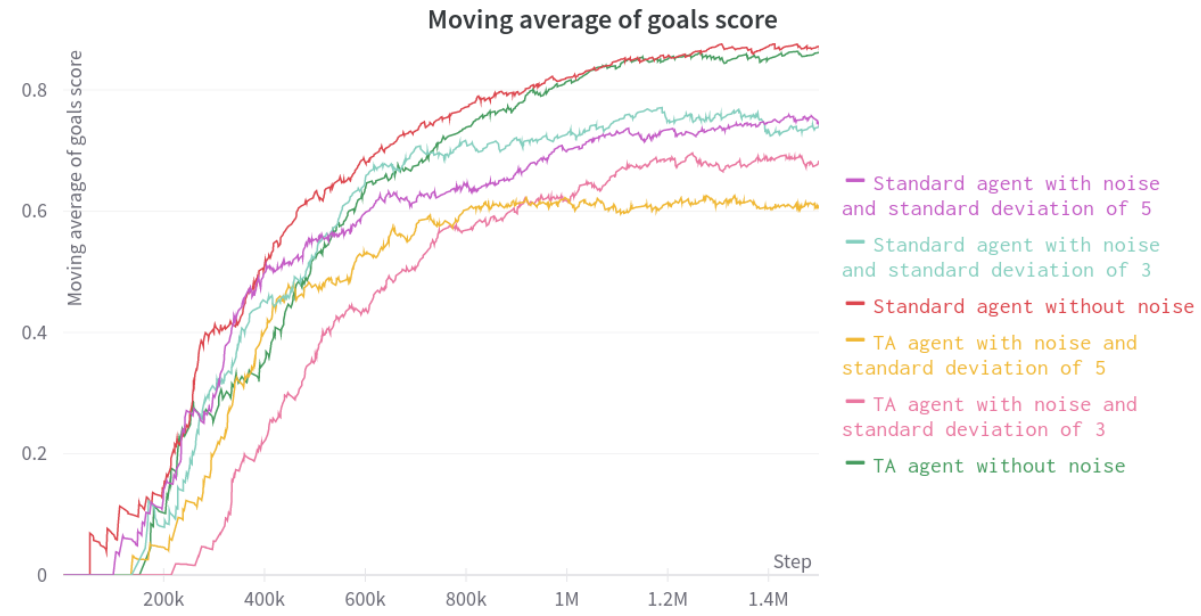


Figure 11: Comparison between the number of goals for agents with and without time awareness and different zero mean noises added to observation of the robot angle.

