FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Multi-Robot Learning of High-Level Skills in RoboCup

Pedro Amaro

DISSERTATION

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Prof. Luís Paulo Reis

Second Supervisor: Prof. Armando Jorge Sousa

July 25, 2019

Multi-Robot Learning of High-Level Skills in RoboCup

Pedro Amaro

Mestrado Integrado em Engenharia Informática e Computação

July 25, 2019

Abstract

Solutions for low-level skills in RoboCup are sufficiently developed to allow the development of better high-level skills, and the recent rule changes of the 3D Simulation League emphasize high-level skills. Improving the performance of the high-level skills of the FCPortugal3D team is thus critical to keep it competitive.

We analyzed in depth the literature relevant to this topic, consisting of three main parts. The first part surveys the main topics behind Robot Learning, first introducing Machine Learning and Reinforcement Learning (RL), and then delving into RL algorithms, such as TRPO, and optimization algorithms, such as CMA-ES. Regarding the second part, it explores topics regarding learning when there are multiple robots involved, Multi-Robot Learning, introducing high-level and low-level skills, and the considerations that must be taken when implementing a Multi-Robot Learning system, such as whether the robots communicate or whether they're heterogenous or homogenous. The final part studies the topic of learning in RoboCup, specifically, and the ideas behind the high-level skills of three teams (FCPortugal3D, magmaOffenburg and UT Austin Villa) are studied, along with the low-level skills of FCPortugal3D.

Afterwards, in order to speed up the computation time, we implemented a multi-robot distributed learning framework for high-level skills in the FCPortugal3D team, with possible generalization to other types of skills or teams. The architecture of this system is outlined, including topics such as the web interface of the framework and continuous integration. Furthermore, the performance of the system is evaluated, namely the performance of single core, multithreaded and distributed implementations, with promising results: a 1400% performance increase when comparing the single core and distributed implementation. Also, a comparison between the expected and actual times taken for optimizing high-level skills was made, and these were relatively similar.

Subsequently, we described the 2 high-level skills we implemented, both of which are kick-offs, analyzed the fitness function that was used for optimization and the parameters for each kick-off. The results of these two kick-offs after optimization against the baseline model are impressive, with fitness increases of 55% and 182%. The performance of the FCPortugal3D team was also evaluated in simulated and real world matches against other teams. The performance of the simulated matches was divergent from the optimization performance. We averaged 0.4 goals in real world matches each time the skill was performed. Our approach was successful in implementing an high-level kick-off skill that performed well, and the distributed framework approach is applicable to other 3D Simulation Sub-League teams.

Summary

Simulated robotic soccer is a test bed for trying out ideas and concrete implementations of all topics related to robotics, including basic low-level skills and collaborative actions. With changes to the rules in the 3D Simulation Sub-League to favor collaborative robots, it is important that collaborative skills are further developed. Thus, a multi-robot distributed learning framework has been developed, along with a model an high-level kick-off skill, which will then be optimized and evaluated against other teams. For optimization, state of the art machine learning and multi-robot learning techniques will be used. In the end, the performance of the FCPortugal3D team has been improved, and the effectiveness of the implemented kick-off has been proven. There is still a large body of high-level skills that can be optimized, both in the FCPortugal3D team, and in other 3D Simulation Sub-League teams and there are two main approaches to optimizing them: simplifying the simulation in order to quickly simulate high-level skills, as performed by magmaOffenburg, or distributing the optimization process so that computing resources can be used to speed up the computation.

Acknowledgements

I would like to thank my supervisors Prof. Luís Paulo Reis and Prof. Armando Jorge Sousa for their invaluable contribution to this dissertation.

I would also like to acknowledge my family for the support provided during the dissertation.

Pedro Amaro

"What all of us have to do is to make sure we are using AI in a way that is for the benefit of humanity, not to the detriment of humanity."

