IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 25 May 2022, accepted 23 June 2022, date of publication 7 July 2022, date of current version 15 July 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3189021

RESEARCH ARTICLE

Multiagent Reinforcement Learning for Strategic Decision Making and Control in Robotic Soccer Through Self-Play

BRUNO BRANDÃO[1], TELMA WOERLE DE LIMA[1], ANDERSON SOARES[1], LUCKECIANO MELO[1], AND MARCOS R. O. A. MAXIMO[2]

[1]Deep Learning Brazil, Federal University of Goiás (UFG), Goiânia, Goiás 74690-900, Brazil
[2]Autonomous Computational Systems Laboratory (LAB-SCA), Computer Science Division, Aeronautics Institute of Technology, São José dos Campos, São Paulo 12228-900, Brazil

Corresponding author: Bruno Brandão (brunobrandao1523@gmail.com)

This work was supported in part by the Brazilian Coordination for the Improvement of Higher Education Personnel (CAPES) under Grant 88882.385785/2019-01.

ABSTRACT Reinforcement Learning (RL) has shown promising performance in environments for both robotic control and strategic decision making. However, they are usually treated as separate problems with different objectives. In this work, we propose the use of Reinforcement Learning to solve both control and strategic problems as one, in a multi-agent robotic soccer environment. We use the IEEE Very Small Size Soccer (VSSS) challenge from the Latin American Robotics Competition (LARC) as a study case. In the VSSS, two autonomous teams of wheeled robots compete by pushing the ball around to score goals. To unify both control and strategy problems, our approach gives full control of the actuators' speed to the RL algorithm whilst keeping the broader objective of winning the game. Our method achieves win rates as high as 93% against hand-coded heuristic strategies. In this work we contribute by developing an RL agent that can learn from self-play and generalize against new opponents. Our methodology uses multi-agent Reinforcement Learning with self-play in order to build up the knowledge for complex tasks. We also developed a simulated environment for the robotic soccer game.

INDEX TERMS Decision making, multi-agent, reinforcement learning, self-play, strategy.

I. INTRODUCTION

Robots are primarily used for automation since they can be designed specifically for each task, and in some cases are cheaper, and safer than human workers. As the use of robots become more widespread, the problem of decision-making and control becomes more relevant.

Reinforcement Learning is capable of solving high-level decision-making tasks. Studies prove that it can surpass humans in strategy games such as Chess, GO, ATARI, StarCraft2,[1] and DOTA2[2] [1]–[4]. These games are famous for demanding that players plan ahead and consider different outcomes. In the special cases of StarCraft2 and DOTA2, they even require different levels of decision-making called macro and micro management [3]–[5]. RL is also capable of solving several robotic control tasks. Those include walking on uneven terrain using humanoids and ant-like agents [6], stabilizing drone flights [7], and dexterity manipulation using robotic arms and hands [8]. These control tasks require the algorithm to apply either speed or torque to robot actuators without intermediate systems to accomplish specific behaviors.

To further test the capabilities of Reinforcement Learning, some researchers show that it can also solve multi-agent environments [5], [9]–[11]. In these problems, the algorithm interacts with an environment that contains other learners. These interactions lead to the emergence of both cooperative

The associate editor coordinating the review of this manuscript and approving it for publication was Zhan Bu.

[1]StarCraft 2 is a real time strategy game, developed by Blizzard Entertainment Inc. More information can be found at https://starcraft2.com/en-us/

[2]DOTA2 is a multiplayer battle arena game developed by Valve Corporation. More information can be found at http://blog.dota2.com/?l=english

and competitive behaviors [9], [12]. Some applications of Multi-Agent Reinforcement Learning (MARL) include managing smart grids [13], unmanned aerial vehicles [14], and micro-managing units in strategy games [5].

In this work, we demonstrate that RL can solve both control and strategy problems simultaneously. We make use of a robotic soccer environment to tackle both problems at once. More specifically, we use the IEEE Very Small Size Soccer category from the Latin American Robotics Competition[3] [15]. The robotic soccer is a multi-agent environment where teams of robots–wheeled robots in the case of the VSSS category– attempt to score goals on each other. Winning the game demands accurate robot control to accomplish behaviors such as pursuing the ball and blocking attacks. The environment also demands strategic decision-making to coordinate players, overcome the opponent team, and score goals.

There have been other attempts to solve robotic soccer games with Reinforcement Learning [16]–[18]. However, they pursue either the control problem or the strategy problem [19], [20]. Both problems are often looked at separately because they form a hierarchy. Decisions in strategy can generate either targets or behaviors that will be executed by controllers [19]–[21]. This separation can be useful, as it makes the search space smaller, and thus the optimization process shorter. Nonetheless, it limits the available behaviors agents can perform as it bounds their decisions to a single level of the hierarchy. It is worth remembering that the main objective of robotic soccer is not only to optimize control or find suitable strategies, it is to do both in order to effectively win the game. Thus this separation comes from practicality rather than efficiency.

Approaches that focus on control commonly strive to solve a single task such as passing, or kicking [22], [23]. Meanwhile, strategy-focused approaches often make use of RL for high-level decision making. That includes deciding among a discrete set of actions for robots, choosing a specific strategy, or selecting the best angle to kick [16]–[18]. In this work, we give the artificial neural network full control of the actuators from a single robot, which require continuous actions to determine their speed. In our design of this multi-agent environment, all robots (i.e. agents) of the same team share the same network. In other words, the decision-making policy is the same for all teammates. It reads a single state and acts on a single robot, however, we run it once for each player in a single timestep. This configuration allows us to collect more experiences from a single VSSS match than we would on a single-agent configuration or multiple agents with different policies. Essentially, for every situation, our training algorithm uses experiences from all players on the team to learn. Our approach also saves computational resources by allowing us to store a single policy model that is used on all agents instead of multiple different models. Since our actions are designed as velocities for actuators, the task demands both control proficiency for achieving complex behaviors and strategic decision making for coordinating multiple players to win the game. We use the state of the art Proximal Policy Optimization (PPO) training algorithm due to its higher sample efficiency and performance on continuous environments compared to other on-policy methods [6], [24]. The PPO algorithm is also able to solve similar tasks in control, soccer, and multi-agent environments [9], [22]. We also perform self-play [25] during training and show that this creates an auto-curriculum learning where the difficulty of the adversary increases as learning progresses. It also allows for the resulting model to generalize against previously unseen opponents.

To validate our work we use heuristic strategies and control developed by the Brazilian team Pequi Mecânico[4] [19], [20]. The team is an avid participant of the IEEE Very Small Size Soccer category for over ten years. After training, our model was able to surpass the Pequi Mecânico's best heuristic strategy and achieve a winning rate of 83% against it.

Our main contributions in this paper are:

- An agent that can learn from self-play and generalize against previously unseen opponents;
- A simulated environment of the robotic soccer game;
- An analysis of the learning process to show how self-play is essential for learning in this environment;
- A unified approach to control and strategic decision making.

The remainder of this paper is structured as follows. Section II presents a theoretical background. Section IV discusses related works. Section III describes the VSSS category in more detail. Section V depicts the simulated environment. Section VI contains the problem modeling. Section VII details our methodology, including the training algorithm and evaluation methods. Section VIII displays and discusses the results from experiments. Finally, section IX holds concluding remarks.

II. BACKGROUND

A. MULTIAGENT REINFORCEMENT LEARNING

Reinforcement Learning problems are mathematically formulated as Markov Decision Processes (MDP) [26, pp. 47]. In cases where the agent does not have the complete information about the state of the environment, this formulation can be extended to Partially Observable Markov Decision Processes (POMDP) [27, pp. 12], [26, pp. 197]. A POMDP is represented by the following tuple $F = \langle \mathcal{A}, \mathcal{S}, \mathcal{Z}, \mathcal{R}, P \rangle$. An agent observes the environment through $s \in \mathcal{S}$ and chooses an action $a \in \mathcal{A}$. The action triggers a change in the *true* state of the environment $z \in \mathcal{Z}$ according to a probability function $P(z'|z, a) : \mathcal{Z} \times \mathcal{Z} \times \mathcal{A} \rightarrow [0, 1]$. The new state also implies in a new observation $s' \in \mathcal{S}$. Afterwards, the agent receives a reward r according to the function $\mathcal{R}(z, a)$:

[3]The Latin American Robotics Competition's website is http://www.cbrobotica.org/

[4]Pequi Mecânico is a Brazilian robotics lab from Federal University of Goiás; their publicly available code releases can be found at https://github.com/PEQUI-MEC

$\mathcal{Z} \times \mathcal{A} \rightarrow \mathbb{R}$. Each transition can be represented as a tuple of an observation, an action, a reward, and the next observation (s, a, r, s'). The goal of the RL algorithm is to find the policy π that maximizes the return:

$$R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}, \tag{1}$$

which sums rewards r across time discounted by $\gamma \in [0, 1)$, via interactions between the policy and the environment.

In Multi-Agent Reinforcement Learning, sequential decision-making problems can be formalized as Decentralized Partially Observable Markov Decision Processes (Dec-POMDP). These processes are similar to regular POMDPs, except each agent will have its own observation, action, and reward. This multiplicity means that instead of a single observation, the environment will provide a set of joint observations, likewise for actions and rewards. Dec-POMDPs can be represented by a tuple $G = \langle \mathcal{D}, \mathcal{A}, \mathcal{S}, \mathcal{Z}, \mathcal{R}, P \rangle$ [27]. At each timestep t a set of agents $d \in \mathcal{D} \equiv \{1, \ldots, n\}$ process their each individual observations within their *observation space* $s^d \in \mathcal{S}$. The combination of these observations result in the set of joint observations $\mathbf{s} \in \mathbf{O} \equiv \mathcal{S}^n$. Afterwards, each agent processes observations and chooses actions within their *action space*, $a^d \in \mathcal{A}$, according to its own decision policy $\pi^d(a|s)$. A policy π^d denotes the probability of agent d choosing action a given observation s. These actions combined form unified actions $\mathbf{u} \in \mathbf{U} \equiv \mathcal{A}^n$ which trigger a change in the *true* state of the environment $z \in \mathcal{Z}$ according to the probability function $P(z'|z, \mathbf{u}) : \mathcal{Z} \times \mathcal{Z} \times \mathbf{U} \rightarrow [0, 1]$. The sets of unified actions $\mathbf{u}$ and observations $\mathbf{s}$ are vectors whose elements are individual actions and observations of each agent. After changing the state of the environment, the agents receive rewards separately, though according to the same function $\mathcal{R}(z, \mathbf{u}) : \mathcal{Z} \times \mathbf{U} \rightarrow \mathbb{R}$. The vector with all the agents' rewards is denominated $\mathbf{r}$. Therefore, each interaction, or transition, between the agents and the environment in MARL is denoted by a vector form of the regular RL transition tuple $(\mathbf{s}, \mathbf{u}, \mathbf{r}, \mathbf{s}')$.