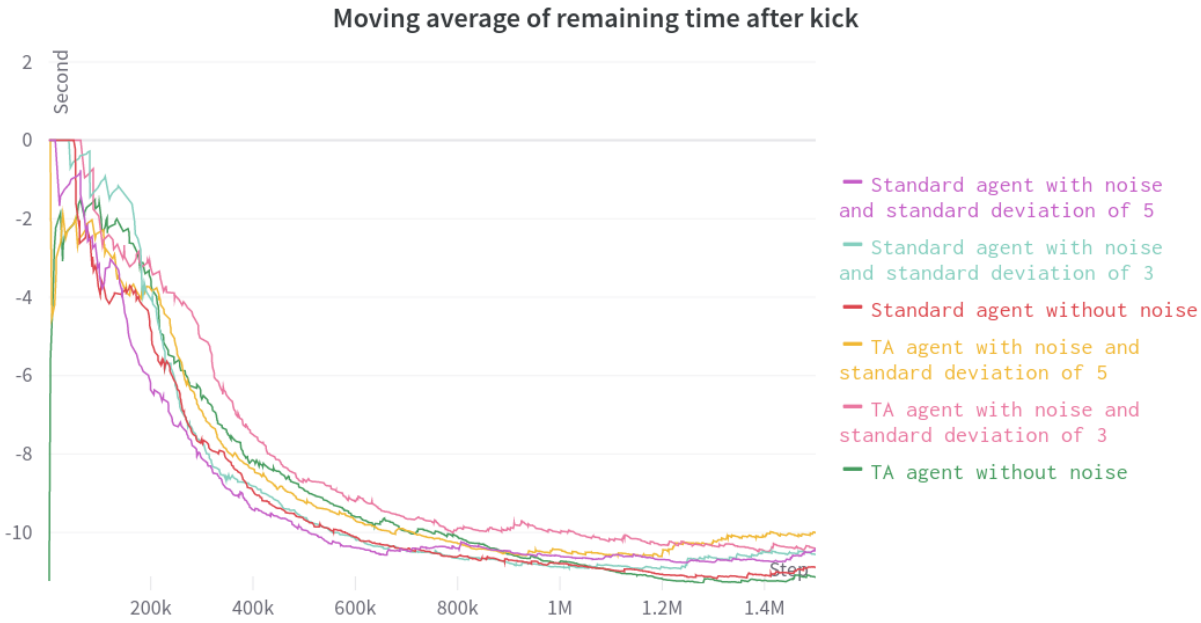


Figure 12: Comparison between the time left after the kick for agents with and without time awareness and different levels of zero mean noise added to the robot angle observation.

In Figure 14, we compare the agents from the points in which both models were trained for 5 millions of steps. It can be seen that the TA agent presents a higher performance in the number of goals. In the graph that shows the distance between the ball's initial position and the position in which the ball was kicked, it is clear that, on average, the TA agent takes the ball closer to the goal than the other agent.

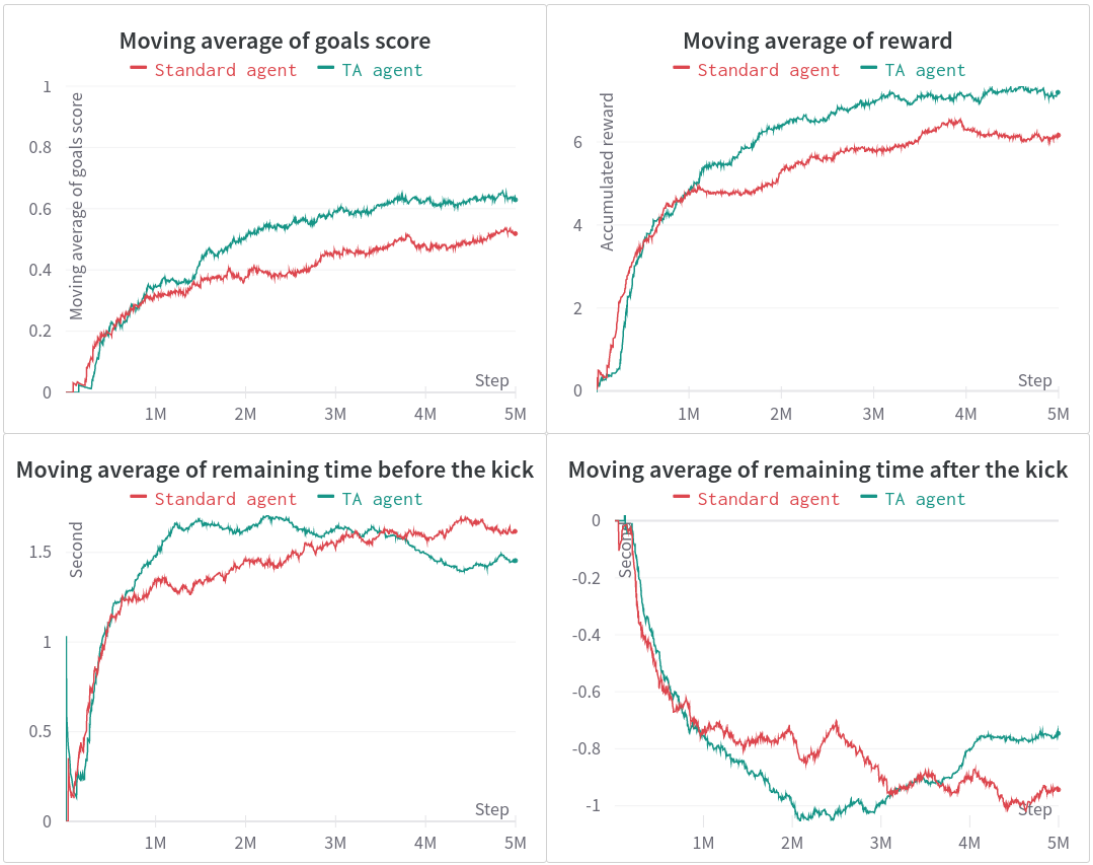


Figure 13: Comparison between the agents with and without time awareness for a 5 second time limit with the discount factor of 0.99.