Tim Cook

Contents

List of Figures

LIST OF FIGURES

List of Tables

Abbreviations

3D-LIPM	Three-Dimensional Linear Inverted Pendulum Model
CMA-ES	Covariance Matrix Adaptation Evolutionary Strategy
DPRE	Dynamic Positioning and Role Exchange
MAS	Multi-Agent System
ML	Machine Learning
REPS	Relative Entropy Policy Search
rcssserver3d	RoboCup Soccer Simulation Server 3D
RL	Reinforcement Learning
RL	Reinforcement Learning
RoboCup	Robot Soccer World Cup
Robotica 2019	Festival Nacional de Robótica 2019
SBSP	Situation Based Strategic Positioning
SPlanner	Strategy Planner
TRPO	Trust Region Policy Optimization

Chapter 1

Introduction

Learning how to do simple actions, such as walking, or getting up after falling, is a solved problem, even if not perfectly. Skills requiring close collaboration between multiple robots, such as performing cooperative actions in soccer is an open problem. This dissertation introduces how robots learn how to do simple actions, how this can be extended to multi-robot actions, and how they can be applied to simulated robotic soccer, specifically.

1.1 Context

Robot Soccer World Cup (RoboCup) is a landmark project in the area of robotics, that is, building a soccer-playing robot will not have a significant practical impact, but will be an important achivement in the field [KAK+97]. Furthermore, it serves as an important benchmark of cooperative and competitive behavior for multi-robot teams, which can be considered as a Multi-Agent System (MAS).

One of its objectives is to increase the visibility of Artificial Intelligence and Robotics, and it has a challenging stated ultimate goal: "By the middle of the 21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup." [Rob18a].

There are many RoboCup soccer leagues, ranging from the Humanoid League, where a robot as close as possible to a human is used, to the Simulation League, where there is no game being physically played, and all is simulated on a computer.

This work will mainly focus on a sub-league of the Simulation league, the 3D Simulation Sub-League, which currently consists of a team of 11 players playing against another team of 11 players, while simulating all 3 dimensions and complex physics, and using realistic NAO robots.

Many teams compete in the 3D Simulation Sub-League, and develop increasingly complex solutions, resulting in more intelligent behavior year after year. One of these teams is FCPortugal3D, which already includes low-level skills such as walking, running and kicking to achieve its

goals [LRS+13]. More than simple keyframe based approaches, these low-level skills have been implemented in a flexible way. For instance, the kicking has been implemented omnidirectionally, so that the ball can be kicked from multiple positions and multiple angles, removing the necessity of executing an expensive, in terms of time, preparation phase [FRM12].

1.2 Motivation

Initially, the 3D Simulation Sub-League was focused on the low-level behavior of humanoid robots, such as walking, kicking and getting up, but as these behaviors have been increasingly improved, the focus has shifted to higher-level skills, such as pass and shoot. Additionally, the rules of the various RoboCup competitions are changed every year, and with measures such as requiring the use of heterogeneous robots in this subleague, there is a strong indication that strategy and team play will be favored in the future [Rob18b]. Thus, creating and optimizing high-level robot skills will be crucial to obtain good results.

1.3 Objetives

FCPortugal3D's low-level skills are sufficiently developed to allow the implementation of high-level skills [LR14] and, in fact, these skills have been developed, with setplays and multi-robot formations.

This dissertation looks into applying machine learning to the multi-robot domain, in order to learn collaborative high-level skills as building blocks for planning. Thus, a learning model for optimizing the high-level skills of a simulated humanoid team will be built, along with an evaluation of the various learning algorithms and the evaluation against other teams of the newly acquired high-level skills.

Additionally, the solution built will also be able to be used for analogous problems in the field of robotics.

1.4 Work structure

This dissertation is divided into 4 chapters. The next chapter, Learning in Multi-Agent Systems, discusses the state of the art in Robot Learning, Multi-Robot Learning and Learning in RoboCup, starting with foundational knowledge of Machine Learning, and delving into deeper topics such as optimization algorithms, and what aspects should be considered in Multi-Robot Learning, along with a description of how they're applied to learning in RoboCup. After that, we describe our implementation of a distributed optimization framework. The fourth chapter presents the proposed solution, what will be done, along with how it will be done. Finally, the last chapter concludes the dissertation.