B. PROXIMAL POLICY OPTIMIZATION

Policies in Reinforcement learning can be parameterized [26, pp. 321]. A parameterized policy performs transformations on the state information in order to output action probabilities. The parameter vector is often referred to as θ, and a policy parameterized by θ is represented as π_θ. Policies can be parameterized in more than one way as long as the probabilities $\pi_\theta(a|s)$ are differentiable with respect to the parameters θ [26, pp. 322]. In recent RL development, most applications make use of Artificial Neural Networks (ANN) to compute action probabilities [3], [4], [6], [28], [29]. When this is the case, the parameters θ correspond to the ANN weights across all its neurons.

Policy gradient algorithms for Reinforcement Learning aim to maximize the return in (1) with respect to the parameters θ of a parameterized stochastic policy π_θ [26, pp. 321]. However, these type of algorithms can be prone to high variance, where a policy's action probabilities change abruptly between updates [6]. In an attempt to stabilize policy updates, Schulman *et al.* propose the use of *trust regions* that do not allow for a policy to diverge beyond a certain threshold from its former distribution [24], [30]. The Proximal Policy Optimization algorithm implements these constraints by clipping its loss function [24], which can be expressed as

$$L(\theta) = \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right] = \mathbb{E}_t \left[r_t(\theta) \hat{A}_t \right]. \tag{2}$$

The loss function is composed of a ratio $r_t(\theta)$ and an advantage estimation $\hat{A}_t$. The ratio is computed between the probability of action a_t from the current policy π_θ and the probability of the same action from $\pi_{\theta_{old}}$, the policy before the update [24]. The advantage is calculated via (3) and (4), also called Generalized Advantage Estimation (GAE) [31]. Its purpose is to take the return R_t into account whilst using an estimated value of the state $V(s)$ as a baseline.

$$A_t = \delta_t + (\gamma\lambda)\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \ldots + (\gamma\lambda)^{T-(t+1)}\delta_{T-t}, \tag{3}$$

$$\delta_t = r_{t+1} + \gamma V_\phi(s_{t+1}) - V_\phi(s_t), \tag{4}$$

where T is the time length to be considered. T does not need to match an entire episode, since (4) allows for bootstrapping with a value function $V_\phi(s)$ parameterized by ϕ. γ still acts as a discount factor for future rewards, however, $\lambda \in [0, 1]$ regulates between relying more on the value estimation $V_\phi(s)$, when $\lambda \rightarrow 0$, or relying more on the actual rewards received at each step, when $\lambda \rightarrow 1$ [31].

The loss in (2) is then clipped according to (5). Where $\min(x, y)$ takes the minimum between the two arguments separated by comma, and $\text{clip}(x, y, z)$ clips the value of the first argument into the bound delimited by the lower bound y and upper bound z. The clipping constant $\epsilon \in (0, 1)$ defines how far π_θ is allowed to diverge from $\pi_{\theta_{old}}$ [24].

$$L^C(\theta) = \mathbb{E}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right]. \tag{5}$$

The PPO algorithmic flow takes turns between collecting trajectories via interactions with the environment and making policy updates [24]. The clipping method allows for a series of subsequent updates, also called epochs, to happen with the same batch of data for a greater sample efficiency. Therefore, the algorithm can run multiple epochs on the same batch before discarding it and collecting new data. Heess *et al.* showed that PPO can also make use of distributed training [6]. That means one can instantiate multiple environments and acquire data from all of them in parallel [28]. Not only this process can speed up data collection, but it also allows for more diverse interactions to be batched together, which has shown to improve training stability [6], [28].

III. THE VERY SMALL SIZE SOCCER CATEGORY

The VSSS category proposes a soccer game between two teams of three robots each. The robots must act autonomously

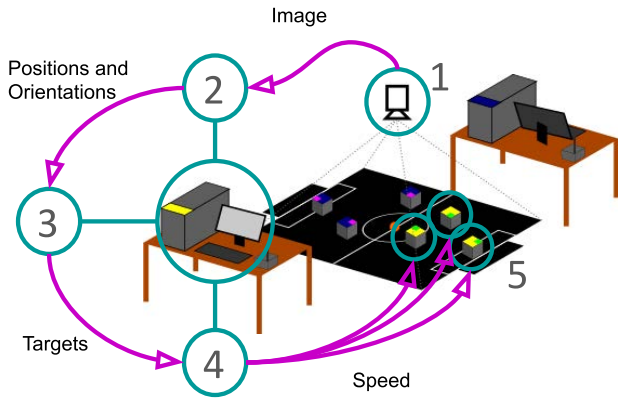

FIGURE 1. Diagram of the Very Small Size Soccer category processing flow: 1) Camera captures image; 2) Information is extracted from the image, such as positions and orientations of all objects; 3) Conditional algorithm chooses target positions for the robots; 4) Control system computes speed for each wheel; 5) Robots receive the target speeds via radio.

though they are allowed to process information on hardware not physically attached to them. Other than complying with a size limit, the teams are free to define their robot's structure and algorithms for both control and strategic decision-making. This freedom demands that those same algorithms are capable of generalizing against previously unseen strategies in order to beat them.

Fig. 1 shows the setup for a match. A camera is placed above the field for each team to capture images. One computer for each team may process the pictures, make calculations, and communicate with the robots. In order to facilitate the feature extraction process, the teams must have predefined colored tags on top of each robot, identifying both the team and the player. Fig. 1 also presents the flow of information processing used by the Pequi Mecânico team [19].

The decision process depicted in Fig. 1 can be formulated as a POMDP. We can regard the decision-making agent as a union between steps 2, 3, and 4 in Fig. 1. The image captured by the camera is the observation $s \in \mathcal{S}$ and the desired speed for each wheel of all three robots is an action $a \in \mathcal{A}$. The *true* state of the environment $z \in \mathcal{Z}$ is composed by all the information regarding positions, orientations, velocity, forces, and directions of all components in the field, and the overall probabilities of state transitions $P(z'|z, a)$ are dictated by the world's physics.

The formulation of the problem as a POMDP may vary depending on the focus of the application or research. Control-focused approaches can use target positions and actuator angles of a single robot as observations and velocities as actions, thus bypassing strategic decision-making. Strategy-focused approaches on the other hand may use current positions and orientations as observations and target positions as actions, thus omitting the image processing and control steps.

To formulate the robotic soccer problem as a Dec-POMDP we can first define each robot as an agent $d \in \mathcal{D} \equiv \{1, \ldots, n\}$ where $n = 3$. Each agent will have its own policy π^d. An agent's policy processes its observation s^d and outputs its action a^d. Observations can be different for each agent with local or relative information, or they can all process the same global view of the environment. The joint observation $\mathbf{s}$ is a vector of the observations for each agent $[s^1, s^2, s^3]$. Similarly, the joint action is the vector of actions for each agent $[a^1, a^2, a^3]$. Once again, the actions depend on the approach, only this time they are focused on a single robot. Thus if the actions are actuator speeds, or target positions, they will refer to a single robot. The environment also responds with a joint reward $\mathbf{r} = [r^1, r^2, r^3]$ which can be some form of individual scoring for each agent or a shared reward when the team scores a goal. The steps in Fig. 1 do not necessarily change, since the agent's processing can happen on outside hardware. Though it is important to say that each agent would have its own information flow. Our formulation of the robotic soccer as a Dec-POMDP is detailed in section VI.

IV. RELATED WORK

A. ROBOTIC SOCCER

The robotic soccer environment, as a sequential decision-making problem, has multiple variations. It has been formulated as a completely virtual environment with and without robotics [18], [32], or as real environments that escalate on difficulty of robot control and coordination from wheeled robots (i.e. the VSSS category) to humanoids [15], [33], [34]. There have been studies that use Reinforcement Learning in most variations of the robotic soccer game [17], [18], [23], [32], [35]. However, as stated before, they either focus on the control part or the strategy part of the problem.

1) CONTROL-FOCUSED APPROACHES

Control-focused approaches often optimize a subsection of the game. Sheikhlar and Fakharian use RL to make an adaptive controller that improves path following in an omni-directional robot [35]. Melo makes use of RL to control a simulated humanoid robot's joints to effectively kick the ball [23]. These tasks will unlikely single-handedly win the game, though they are definitely crucial parts of the problem. They are also by no means easy or simple, which helps us scope the capabilities of Reinforcement Learning.

2) STRATEGY-FOCUSED APPROACHES

Strategy-focused approaches have more in common with our work because of their ultimate goal. Strategic decision-making can be considered a high-level process, as opposed to control tasks that perform on actuators [7]. Thus, their target is often to win the game, or score goals. Shi *et al.* use RL as a method for selecting predefined strategies [17]. They map the game into a finite set of situations based on the ball's position and match them to a set of three predefined strategies. Liu *et al.* uses RL in a similar approach to ours as their algorithm also controls simulated agents in an effort to win the game [10]. They use population-based training where multiple learners train together. Though their

approach is similar to ours, their environment was not meant to depict real situations [10].