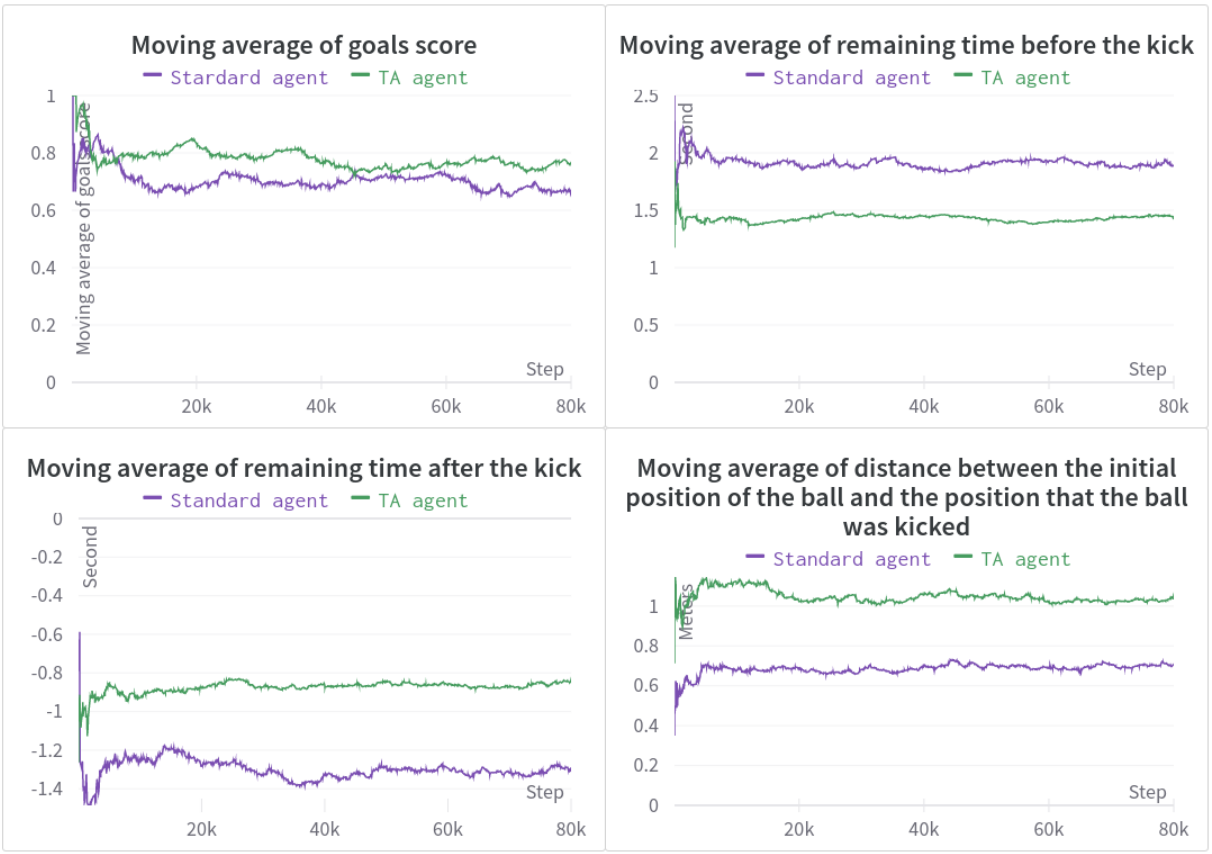


Figure 14: Comparison between trained agents with and without a time awareness for a 5 second time limit.