Chapter 2

Learning in Multi-Agent Systems

Learning in Multi-Agent Systems encompasses learning individually, as a single agent, then moving onto learning collaborative multi-robot skills, and finally onto RoboCup specific tasks, such as performing kicks, at a first level, and soccer formations and tactics, at a second level.

2.1 Robot Learning

Creating a robot that learns how to perform a task would be a major breakthrough in the field of artificial intelligence, and would require integration of robot perception, planning, learning and action [CM12]. The end result would be that we would tell robots what they should achieve, rather than how to achieve it.

2.1.1 Machine Learning

Machine Learning (ML) allows computers to improve their results on a task, basing their decisions on data, with a wide range of applications, including email filtering and computer vision. There are 3 main approaches to machine learning: supervised learning, unsupervised learning and reinforcement learning [JM15], which are summaried in table 2.1.

Supervised learning assumes that we know a priori the input values and corresponding output, and learning is done by optimizing an objective function. Ideally, this function would be able to correctly classify novel inputs. Included in supervised learning are classification and regression,

Table 2.1: Machine Learning approaches

	Discrete	**Continuous**
Supervised	Classification	Regression
Unsupervised	Clustering	Dimensionality Reduction
Reinforcement Learning (RL)	Discrete RL	Continuous RL

Table 2.2: Comparison of RL algorithms along the discrete/continuous and state/action spaces axes

	Discrete state space	**Continuous state space**
Discrete action space	Q-learning [WD92] SARSA [RN94] Q-learning (λ) [Wat89] SARSA (λ) Monte Carlo [SS96] I2A [WRR+17]	DQN [MKS+15]
Continuous action space		DDPG [LHP+15] A3C [MBM+16] NAF [GLSL16] TRPO [SLM+15] PPO [SWD+17]

the former limits its outputs to a certain range of values, such as a binomial 'yes'/'no' distribution, while the latter has continuous output variables.

Unsupervised learning makes no such assumptions: there is no known correct output variable, and algorithms must make their decisions based on the similarity or dissimilarity of data.

Reinforcement learning is about which actions agents should take in which states to maximize their rewards. An explicit, exact model of the world is not required, and algorithms based on this approach rely on the visible state of the world to decide on the actions to perform. Like unsupervised learning, there are no known output variables, and the agent, by knowing its previous state and the state it arrived at after performing an action and received a reward, must decide on the best policy to obtain rewards. In addition, the agent must have a good balance of exploration and exploitation: although finding the first rewarding state/action pair relies on random chance and complete exploration, the agent must explore other actions, that may be rewarding (or not) on the same state, and only after a few attempts should it decide to exploit the most rewarding state/action pair.

2.1.2 Reinforcement Learning

In regards to reinforcement learning, many algorithms are available, from ones that operate in discrete spaces, such as the well-known Q-learning algorithm [WD92], to others that operate on continuous spaces, and thus have more applicability in robotics, such as Trust Region Policy Optimization (TRPO) [SLM+15]. There are many others, and they can be classified on whether they have a discrete or continuous state space or a discrete or continuous action space, as Table 2.2 shows. Additionally, Table 2.3 shows 2 other possible classifications: on one hand, RL algorithms can be model-free or model-based, that is, if they are able to make predictions about future rewards or not; on the other hand, algorithms can be on-policy, or off-policy.

Table 2.3: Comparison of algorithms along the model-free/model-based and on-policy/off-policy axes

	On-Policy	**Off-Policy**
Model-Free	SARSA [RN94] SARSA (λ) TRPO [SLM+15] PPO [SWD+17]	Q-learning [WD92] Q-learning (λ) [Wat89] DDPG [LHP+15] A3C [MBM+16] NAF [GLSL16] Monte Carlo [SS96] DQN [MKS+15]
Model-Based	I2A [WRR+17]	I2A [WRR+17]

2.1.2.1 Q-learning

Q-learning works by experimentation, according to Equation 2.1: at state s, an agent will try an action a, which will have a consequence, based on the immediate reward given to that action, plus an estimate of any future rewards of the state s_2 that the action brought the agent to. By experimenting with all actions in each state, the best policy for that state can be learned, by choosing the action that maximizes the long-term reward. Its advantages are that it is simple to implement and does not require an explicit model of the environment.