Other forms of strategic decision-making act by micro managing each agent. Instead of controlling their actuators, these approaches choose from a predefined set of actions for each agent individually [16], [18], [32], [36]. Xuanyu *et al.* and Zhao *et al.* both use the Sarsa algorithm to choose discrete actions for optimizing a task called Keepaway [32], [36]. The purpose of the task is to maintain ball possession whilst avoiding opponents to improve coordination. In their work, the algorithm chose among predefined actions such as passing, dribbling, and kicking [32], [36]. Fahami *et al.* uses RL to choose the best angle at which to kick in order to score [18]. They also define a finite set of actions representing possible angles for the algorithm to choose from. A Brazilian team called RoboCin used Deep Reinforcement Learning techniques to train a single robotic agent in a simulation of the VSSS category [16]. Similar to other studies, they used predefined actions for moving the robot. Since the game has multiple agents, the same policy was deployed on all robots for execution [16]. However, it is important to notice that other agents' behavior was not considered during training.

As we can see, most of the approaches focused on strategic decision-making have discrete action spaces whilst the control-focused ones use continuous action spaces. It is well established that continuous spaces are high-dimensional and take longer to explore [37]. The idea behind the use of continuous actions in control tasks is that they can be encapsulated without simulating a whole game or entire teams, and thus their state space is also smaller [22], [23]. For strategy tasks, it makes sense to discretize both action and state spaces to make the optimization process shorter and comply to algorithmic restrictions [16], [18], [32]. However, by doing so, one limits the possible combinations of state-action pairs, thus constraining the solutions an RL algorithm can provide. It has already been demonstrated that giving full control of joint motion in RL training can provide solutions and outcomes that the research team did not foresee [6].

B. MULTIAGENT REINFORCEMENT LEARNING

Multi-agent Reinforcement Learning allows for a series of combinations and different approaches since there are multiple learners interacting with the same environment. Agents can each learn a different behavior [11]–[13] or share parameters and learn the same policy together [5], [9]. Shao *et al.* use MARL to solve a StarCraft micro management problem which requires simultaneous commands for multiple in-game units [5]. They use a single neural network to control all the agents separately though simultaneously, meaning the agents share the same policy. Since each one has a different perspective of the environment, they do not act equally [5], [9]. Sharing parameters can save memory and create circumstances for a better generalization since the network learns from all the agents. It also allows for an increase in the amount and diversity of data collected from a single instance of the environment.

In some applications agents are able to communicate with each other [13]. It can be useful to allow agents to share individual information. However, this is not necessary when their information comes from a centralized source, which is the case for the VSSS category. Centralization in MARL can be useful during training, though not always desirable during execution [11]. Sharing policy parameters can be a form of centralization as it allows for training using experiences from all agents. Other forms of centralization use mixing networks as an attempt to centralize only the training process whilst maintaining each agent with their own learned policy [11], [38].

Shao *et al.* along with Muzio in his control tasks, use a technique called curriculum learning [5], [22]. This technique places the learner on tasks with increasing levels of difficulty which helps build up the knowledge towards more complex tasks [39]. Curriculum can also be achieved in competitive multi-agent games. Baker *et al.* achieve curriculum learning through another technique called self-play when solving a hide and seek game [9]. In this scenario agents play against themselves, or different versions of their policies. This method has proven to be useful for games that do not have a default adversary and has shown to be effective in complex game environments, such as Starcraft2 and DOTA2 [3], [4]. Given that agents are theoretically always at the same skill level, this technique can automatically create a curriculum learning. However, it needs to be used with caution. There are reports of cyclic behaviors emerging when the agent learns to beat its most recent versions, but forgets how to counteract older strategies [40]. Also, in some environments with two competing learning agents, Bansal *et al.* show that when one starts winning, the other becomes unable to learn whilst always being bested by its opponent [41].

V. SIMULATED ENVIRONMENT

Since RL algorithms require a lot of timesteps to converge, training on real robots is unfeasible. Therefore, we resorted to a simulated environment. To simulate the robotic soccer environment, we require a physics simulator. We chose the MuJoCo physics engine, designed for fast and accurate robot simulation [42]. The simulation was constructed following the official rules of the VSSS category [15]. We also coded into the simulator a virtual referee to incorporate situations where a team commits a foul and the robots have to be rearranged. The two main situations that require rearrangement are for a *penalty kick* and a *free ball*. The penalty kick occurs when two players of the defending team enter the large area in front of the goal along with the ball, which grants the attacking team a penalty kick. A free ball occurs when the ball has not moved for ten seconds. The measurements of the field and its markings can be seen in Fig. 2.

Since other teams' robots were unavailable for measurement, we used the Pequi Mecânico's robots as models. There was no need to create models of every component inside the robot, such as controllers, radio, and battery. Therefore, we simplified the model into fewer pieces; a central block,

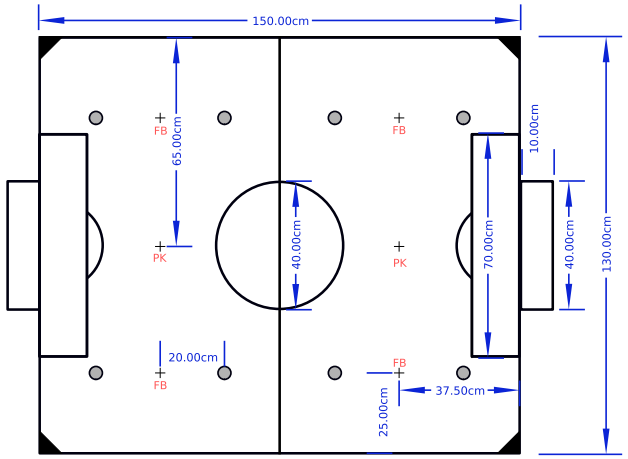

FIGURE 2. Diagram of the VSSS field measurements. PK stands for Penalty Kick, and FB stands for Free Ball. The crosses show where the ball starts from when a foul is committed. The gray circles represent where the robots start from in the case of a Free Ball.

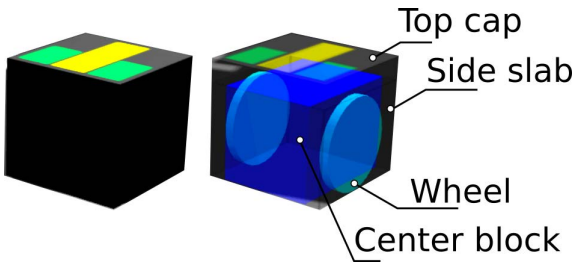

FIGURE 3. Simulated robot in MuJoCo.

TABLE 1. Dimensions for the pieces of the simulated robot.

Part	Dimensions	Mass
Center block	7.9 cm x 6.0 cm x 6.0 cm	396 g
Side slab	0.05 cm x 8.0 cm x 6.0 cm	1 g
Top cap	8.0 cm x 8.0 cm x 1.5 cm	1 g
Wheel (on each side)	radius: 3.0 cm; width: 0.6 cm	12 g

two wheels, a top piece, and four slabs, one for each side. Fig. 3 shows a diagram of the simulated robot pieces and in Table 1, their measurements.

VI. MODELING

As our goal is to apply RL to strategic decision-making and control, we aimed to replace steps three and four of the diagram in Fig. 1. To model this problem in terms of a Dec-POMDP we started by defining each robot in the field as an agent d. Each agent has its own observations $s^d \in \mathcal{S}$ and actions $a^d \in \mathcal{A}$. The observations are composed of positions and orientations relative to the agent, and each action is a tuple of two continuous values of speed, one value for each wheel of the agent. Each agent will have a policy π^d to make decisions. However, instead of parameterizing each policy individually, we have them share the same parameters θ. All policies are parameterized by the same neural network architecture, and share the same weights. In other words, at each timestep t the neural model runs once for each agent, reading its observation s^d and outputting its action a^d. The combination of all three actions is the unified action $\mathbf{u} \in \mathbf{U} \equiv \mathcal{A}^n$ of that timestep. Similarly, the rewards are returned for each individual agent r^d, where the goal is for each agent to maximize its return function (1). The agents do not communicate or share observation information between each other.

An agent's policy can be trained with a set of transitions, where a transition is represented as the tuple: (s^d, a^d, r^d, s'^d). Since our agents share the same parameters θ, their shared policy can be trained with transitions acquired from all agents. Therefore, our configuration allows us to train our models with more experiences than with either a single agent approach or a multi-agent approach with individual policies. In both alternate cases each timestep would provide a single transition for the single agent policy or for each individual policy. Meanwhile, our configuration collects one transition for each agent, all of which can be used to train the same shared policy.

To maintain consistency and allow future applications, we composed the agent's observation s^d as a function of the Pequi Mecânico's visual system's output (i.e. positions and orientations). We also added noise to the measurements comparable to the visual system, sampled from a Normal distribution [43]. The noises had standard deviations of 1.854e-3 m for the position along the X axis, 1.679e-3 m for the position along the Y axis, and 3.123e-2 rad for the orientation of players. By combining positions and orientations, each agent receives the following information as input:

- Cartesian positions of itself, allies, adversaries, and the ball, zeroed at the center of the field.
- Sine, cosine, and arc tangent of its orientation.
- Sine, cosine, and arc tangent of the angles formed by the line between the agent and each ally, each adversary, and between the ball and each goal center.
- Euclidean distances between the robot and each ally, each adversary, and the ball.
- Distance between the ball and each goal.
- Actions chosen by all ally agents in $t - 1$.
- Time left in episode.

To add time-sensitive information, such as direction of movement and speed we stacked the data eight fold. The Pequi Mecânico's visual system processes thirty frames per second [19]. However, we only use ten readings per second to avoid stacking nearly identical states [2], [44]. All input data is normalized by their respective maximum values to remain bounded in $[-1.0, 1.0]$.

Each agent requires two values of speed in radians per second, one for each wheel. Therefore an action a represents a tuple of two values. Since the real robots have limitations in speed, we used the same limitations to define our action space. The Pequi Mecânico's actuators have a speed limit of 46.66 rad/s. Thus our action space is defined as $\mathcal{A} \equiv \mathcal{V}^2 : \mathcal{V} \rightarrow [-1.0, 1.0]$ where we can scale a multiplying it by the maximum speed.