4.2.1 Tests with static scenes

We also performed tests with static scenes, in these tests, we took points in which both agents were already trained and performed a test for 10000 steps and computed an average of the goal score and time measures obtained in each scene. The results are shown in Tables 2, 3, 4, 5, 6 and 7. These scenes can bee seen in the Figures 15 , 16 and 17. The training videos for these scenes are available in the repository on GitHub [1].

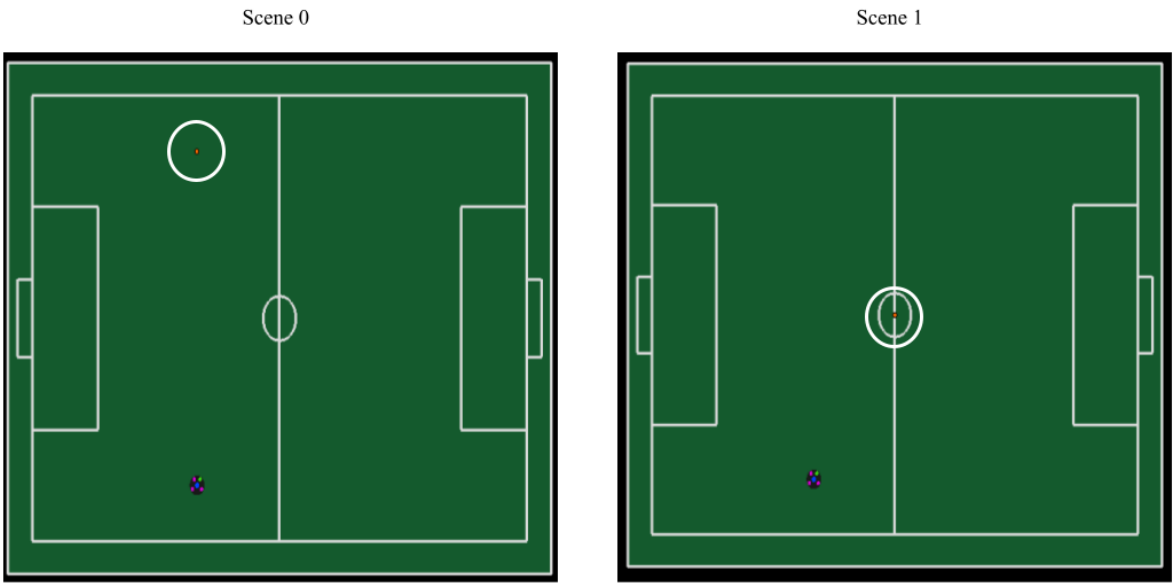


Figure 15: Static scenes set to perform the final tests. On the left is scene 0 and on the right is scene 1. The white circle shows the position of the ball in the field.

We also analyze the performance of the agents in these scenes. In Table 2 we present the performance of the agents when we initialize the positions of scene 0. Both for the scene with a limitation of 200 steps and the scene with a limitation of 400 steps, the default agent has a better performance in the number of goals, even having a longer remaining time than the TA agent. A different behavior we see in this scene is when we have the step limit of 400, the standard agent takes the ball closer to the goal than the agent TA.

In Table 3, the time-aware agent's performance is better when we have a step limit of 200 and a better performance of the standard agent when we have a time limit of 400 steps.

The TA agent could not execute the task because it did not score goals in Table 4, when we have a time limit of 200 steps. Observing the videos generated during this test in this scene, the ball, and the robot are far apart, and that the TA agent takes a long time to get to the ball and when it executes the kicking action, and in some cases the time was short and in other cases he did not have a good aim to kick from a great distance. Also, observing Table 4 it is remarkable