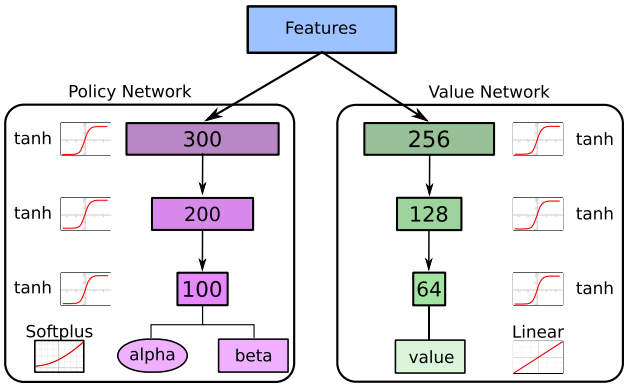

FIGURE 4. Diagram of the feed forward architecture of both policy and value networks. The number of neurons is shown in each layer, and the activation functions are shown on the sides.

A. NEURAL NETWORK MODEL

As explained in Section II-B the Proximal Policy Optimization algorithm has two parameterizable functions, a policy π_θ and a value function V_ϕ. To approximate these functions we used artificial neural networks. Fig. 4 shows the network architecture in more detail. The model for π_θ is called the *policy network* whilst the model for V_ϕ is called the *value network*.

In order to output continuous actions we used Beta probability distributions [29]. By definition these distributions are bounded in [0.0, 1.0] and thus fit into our control problem where there is a maximum and minimum speed for the wheels. Beta distributions are defined by two parameters α and β. Our policy network outputs a pair of each. A single (α, β) tuple defines a distribution from which a single continuous value is sampled. Therefore we end up with two continuous values sampled from different distributions, one for each wheel. Additionally, we add $+1$ to both α and β to avoid bimodal distributions where the extreme opposites would have high probabilities [29].

B. REWARD FUNCTION

To create a reward function for this problem, we defined two types of reward components, specific and universal [10]. Specific components consider the agent's aspects and behaviors alone whilst universal components consider the entire team's performance. Together, they create the reward signal; the agent itself does not perceive the difference between them; it only receives their linear combination.

As part of the universal reward, we used two main aspects of the game, the proximity of the closest team member to the ball, and the speed of the ball towards the adversary's goal, both represented in Figs. 5(a) and 5(b) respectively.

The speed-related reward shown in (6) and (7) is based on the distance traveled by the ball in a single timestep towards the goal. $\text{dist}(\mathbf{b_t}, \mathbf{G})$ computes the Euclidean distance between the ball's position $\mathbf{b}$ at time t and the goal's position $\mathbf{G}$. We normalize it by a maximum of 0.14 m and subtract a small value of 0.05 to send a negative reward signal, or punishment, should the ball remain still, thus shifting its point of zero reward.

$$v = \frac{\text{dist}(\mathbf{b_{t-1}}, \mathbf{G}) - \text{dist}(\mathbf{b_t}, \mathbf{G}) - 0.05}{0.14} \tag{6}$$

$$r_{speed} = \text{clip}(v, -1.0, 1.0) \tag{7}$$

The proximity term is presented in (8) and (9), and takes into consideration only the closest team member to the ball. The distance is calculated between each agent's position $\mathbf{d} \in \mathbf{D}$ at time t for a single team $\mathcal{D}_a \subset \mathcal{D}$. As the distance value cannot be lower than zero, this term receives only an upper bound of 1.0 m. This choice is intended to show that there is little difference between being 1.0 m away from the ball and 1.7 m away. The reward reflects that the agent in those cases has little influence in the ball's motion either way.

$$\mathcal{P} = \text{dist}(\mathbf{D}_{a,t}, \mathbf{b}_t) \tag{8}$$

$$r_{dist} = \begin{cases} -1, & \text{if } \min(\mathcal{P}) \geq 1 \\ -\min(\mathcal{P}), & \text{if } \min(\mathcal{P}) < 1 \end{cases} \tag{9}$$

Moving on to the specific rewards, we decided to take a unique approach. By looking at the middle angle between three points *a, b*, and *c*, it is possible to create a reward signal based on the relative position of the agent. If the agent is positioned between the ball and the goal, we consider it a defensive position. If the agent is positioned behind the ball with a straight line to the adversary's goal, we consider it an offensive position. We could not determine whether it was better to keep an offensive or defensive positioning, therefore, we placed both as terms of the reward signal. The equations for computing these angles are depicted in (10) to (13).

$$\overrightarrow{ba} = a - b \tag{10}$$

$$\overrightarrow{bc} = c - b \tag{11}$$

$$\psi = \arccos\left(\frac{\overrightarrow{ba} \cdot \overrightarrow{bc}}{||\overrightarrow{ba}|| \times ||\overrightarrow{bc}||}\right) \tag{12}$$

$$r_{pos} = \frac{\psi}{\pi} - 1.0 \tag{13}$$

To measure the value for an offensive position r_{off}, we set the points *a, b*, and *c*, as the robot in question, the ball, and the adversary's goal, respectively, as presented in Fig. 5(d). Thus, the closest the angle formed at the ball is to π, the greater is the reward signal. In this case, the angle approaches π as the robot's position lines up behind the ball and the goal. Similarly, to compute the value of a defensive position (r_{def}), we set those points as the ball, the robot, and the current team's goal, respectively, as presented in Fig. 5(c). The angle approaches π while the robot positions itself between the ball and the goal. Both reward signals are normalized within the interval $[-1.0, 0.0]$. It is important to notice that even though these reward signals might seem competing, they are not. It is possible for the agent to maximize both depending on the situation. Should the ball be at the center of the field and the robot directly behind it, it has simultaneously an offensive and defensive position.

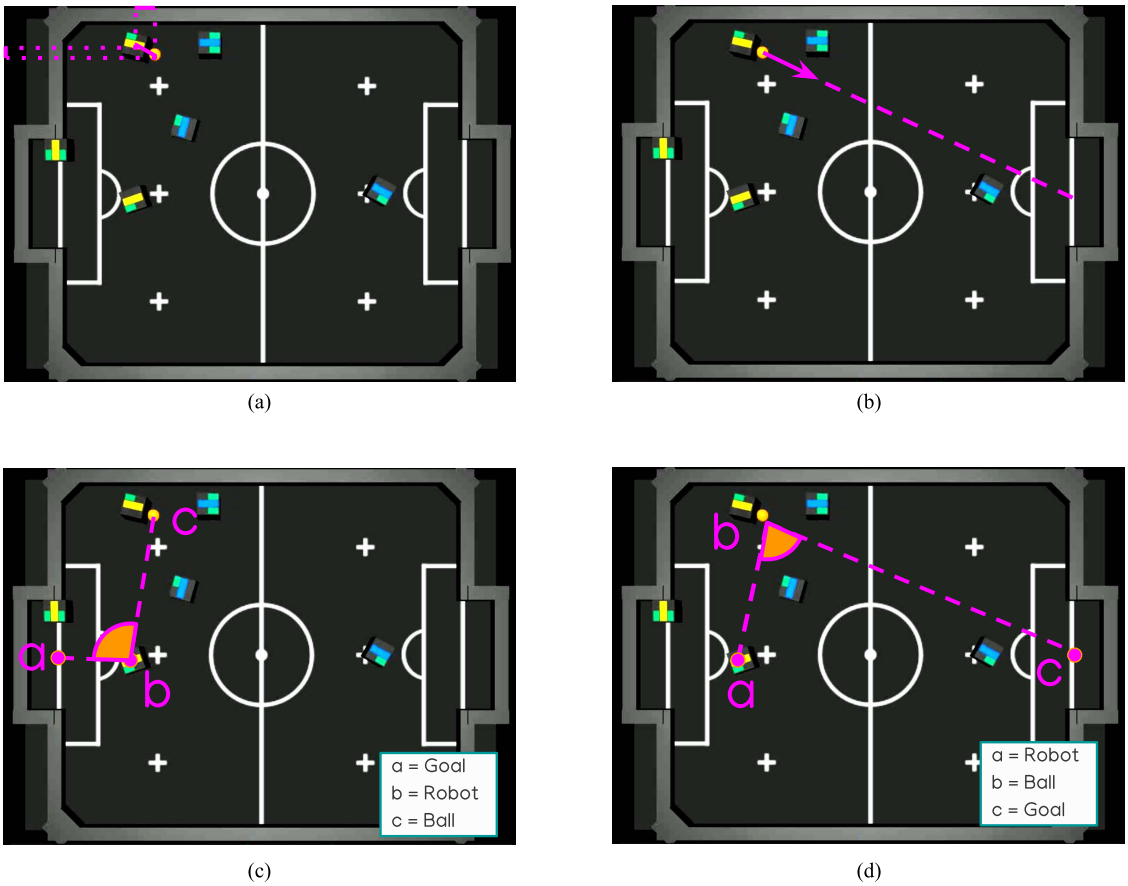

FIGURE 5. Different reward terms that compose the final reward signal. (a) presents the distance of the closest team member to the ball; (b) is the velocity of the ball towards the adversary goal; (c) is a defensive angle formed between the team's goal, a robot, and the ball; and (d) is an offensive angle, formed between a robot, the ball, and the adversary goal.

Finally, the resulting reward signal is a linear combination of all terms, both universal and agent-specific rewards. As shown in (14), the speed receives the biggest weight because it summarizes how fast the ball is getting to, or away from, the objective. As the other three terms, we could not decide which one was most important, if it was a good offensive or defensive positioning, or a better control of the ball's motion. Therefore, we determined equal weights, so the agent would then decide which ones to maximize. In the worst case, depleting one to favor the other would not result in any loss of reward.

$$r = 0.7\, r_{speed} + 0.1\, r_{dist} + 0.1\, r_{off} + 0.1\, r_{def} \tag{14}$$

In addition to the continuous reward present at each step, we also used a scoring and conceding reward of $+10$ and -10 respectively, [10], [17]. In order to translate the fact that a goal scored earlier in the episode was better than a later one, we weighted those values by the remaining time:

$$g = \begin{cases} +10, & \text{if scoring} \\ -10, & \text{if conceding,} \end{cases} \tag{15}$$

$$r_{goal} = g\left(1 + \frac{(T-t)}{T}\right). \tag{16}$$

where T is the total time of an episode, and t is the current timestep. We also used a -1.0 reward for getting a penalty foul. In the event of a goal or foul, this reward replaced the one in (14) and was given to all players of the team equally.

VII. METHODOLOGY

A. TRAINING ALGORITHM

Our algorithmic flow follows the distributed method for PPO. In our case, we ran 200 instances of the simulated environment in parallel. Given that we modeled the problem as a multi-agent problem, we were able to collect one transition of state, action, reward and next state, per agent per environment. This means for each timestep of a single environment we collect three different transitions, which gives us more diversity in a batch of data. We can only use them in the same batch because our agents share the same policy, thus it is updated with data from all of them.

When running our policy in multiple environments, we collect a total of 128,400 experiences for our batch. After the data is collected, we use it to update our policy and value networks through five epochs. After five updates, the data is discarded and the environments resume to collect new data. This process continues until the model has collected 160 million transitions, or experiences, in total. Other hyperparameters

TABLE 2. Hyperparameters for training.

Hyperparameter	Value
Lambda (λ)	0.95
Gamma (γ)	0.99
Clip constant (ϵ)	0.2
Learning Rate	0.0004

for the PPO algorithm, mentioned in II-B are displayed in Table 2.

A real-life match of the VSSS category has a maximum time of twenty minutes. However, since the robot's positions essentially reset after a goal, and the intent of the team is to score more, regardless of the score, we decided to end the match after the first goal. Also, to quickly end matches where no team scores, the maximum time was shortened to forty seconds.

B. SELF-PLAY

The Very Small Size Soccer category is a competitive environment that does not have a default strategy. Therefore, we had to define an opponent against which our policy would train. One option would be to use the best strategy available to us from the Pequi Mecânico team. Another option was to use self-play since the VSSS environment is symmetric. We executed training runs with both types of opponents, the evaluation methods and results are presented in the following sections.

When using self-play training, we chose to set the opponent as an earlier version of the learning model. The adversary is fixed, thus it does not learn or collect experiences. However, it is noticeable that eventually the learning model would improve beyond its opponent and overfit against it. To keep the opponent stable, yet at the same skill level of the current policy, we used a measurement of score that is computed across multiple games. The score

$$s = \frac{G_s - G_c}{n}, \tag{17}$$

measures the rate of the difference between goals scored, as G_s, and goals conceded, as G_c, across n number of games. Whenever the score achieved a certain threshold, we would update the adversary to the newest version of the model. We set the threshold for updating the opponent at 0.6, in an attempt to require more goals scored than conceded, yet not allowing for the learning model to overfit on a single opponent.

C. EVALUATION

In order to evaluate a learned policy we required fixed opponents. As the VSSS environment does not provide a default strategy for evaluation, we used the Pequi Mecânico's heuristic strategies. The Brazilian team provided two different approaches. The VSSS-EMC code has been in development for longer and, in 2019, placed fifth in the Latin American Robotics Competition. The VSSS-INF code is newer and uses a different strategy. We also added to the opponent set, a strategy that behaves randomly. This random adversary chooses the speeds of the robots' wheels from a uniform distribution. With a set of different opponents we can evaluate not only the performance of a policy, but also its generalization capabilities.

TABLE 3. Evaluating heuristic algorithms in one hundred matches against a strategy that behaves randomly.

Heuristic	VSSS-EMC	VSSS-INF
Win Rate	**0.99**	0.98
Draw Rate	0.01	0.02
Loss Rate	0.0	0.0
Mean Goals Scored	**4.94**	4.82
Mean Goals Conceded	0.18	0.51
Mean Score	**4.76**	4.31

In the evaluation process the learned policy plays one hundred matches against each of the three opponents, VSSS-EMC, VSSS-INF, and the random strategy. These evaluation games, apart from training matches, did not end upon the first goal, and had a time limit of 120 seconds. Following the rules of the VSSS category however, the game is interrupted when the difference of goals reaches 10. This process of running evaluation games was repeated across multiple checkpoints during training. Each checkpoint configures 50 loops of the PPO algorithm, which means, with our specifications, 2.14 million timesteps executed.

The metrics used in the evaluation process were computed across all one hundred games for each different opponent. They are the following:

- Win, loss, and draw rates.
- Mean scored and conceded goals.
- The score metric from (17).

VIII. RESULTS

All experiments were done in a machine with Intel(R) Xeon(R) CPU E5-2698 v4 @2.20GHz processors totaling 80 cores, and 528GB of RAM running Ubuntu 16.04. We did not use any GPU resources in these experiments.

The first experiment we executed was intended to measure the performance of both heuristic strategies provided by the Pequi Mecânico Team. We conducted a round of one hundred games against the random strategy for each of the heuristics. Results are shown in Table 3. It is clear that against the random opponent both heuristics had high win rates, goals scored and mean score, with slightly higher values for the VSSS-EMC.

We also conducted another round of one hundred games, however, this time, placing the VSSS-INF and VSSS-EMC heuristics against each other. The results are shown in Table 4. This new round of matches confirms that the VSSS-EMC heuristic surpasses the VSSS-INF by winning 56% of matches with a positive mean score of 0.68.

As stated before, we ran two types of training, a training with self-play, and a training where the opponent was fixed as the best heuristic strategy available. In light of our first experiments' results, such strategy was defined as the VSSS-EMC. We set a maximum of 160 million transitions, or experiences, collected for both training experiments. Since one timestep

TABLE 4. Evaluating heuristic algorithms in one hundred matches against each other.

Heuristic	VSSS-EMC	VSSS-INF
Win Rate	**0.56**	0.29
Draw Rate	0.15	0.15
Loss Rate	0.29	0.56
Mean Goals Scored	**1.95**	1.27
Mean Goals Conceded	1.27	1.95
Mean Score	**0.68**	-0.68

of an environment provides us three different transitions (i.e. one for each agent), this maximum can be interpreted as 53.33 million timesteps executed in total. We used Bootstrap Confidence Intervals among five runs to acquire statistical significance [45]. The error marks in the following result charts show the interval where the real value lies, with 95% confidence.

A. COMPARATIVE RESULTS

Fig. 6 shows the win rate evaluation results for both experiments against only the VSSS-EMC strategy, without the other opponents. In green we have the training with self-play, and in blue, the training against the VSSS-EMC. Thus the blue line represents a model being evaluated against the same adversary we used to train it. Each point in the graph represents one hundred evaluation matches of a checkpoint during the model's training. Meanwhile, Fig. 7 shows the score, from (17), of these experiments against the VSSS-EMC.

From Figs. 6 and 7 we can see that training against the heuristic alone showed to be inadequate for the learning process. The experienced adversary won most training and evaluation matches. The score metric in Fig. 7 shows that the model was even unable to defend, as it reaches the lowest possible value of -10. At its earliest stage, the model is mostly random. Thus the VSSS-EMC strategy would have little trouble scoring goals and ending episodes early. By not allowing the model to explore both action and state spaces properly, it essentially deprived the agents of experiences for improving. The narrow confidence intervals on the blue line show us how the five runs had steady performances, meaning the results are consistent among them.

On the other hand, the self-play training shows an ascending curve in both win rates and scores. The ascending difficulties of the adversaries allowed the model to explore different states and actions. Therefore it learns different aspects of gameplay progressively. The self-play experiments show wider confidence intervals, which indicates more diverging values among the five executions. However, as the ascending curve implies, we verified consistent ascending performance across the five runs. The following subsections explore both training configurations' results in more detail.

B. TRAINING AGAINST VSSS-EMC

In Fig. 8, we can see evaluations regarding only the training against the VSSS-EMC heuristic. Each line in the graph represent a different opponent in the evaluation process.

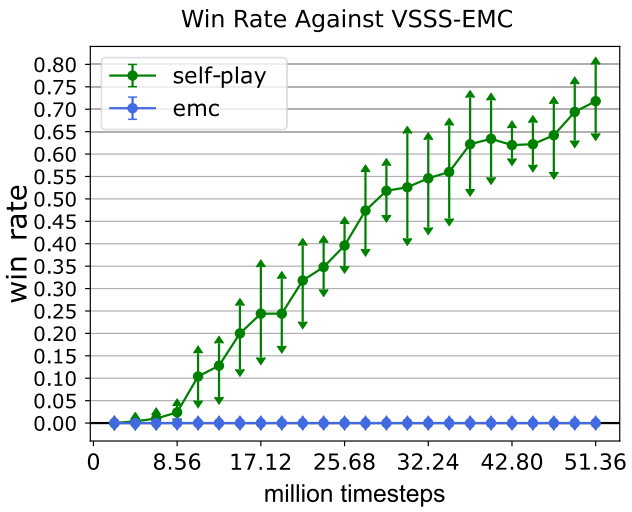

FIGURE 6. Win rate evaluation against VSSS-EMC heuristic. The green line shows the training with self-play. The blue line shows the training with the VSSS-EMC as a fixed opponent.

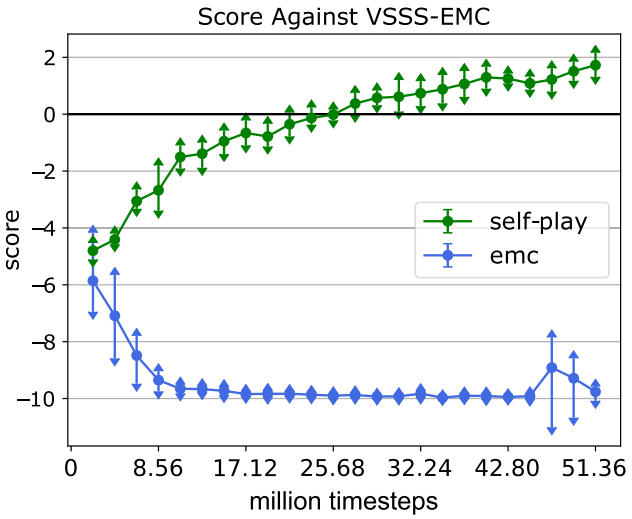

FIGURE 7. Score against VSSS-EMC heuristic. The green line shows the training with self-play. The blue line shows the training with the VSSS-EMC as a fixed opponent.