[1] https://github.com/JulianaDamurie/timeLimitsRobotSoccer

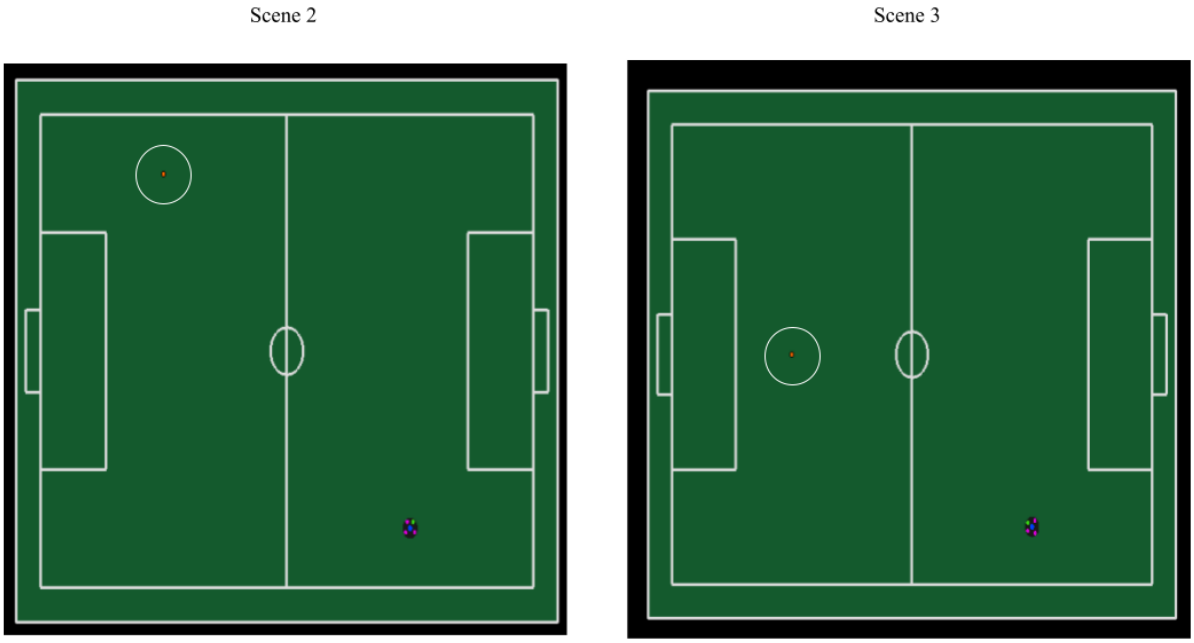


Figure 16: Static scenes set to perform the final tests. On the left is scene 2 and on the right is scene 3. The white circle shows the position of the ball in the field.