The win rate measure against the random strategy in Fig. 8(a) is a way to assess how fast the model is learning, as it plays against a supposedly easy opponent with no data processing whatsoever. By looking at Fig. 8(s), it is possible to observe that the win rate did not increase; it decreased throughout the training. We could use the draw rate as another indication of learning. If the model is losing less, and obtaining more draws, this is an indication that it is learning to defend. However, this is not the case. As Fig. 8(b) shows, there is no apparent change in the draw rates as well as the loss rates in Fig. 8(c). Finally, we can use the score metric to definitively observe if the model is conceding less goals as training progresses. Nonetheless, Fig. 8(d) shows that the score metric regarding all opponents have decreasing aspects. Both measures against VSSS-INF and VSSS-EMC heuristics achieve the lowest possible value of -10.

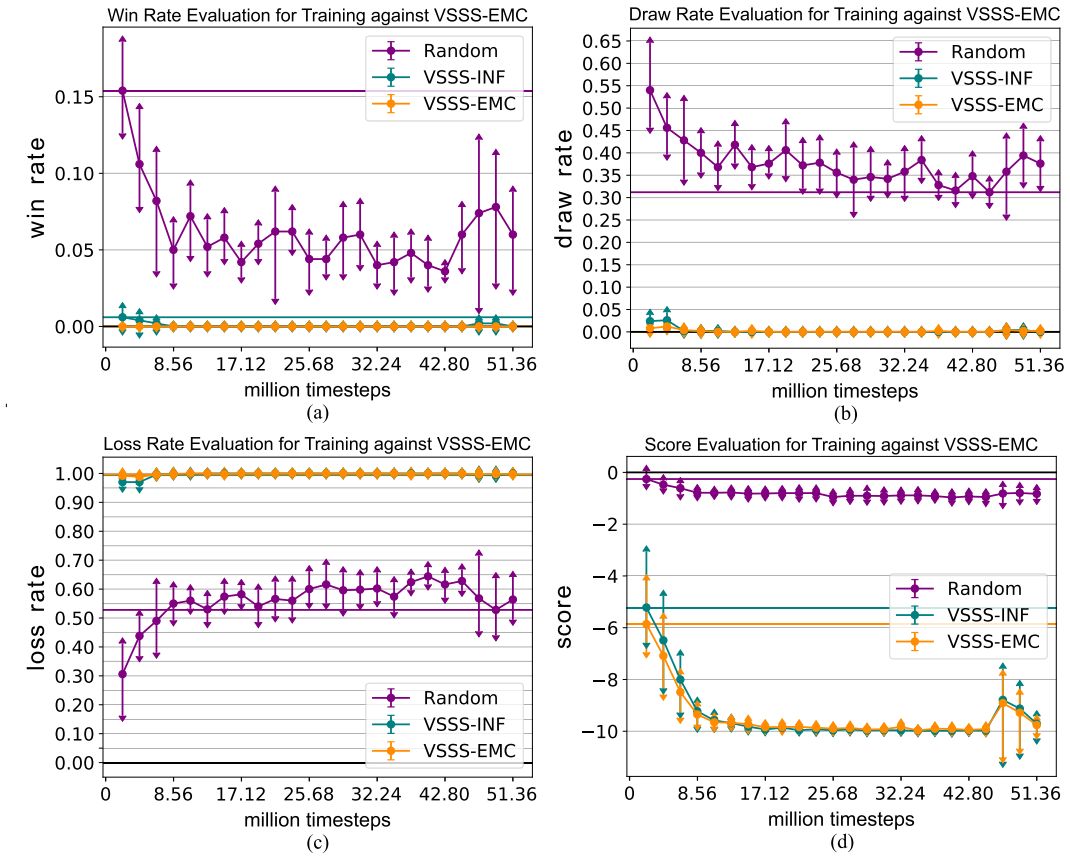

FIGURE 8. Evaluation metrics for the training against the VSSS-EMC heuristic. Each line represents the values against a different opponent. (a) Win rate; (b) draw rate; (c) loss rate; and (d) score metric. Horizontal colored lines represent the best value achieved against each opponent, in (a) and (d) they show the highest value, and in (b) an (c) they show the lowest values.

At sight of these results, a reasonable conclusion could be that there is a bug, or an error in the algorithm itself disrupting the learning process. Therefore, we compared the mean reward of both training configurations, as shown in Fig. 9. We can see that in both cases the reward increases as the training progresses. This leads us to conclude that the algorithm is actively learning, only at a slower pace.

This behavior is an example of how the design of the problem affects the learning process. In our case, the fundamental control aspects of the robots, such as going forward, backward, and turning, are the basis for creating more sophisticated strategies. However, it is all bound to the ultimate purpose of scoring goals. When playing against an experienced adversary, it scores so early that there are not enough time steps in the episode to explore the action space for even the most basic behaviors. It is essential to remember that in Reinforcement Learning, the model has no prior knowledge of the problem. Thus, by not scoring goals the model does not acquire knowledge of that state, in order to pursue it during training.

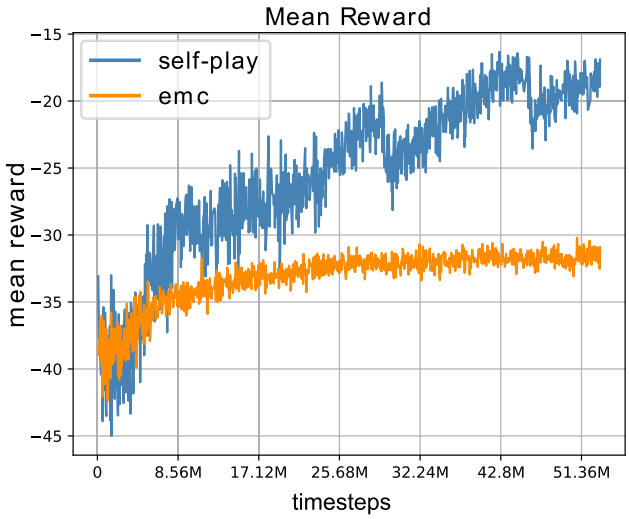

FIGURE 9. The mean reward averaged over the last 100 episodes. The blue line represents the training with self-play. The orange line represents the training with the VSSS-EMC as a fixed opponent. Distinctive drops in the self-play training show when the adversary was updated.

C. TRAINING WITH SELF-PLAY

Fig. 10 shows the results of the training using self-play. At first glance in Fig. 10(a), it is possible to see how the model quickly overcomes the random opponent. It achieves near zero percent loss rate very early in training, as seen in Fig. 10(c). Its win rate takes longer but eventually achieves

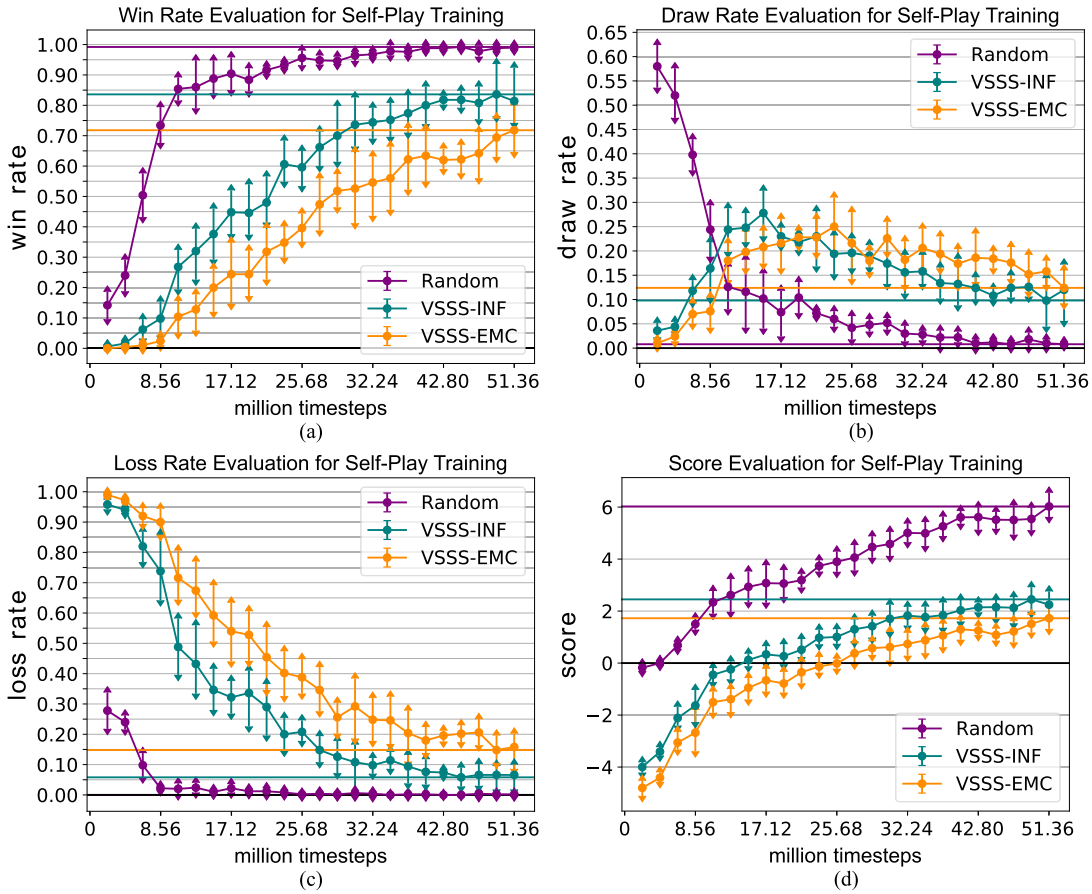

FIGURE 10. Evaluation metrics for the training with self-play. Each line represents the values against a different opponent. (a) Win rate, (b) draw rate, (c) loss rate, (d) score metric. Horizontal colored lines represent the best value achieved against each opponent, in (a) and (d) they show the highest value, and in (b) an (c) they show the lowest values.

a hundred percent. It is possible to see, by the score in Fig. 10(d), that even though it manages not to lose against the random opponent, it still takes more time to improve its scoring technique.

The curves representing games against the VSSS-INF and VSSS-EMC strategies have similar forms among themselves. As expected, the model's score and win rate are higher against the former than the latter in Fig. 10(a). Their shape indicates that the self-play technique indeed managed to create a task with increasing difficulty. By starting against a somewhat random adversary, the learning agent can explore the state and action spaces in order to acquire fundamental control aspects and knowledge of the environment. Whenever the adversary changes, the model transfers the knowledge to the new task, and learns more complex aspects of gameplay in order to defeat its new opponent. This scaling in difficulty is similar to the one used in curriculum learning [5], and when it happens without specific human design, it is called autocurricula [9].

The score in Fig. 10(d) shows that the model was only able to win more matches than lose, after approximately 14.98 million timesteps against the VSSS-INF strategy, and after 23.54 million against the VSSS-EMC. Before that, it had a distinct difficulty against the more elaborate strategies. The stages for overcoming the heuristic algorithms can also be seen in the draw rate curves in Fig. 10(b). They present a notable rise in the beginning and a slow descent afterward. This shape reflects how the model starts by losing, then improves towards a draw, and later winning.

These curves show us that the model trained with self-play was able to generalize its knowledge against opponent strategies it has never learned against. Since it learned on games against a past self, the VSSS-INF and VSSS-EMC strategies are completely new to it. Nonetheless, it managed to achieve higher win than loss rates and positive scores against all three strategies. It is interesting to notice that these curves do not show a clear convergence point, meaning their values could be higher with more training time.

In order to achieve a more stable convergence point for the self-play training, we ran the experiment for an extra 160 million transitions, totaling 320 million. The results for this extended training can be found in Fig. 11. From the curves against a random opponent, we can see how the model maintained its hundred percent win rate in Fig. 11(a) while still increasing the score measure in Fig. 11(d).

For the VSSS-INF and VSSS-EMC adversaries, it is possible to notice signs of convergence given how the win rate increases at a slower pace. Win rates peak at 94% against the VSSS-INF strategy and 84% against the VSSS-EMC.

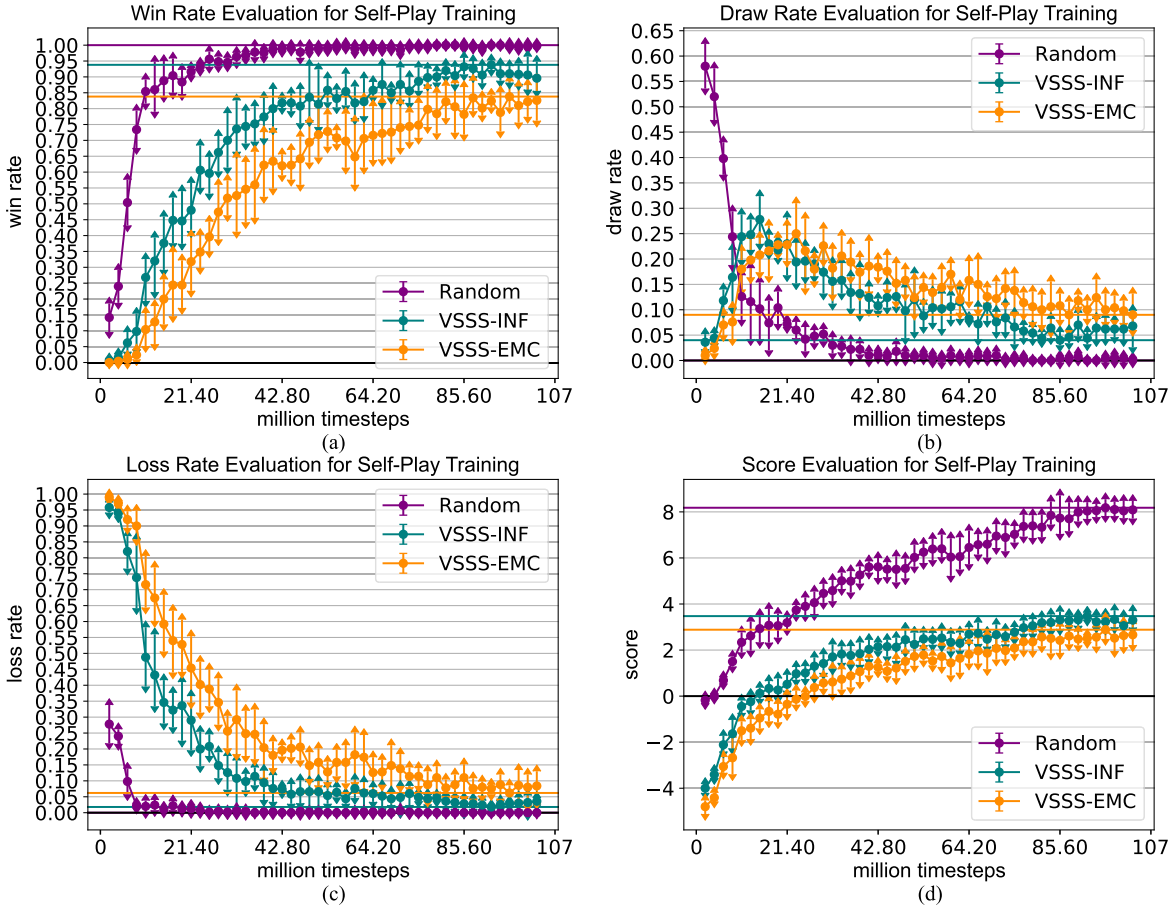

FIGURE 11. Evaluation metrics for the extended training with self-play. Each line represents the values against a different opponent. (a) Win rate, (b) draw rate, (c) loss rate, (d) score metric. Horizontal colored lines represent the best value achieved against each opponent, in (a) and (d) they show the highest value, and in (b) an (c) they show the lowest values.

TABLE 5. Results for the extended Self-Play training taken from the mean of the bootstrapped values.

Opponent	Random	VSSS-INF	VSSS-EMC
Win Rate	1.0	0.938	0.838
Draw Rate	0.0	0.04	0.09
Loss Rate	0.0	0.018	0.062
Score	8.179	3.474	2.884

The loss rate decreased and achieved values as low as 6.2%. The score measure against the heuristics kept increasing, though at a slower pace as well. Table 5 shows the values achieved by the end of this extended training.

D. SUBJECTIVE ANALYSIS OF BEHAVIORS

Upon watching videos of matches, we were able to notice some interesting behaviors and patterns.[5] We observed that early in training, the agents learn to pursue the ball and position themselves behind it. This behavior starts crudely with robots going back and forth, attempting to catch the ball from behind. Later on, it is possible to observe more complex curves to catch the ball even in motion. We also recognized a typical blocking behavior used by teams on the VSSS category. Whenever the ball slides on the sides of the field, a robot positions itself perpendicularly to the wall blocking the passage. The standard response from the other team on competitions is to spin the robot, throwing the ball towards the center of the field. However, our model learned to push forward by hammering the ball through the blockade. It starts with one robot going back and forth. By the end of the training, it is possible to see a coordinated hammering with the entire team.

We perceived a distinct difficulty in our model to defend penalty situations. Two reasons come to mind when analyzing this deficiency. The first is that there might not have been enough penalty situations during training for the model to properly learn to behave in them. The other reason could regard the positioning of the robots in such situations. We assumed that the best orientation for defending a penalty was with the defending robot pointed straight at the ball, similar to how the VSSS-EMC strategy behaves. Thus we programmed the simulation to reposition the robots this way. However, now we see that this was a possibly harmful bias inserted into the learning process.

[5] Video of the observed behaviors can be found in supplementary material.

IX. CONCLUSION

In this work, we apply Reinforcement Learning into the Very Small Size Soccer category, a robotic soccer environment. By using multi-agent and self-play techniques, we show how RL is capable of learning both the control and strategic aspects of the problem.

We analyze the learning process by comparing two different types of training: training with self-play, and training against Pequi Mecânico's VSSS-EMC hand-coded heuristic. By comparing the results, we show how self-play is essential for learning in this environment. It builds up knowledge by matching the learning model's skill as an adversary and allowing exploration.

The model trained with self-play was able to generalize and defeat strategies it had never played against. It managed to win against the Pequi Mecânico's best strategy with an 84% win rate. Meanwhile, the hand-coded heuristic used as an experienced adversary, did not allow the model to explore the state-space and develop new behaviors promptly. The results suggest that with enough training time, the algorithm may indeed learn by training against this heuristic. However, assessing how much time it would take is unfeasible with the computational power available.

Our unified approach for both control and strategic decision making is promising as it manages to overcome Pequi Mecanico's hierarchic solutions in the simulated environment. This can be concluded from the final winning rates of our model against the Pequi Mecanico's strategies. It also confirms our hypothesis that Reinforcement Learning is indeed capable of learning these two problems as one.

This approach gives room for a series of possible future works. The most important one being the deployment of the model on real robots. As much as we added elements to approximate our simulation to a real scenario, it might require some form of fine-tuning to make the sim-to-real transfer. We also believe that training could be improved to save time. By using a technique called *Imitation Learning* [46], we could create a model that firstly mimics Pequi Mecânico's heuristic behavior and then fine-tunes the model using RL with self-play. Applying this technique would even generate further analysis on how the imitated heuristic affects the final behavior.

REFERENCES

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," Deepmind Technol., London, U.K., Tech. Rep., 2013. [Online]. Available: https://arxiv.org/abs/1312.5602v1

[3] O. Vinyals *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, Nov. 2019.

[4] (2018). OpenAI. *OpenAI Five*. Accessed: May 3, 2019. [Online]. Available: https://openai.com/five/

[5] K. Shao, Y. Zhu, and D. Zhao, "StarCraft micromanagement with reinforcement learning and curriculum transfer learning," *IEEE Trans. Emerg. Topics Comput. Intell.* vol. 3, no. 1, pp. 73–84, Feb. 2019.

[6] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," 2017, *arXiv:1707.02286*.

[7] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robot. Autom. Lett.*, vol. 2, no. 4, pp. 2096–2103, Oct. 2017.

[8] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," 2018, *arXiv:1808.00177*.

[9] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–28.

[10] S. Liu, G. Lever, J. Merel, S. Tunyasuvunakool, N. Heess, and T. Graepel, "Emergent coordination through competition," in *Proc. 7th Int. Conf. Learn. Represent. (ICLR)*, New Orleans, LA, USA, May 2019, pp. 1–19.