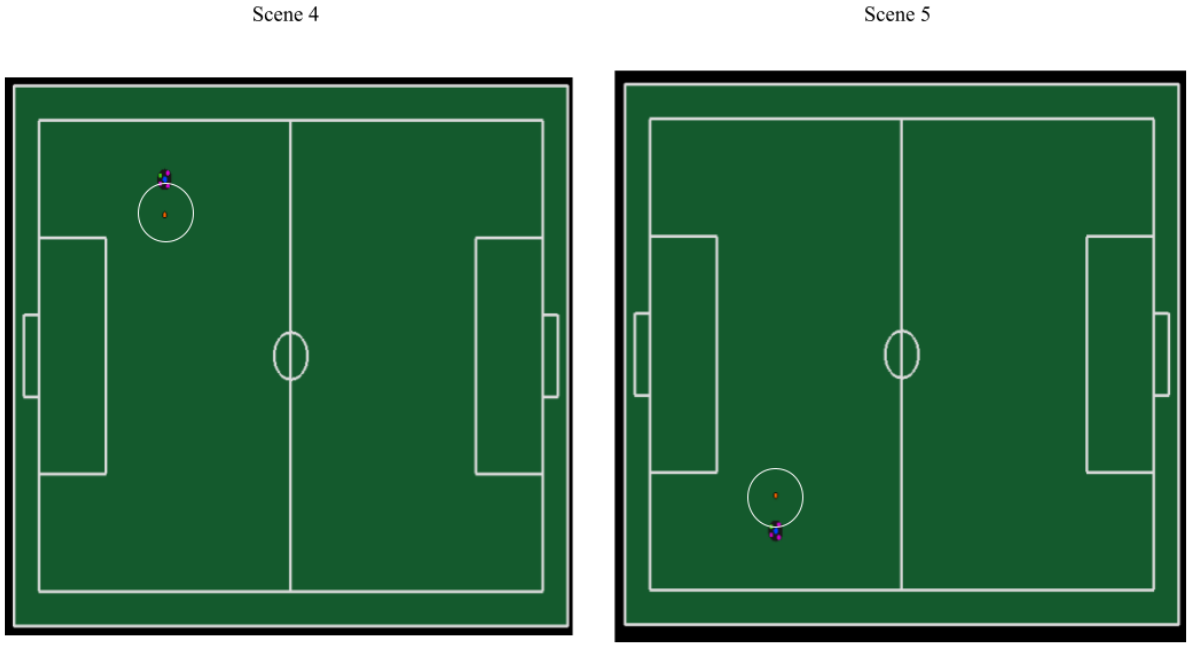


Figure 17: Static scenes set to perform the final tests. On the left is scene 4 and on the right is scene 5. The white circle shows the position of the ball in the field.

that when we increase the limit of steps, the TA agent starts to score goals and take the ball closer to the goal. A hypothesis to be raised is that possibly the TA agent did not develop a skill to shoot from a long distance as well as the standard agent, as the TA agent always tried to get closer to the goal.

Steps Limit	Agent	Goal	Remaining Time before the kick	Remaining time after the kick	Distance between the initial position of the ball and the position that the ball was kicked
200	Standard	0.6805 ± 0.4663	1.3989 ± 0.6990 s	0.8959 ± 0.5028 s	1.0410 ± 0.5180 m
200	TA	0.3223 ± 0.4673	0.1508 ± 0.3676 s	0.3099 ± 0.1048 s	1.1499 ± 0.3386 m
400	Standard	0.7431 ± 0.4369	5.5216 ± 2.5531 s	5.0195 ± 2.3251 s	1.0387 ± 0.5224 m
400	TA	0.5788 ± 0.4937	5.2191 ± 0.5916 s	4.5218 ± 0.5021 s	0.7522 ± 0.3732 m

Table 2: Comparison between the time-aware agent and the standard agent for static scene 0.

Steps Limit	Agent	Goal	Remaining Time before the kick	Remaining time after the kick	Distance between the initial position of the ball and the position that the ball was kicked
200	Standard	0.8342 ± 0.3719	1.6376 ± 0.6722 s	1.1524 ± 0.6214 s	0.5513 ± 0.3026 m
200	TA	0.8905 ± 0.3122	1.7316 ± 0.1119 s	1.3027 ± 0.0961 s	0.8879 ± 0.3386 m
400	Standard	0.8742 ± 0.3316	6.4746 ± 1.3213 s	5.9877 ± 1.2397 s	0.5551 ± 0.3061 m
400	TA	0.8056 ± 0.3957	4.8411 ± 0.2230 s	4.4045 ± 0.1993 s	0.8985 ± 0.1553 m

Table 3: Comparison between the time-aware agent and the standard agent for static scene 1.

Steps Limit	Agent	Goal	Remaining Time before the kick	Remaining time after the kick	Distance between the initial position of the ball and the position that the ball was kicked
200	Standard	0.5451 ± 0.4979	1.5239 ± 0.3200 s	0.6525 ± 0.2322 s	0.4189 ± 0.4394 m
200	TA	0.0000 ± 0.0000	0.3750 ± 0.4896 s	0.0288 ± 0.1107 s	0.0856 ± 0.1363 m
400	Standard	0.6704 ± 0.4700	6.5284 ± 0.3719 s	5.6527 ± 0.2930 s	0.4062 ± 0.4243 m
400	TA	0.5733 ± 0.4946	4.6650 ± 1.6864 s	3.9888 ± 1.4842 s	1.009 ± 0.5423 m

Table 4: Comparison between the time-aware agent and the standard agent for static scene 2.

In Table 5, we observed a case in which the time limit is determinant for a better performance of the time-aware agent. When we have a time limit of 200 steps, it performs better than the standard agent. However, with the step limit of 400, its performance degrades. Moreover, observing the videos generated, we noticed that that the time-aware agent, as the TA agent was trained for 200 steps and one of the tests was performed with 400 steps, this decline in performance is explained because the TA agent was not trained with variations of the number of steps, and the number of steps for it is important information. So it would be valid in a future study to reevaluate these agents but training the TA agent with variations of the time limit.

In the case of scene 4, Table 6, where the robot already initializes near the ball. For both time limits, we have a better performance of the TA agent. Moreover, we can see that the standard agent does not get a good performance when we have a limit of 200 steps, even carrying much more the ball. When we look at the video, it is identifiable that it ends up taking the ball to

a point so far away that it has no more time to kick.

Steps Limit	Agent	Goal	Remaining Time before the kick	Remaining time after the kick	Distance between the initial position of the ball and the position that the ball was kicked
200	Standard	0.8390 ± 0.3675	1.3304 ± 0.1246 s	0.6210 ± 0.1182 s	0.9541 ± 0.0914 m
200	TA	0.9030 ± 0.2959	1.3619 ± 0.0992 s	0.6410 ± 0.0610 s	0.9180 ± 0.2067 m
400	Standard	0.8963 ± 0.3048	6.3273 ± 0.2069 s	5.6174 ± 0.1913 s	0.9535 ± 0.0190 m
400	TA	0.1441 ± 0.3512	4.6650 ± 1.6864 s	0.7173 ± 1.5909 s	0.2202 ± 0.4935 m

Table 5: Comparison between the time-aware agent and the standard agent for static scene 3

In Table 7, the time-aware agent has better performance. In this scene, the ball and the robot are close. It is important to observe that the time-aware agent carries less the ball, but it finishes the activity with less time left than the standard agent.

Steps Limit	Agent	Goal	Remaining Time before the kick	Remaining time after the kick	Distance between the initial position of the ball and the position that the ball was kicked
200	Standard	0.3949 ± 0.4888	1.1523 ± 0.6515 s	0.5126 ± 0.5669 s	1.5009 ± 0.5322 m
200	TA	0.8162 ± 0.3873	1.8168 ± 0.4093 s	1.1566 ± 0.2838 s	1.2162 ± 0.4207 m
400	Standard	0.7749 ± 0.4176	5.9419 ± 1.3864 s	5.2890 ± 1.2503 s	1.5254 ± 0.5053 m
400	TA	0.7813 ± 0.4133	4.891358 ± 0.3271 s	4.2785 ± 0.2860 s	1.5988 ± 0.2321 m

Table 6: Comparison between the time-aware agent and the standard agent for static scene 4.

Steps Limit	Agent	Goal	Remaining Time before the kick	Remaining time after the kick	Distance between the initial position of the ball and the position that the ball was kicked
200	Standard	0.8428 ± 0.3640	2.2082 ± 0.2854 s	1.6447 ± 0.2107 s	2.1254 ± 0.5473 m
200	TA	0.9873 ± 0.1119	1.5809 ± 0.1233 s	1.0216 ± 0.0933 s	1.8580 ± 0.1577 m
400	Standard	0.8396 ± 0.3669	7.2044 ± 2.5531 s	6.6418 ± 0.2657 s	2.1341 ± 0.5433 m
400	TA	0.9146 ± 0.2794	3.1096 ± 0.5818 s	2.6327 ± 0.5245 s	2.2528 ± 0.4334 m

Table 7: Comparison between the time-aware agent and the standard agent for static scene 5.

In that manner, observing the agents' behavior in the scenes, we see it is remarkable that the advantage of using one agent or another depends a lot on the situation. Nevertheless, it is clear that when time is critical, the time-aware agent can develop better, as we had already seen in the analysis of Figure 14.