[11] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. N. Foerster, and S. Whiteson, "Monotonic value function factorisation for deep multi-agent reinforcement learning," *J. Mach. Learn. Res.*, vol. 21, pp. 1–51, Aug. 2020.

[12] K. Ndousse, D. Eck, S. Levine, and N. Jaques, "Emergent social learning via multi-agent reinforcement learning," in *Proc. 38th Int. Conf. Mach. Learn. (ICML)*, vol. 139, M. Meila and T. Zhang, Eds., Jul. 2021, pp. 7991–8004.

[13] Q. Zhang, K. Dehghanpour, Z. Wang, F. Qiu, and D. Zhao, "Multi-agent safe policy learning for power management of networked microgrids," *IEEE Trans. Smart Grid*, vol. 12, no. 2, pp. 1048–1062, Mar. 2021.

[14] J. Cui, Y. Liu, and A. Nallanathan, "Multi-agent reinforcement learning-based resource allocation for UAV networks," *IEEE Trans. Wireless Commun.*, vol. 19, no. 2, pp. 729–743, Feb. 2020.

[15] (2021). *Rules for the IEEE Very Small Competition*. [Online]. Available: http://www.cbrobotica.org/wp-content/uploads/2021/05/vssRules3x321.pdf

[16] H. F. Bassani, R. A. Delgado, J. N. de O. Lima Junior, H. R. Medeiros, P. H. M. Braga, and A. Tapp, "Learning to play soccer by reinforcement and applying sim-to-real to compete in the real world," 2020, *arXiv:2003.11102*.

[17] H. Shi, Z. Lin, K. S. Hwang, S. Yang, and J. Chen, "An adaptive strategy selection method with reinforcement learning for robotic soccer games," *IEEE Access*, vol. 6, pp. 8376–8386, 2018.

[18] M. A. Fahami, M. Roshanzamir, and N. H. Izadi, "A reinforcement learning approach to score goals in RoboCup 3D soccer simulation for nao humanoid robot," in *Proc. 7th Int. Conf. Comput. Knowl. Eng. (ICCKE)*, Oct. 2017, pp. 450–454.

[19] B. S. B. Martins, L. M. B. Oliveira, F. D. Silva, M. D. Leal, P. C. João Porto, G. B. L. Martins, T. S. Santos, M. T. Pires, H. T. Oliveira, and L. W. Guimarães. *Pequi Mecânico—IEEE VSS Soccer Team/CBR 2019*. (2019). [Online]. Available: http://sistemaolimpo.org/midias/uploads/9a13c8e12185eff9806df63c768371c3.pdf

[20] L. M. B. Oliveira, C. C. Costa, G. B. L. Martins, G. A. L. Mattos, A. M. Souza, S. O. Mortosa, A. A. R. Tomé, and F. G. R. Jardim. *Pequi Mecânico INF—IEEE VSS Soccer Team—CBR 2018*. (2018). [Online]. Available: http://sistemaolimpo.org/midias/uploads/aedd8b92ebb8c6bec6d300b66ae210c4.pdf

[21] J. N. de Oliveira Lima Júniorr, P. H. M. Braga, R. D. A. Delgado, V. B. Silva, H. D. F. Bassani, and E. N. D. S. Barros. *Robocin ia Description Paper*. (2019). [Online]. Available: http://sistemaolimpo.org/midias/uploads/9315353ac70c925072cb8aac87a15b19.pdf

[22] A. F. V. Muzio, "Curriculum-based deep reinforcement learning applied to humanoid robots," M.S. thesis, Comput. Sci. Division (IEC), Instituto Tecnológico de Aeronáutica, São José dos Campos, Brazil, 2018.

[23] L. C. Melo, "A deep reinforcement learning method for humanoid kick motion," M.S. thesis, Comput. Sci. Division (IEC), Instituto Tecnológico de Aeronáutica, São José dos Campos, Brazil, 2018.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[25] M. van der Ree and M. Wiering, "Reinforcement learning in the game of othello: Learning against a fixed opponent and learning from self-play," in *Proc. IEEE Symp. Adapt. Dyn. Program. Reinforcement Learn. (ADPRL)*, Apr. 2013, pp. 108–115.

[26] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (Adaptive Computation and Machine Learning Series), 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[27] F. Oliehoek and C. Amato, *A Concise Introduction to Decentralized POMDPs*. Cham, Switzerland: Springer, 2016, pp. 14–15.
[28] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, vol. 48, M. F. B. Kilian and Q. Weinberger, Eds. New York, NY, USA, Jun. 2016, pp. 1928–1937.
[29] P.-W. Chou, D. Maturana, and S. Scherer, "Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, D. P. Yee and W. Teh, Eds. Sydney, NSW, Australia, Aug. 2017, pp. 834–843.
[30] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. 32nd Int. Conf. Mach. Learn.*, vol. 37, F. Bach and D. Blei, Eds. Lille, France, Jul. 2015, pp. 1889–1897.
[31] J. Schulman, P. Moritz, S. Levine, I. M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR)*, Y. Bengio and Y. LeCun, Eds. San Juan, Puerto Rico, May 2016, pp. 1–14.
[32] C. Xuanyu, L. Zhiwei, Y. Yongyi, S. Ping, W. Jiawen, L. Haoran, and F. Kai, "Multi-robot collaboration based on Markov decision process in Robocup3D soccer simulation game," in *Proc. 27th Chin. Control Decis. Conf. (CCDC)*, May 2015, pp. 4345–4349.
[33] (2020). *Rules for the Small Size Soccer League*. [Online]. Available: http://200.145.27.208/cbr/wp-content/uploads/2020/04/sslrules.pdf
[34] (2018). *Rules for the Humanoid League*. [Online]. Available: http://200.145.27.208/cbr/wp-content/uploads/2020/04/RulesLARC2018.pdf
[35] A. Sheikhlar and A. Fakharian, "Adaptive optimal control via reinforcement learning for omni-directional wheeled robots," in *Proc. 4th Int. Conf. Control, Instrum., Autom. (ICCIA)*, Jan. 2016, pp. 208–213.
[36] Q. Zhao, Z. Liang, F. Fang, C. Xia, Z. Huang, and Z. Xu, "Local passing-ball tactics based on a keepaway algorithm," in *Proc. 29th Chin. Control Decis. Conf. (CCDC)*, May 2017, pp. 4884–4889.
[37] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.
[38] B. Peng, T. Rashid, C. S. D. Witt, P.-A. Kamienny, P. Torr, W. Boehmer, and S. Whiteson, "FACMAC: Factored multi-agent centralised policy gradients," in *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds. Red Hook, NY, USA: Curran, 2021.
[39] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: A framework and survey," *J. Mach. Learn. Res.*, vol. 21, pp. 1–50, Jul. 2020.
[40] B. Jordan Pollack and D. Alan Blair, "Coevolution of a backgammon player," in *Proc. 5th Int. Workshop Synth. Simul. Living Syst.*, 1996, pp. 92–98.
[41] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent complexity via multi-agent competition," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–12.
[42] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2012, pp. 5026–5033.
[43] T. H. D. Oliveira, "Fusão de sensores para robô da categoria vsss," Bachelors thesis, School Elect., Mech. Comput. Eng. (EMC), Universidade Federal de Goias, Goiânia, Brazil, 2018. [Online]. Available: https://github.com/thiagohenrique1/tcc_vsss/blob/master/tcc.pdf
[44] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, G. M. Bellemare, A. Graves, M. Riedmiller, K. A. 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, 2015.
[45] B. Efron and R. Tibshirani, "Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy," *Statist. Sci.*, vol. 1, no. 1, pp. 54–75, 1986.
[46] Q. Wang, J. Xiong, L. Han, P. sun, H. Liu, and T. Zhang, "Exponentially weighted imitation learning for batched historical data," in *Advances in Neural Information Processing Systems*, vol. 31, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2018.

BRUNO BRANDÃO received the bachelor's degree in computer engineering and the master's degree in computer science from the Federal University of Goiás (UFG) focused on multiagent reinforcement learning, where he is currently pursuing the Ph.D. degree in researching reinforcement and representation learning. He was experience in developing Machine Learning, and Reinforcement Learning solutions for industry problems. He is currently working as a Reinforcement Learning Researcher at the Artificial Intelligence Center of Excellence (former Deep Learning Brazil).

TELMA WOERLE DE LIMA is currently a General Manager at the Artificial Intelligence Center of Excellence and a Professor of computer science at the Federal University of Goiás. She also works in the artificial intelligence area with a focus on multiobjective optimization with metaheuristics, data structure representations for network problems, reinforcement learning, and pattern recognition.

ANDERSON SOARES is currently a Manager at the Artificial Intelligence Center of Excellence and a Full Research Professor in machine learning and deep learning at the Federal University of Goiás, where he is the Head of Deep Learning Brazil, AI Research Laboratory. He also works are focused in Pattern recognition focused on real-world problems.

LUCKECIANO MELO received the bachelor's degree in computer engineering and the master's degree in electronics and computer engineering from the Instituto Tecnológico de Aeronáutica, in 2018 and 2019, respectively. Current Head of RL Research at the AI Center of Excellence (former Deep Learning Brazil), working in core research problems as well as applications for production systems within industry partners. He also works as an Applied Scientist at Microsoft with Multimodal Representation Learning for Document Understanding. His research interests include develop agents that learn behaviors through interaction, in a efficient, generalist, and adaptive way. Specifically, his primary focus of research and interests is to develop and apply meta-learning and reinforcement learning algorithms for high dimensional control tasks, aiming data efficiency, and reusability of prior knowledge.

MARCOS R. O. A. MAXIMO received the B.Sc. degree (Hons.) *(Summa cum Laude)* in computer engineering and the M.Sc. and Ph.D. degrees in electronic and computer engineering from the Aeronautics Institute of Technology (ITA), Brazil, in 2012, 2015, and 2017, respectively. He is currently a Professor at ITA, where he is a member of the Autonomous Computational Systems Laboratory (LAB-SCA) and leads the Robotics Competition Team: ITAndroids. He is especially interested in humanoid robotics. His research interests include mobile robotics, dynamical systems control, and artificial intelligence.