5

CONCLUSION

Throughout the study, it became apparent that time-aware and standard agents have specific advantages depending on the scenario and environment in which they are tested, but that there are differences in how they learn and perform a task. We also conclude that having a time limit only will not necessarily imply that a TA agent will perform the task more efficiently or behave differently from the standard agent. This difference depends on several factors, such as the discount factor that will define whether or not the agent will give preference to short-term rewards, and this changes the way the agent observes this time limit. Moreover, The standard agent knows best how to handle situations out of distribution.

Significantly, the time-aware agent performed better when we set a smaller time limit much closer to the time it converged to finish the task, that is, in situations where the time limit is critical to finish that activity.

Given the results obtained in this work, some points can be considered in future studies:

- To further investigate some behaviors in which the TA agent behaved better in training and failed during the static scene;
- Train a TA agent with the number of steps varying during training and then evaluate this behavior;
- Investigate the behavior of these agents with other reinforcement learning algorithms such as SAC and PPO;
- Perform the experiments and evaluate in environments with opponents and multiple agents;
- Perform the experiments and evaluate in a match and not only in a specific task, since it is an environment in which there is a time limit to achieve victory,

REFERENCES

Afzal, A., Katz, D. S., Le Goues, C., & Timperley, C. S. (2021). Simulation for robotics test automation: Developer perspectives. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, 263–274.

Barrett, S., Taylor, M. E., & Stone, P. (2010). Transfer learning for reinforcement learning on a physical robot. In *Ninth International Conference on Autonomous Agents and Multiagent Systems-Adaptive Learning Agents Workshop (AAMAS-ALA)*, 1.

Bassani, H. F., Delgado, R. A., de O. Lima Junior, J. N., Medeiros, H. R., Braga, P. H. M., Machado, M. G., Santos, L. H. C., & Tapp, A. (2020). A framework for studying reinforcement learning and sim-to-real in robot soccer.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., *et al.* (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Campbell, M., Hoane Jr, A. J., & Hsu, F.-h. (2002). Deep blue. *Artificial intelligence*, 134(1-2):57–83.

Graesser, L. & Keng, W. L. (2019). *Foundations of deep reinforcement learning: theory and practice in Python*. Addison-Wesley Professional.

Harada, D. (1997). Reinforcement learning with time. In *AAAI/IAAI*, 577–582.

Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P., & Levine, S. (2021). How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4-5):698–721.

Jaakkola, T., Jordan, M. I., & Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural computation*, 6(6):1185–1201.

Kim, J.-H., Kim, D.-H., Kim, Y.-J., & Seow, K. T. (2004). *Soccer robotics*, volume 11. Springer Science & Business Media.

Knyszewski, F. (2018). Model free reinforcment learning (sarsa/q-learning). `https://github.com/filipkny/MediumRare`.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Martins, F. B., Machado, M. G., Bassani, H. F., Braga, P. B. M., & Barros, E. S. (2021a). Rsoccer ssl and vsss openai gym environments. `https://github.com/robocin/rSoccer`.

Martins, F. B., Machado, M. G., Bassani, H. F., Braga, P. H. M., & Barros, E. N. S. (2021b). rsoccer: A framework for studying reinforcement learning in small and very small size robot soccer. *CoRR*, abs/2106.12895.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Pardo, F., Tavakoli, A., Levdik, V., & Kormushev, P. (2018). Time limits in reinforcement learning.

Ravichandiran, S. (2018). *Hands-on reinforcement learning with Python: master reinforcement and deep reinforcement learning using OpenAI gym and tensorFlow*. Packt Publishing Ltd.

Reinke, C. (2020). Time adaptive reinforcement learning.

Rummery, G. A. & Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. Citeseer.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., *et al.* (2017). Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

Stone, P. & Sutton, R. S. (2001). Scaling reinforcement learning toward robocup soccer. In *Icml*, 1:537–544.

Sutton, R. S. & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., *et al.* (2017). Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*.

Watkins, C. J. & Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Weitzenfeld, A., Biswas, J., Akar, M., & Sukvichai, K. (2015). Robocup small-size league: Past, present and future. In Bianchi, R. A. C., Akin, H. L., Ramamoorthy, S., & Sugiura, K., editors, *RoboCup 2014: Robot World Cup XVIII*, 611–623.