Its parameters are the learning rate α and discount factor γ. α controls how quickly it learns new information (and forgets old information), a value of $\alpha = 1$ is ideal for deterministic environments, while stochastic environments require a lower value for Q-learning to be effective. Regarding γ, it is a cumulative multiplier, in that $\gamma = 1$ means that a reward later is as good as a reward now, while lower values mean that a later reward is worth less and less as time goes on. However, it is only able to handle discrete state and action spaces, and does not scale well with increases in the state-action space.

$$Q_{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \cdot \max_a Q(s_{t+1}, a) \right) \tag{2.1}$$

2.1.2.2 Gradient Descent

The most well-known algorithm for finding the minimum of a function could be said to be Gradient Descent. It has uses in ML and RL, more specifically as an optimization algorithm. It minimizes the function by starting at a certain point, computing the gradient of that point, moving in the opposite direction of the gradient by a small enough step η, and iteratively repeating these steps until convergence is reached. It's guaranteed to converge to at least a local minimum.

In other words, it could be stated that we are following the slope defined by the function until we reach the bottom of a local valley.

2.1.2.3 TRPO

Trust Region Policy Optimization (TRPO) works under the assumption that a small difference in the parameters of the model can result in a big difference in performance.

It uses the same ideas as Gradient Descent to compute the best parameters for a given model. However, as we cannot derive the fitness function, and can only know its value with regards to the values of the parameters, the approach must be slightly different.

As with gradient descent, we start with a certain set of parameters. Then, we compute a small enough trust region within which we will vary these parameters, and pick many sets of parameters within this region. After that, we compute the result of the fitness function for each set of parameters, and based on this, an estimate of the gradient. Then, we move away from the gradient, and converge until the minimum is found. Ideally, this would converge to the best policy, the one that minimizees the fitness function.

2.1.2.4 Deep Reinforcement Learning

Additionally, it is possible to combine reinforcement learning with deep learning techniques, called deep reinforcement learning [FLHI+18]. This generally consists in using a deep neural network as an approximator of the Q network, with the agent's perceptions as inputs to the neural network, and the output being the expected reward for executing each action.

Deep reinforcement learning has been found to be useful in state spaces with a high number of dimensions and also in completing complex problems, with less domain knowledge than other methods. This is due to its level of abstractions: separating lower level features in lower layers from higher level features in higher layers [FLHI+18].

2.1.3 Learning in continuous domains

Although many problems in robotics are discrete in nature, such as identifying the object in front of a robot or which kick to use in the case of a soccer robot, there are countless other problems that are continuous, such as deciding where to walk or where to kick a ball [PG17]. Even deciding which kick to use, although a discrete action, requires an interpretation of a continuous state, as it depends on many factors such as the position of the ball and other players.

Classical learning algorithms such as Q-learning only work in discrete state and action spaces, as shown in Table 2.2, but in robotics, and more specifically in RoboCup, it is generally necessary to handle continuous state and action spaces. One approach is to discretize the continuous spaces, by mapping ranges of continuous variables to a single discrete value, such as mapping every position on the left of the field to -1. However, discretizing with enough detail the positions of the ball and all players in a RoboCup match would lead to a very large increase in the size of the state space, which would make classical learning extremely slow, as it requires going through each state many times in order to learn.

The solution is to use algorithms that specifically target continuous action and state spaces.

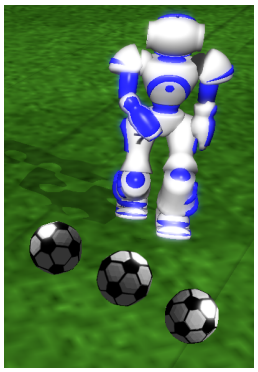

Figure 2.1: Example of contextual learning

2.1.3.1 CMA-ES

Covariance Matrix Adaptation Evolution Strategy is an algorithm like Gradient Descent, which attempts to find the minimum of a function. Although not a RL algorithm, it can be used to find the minimum value for a fitness function, which makes it suitable for learning which parameters result in the best fitness function.

Its functioning is similar on a high level to TRPO: start with many sets of parameters, calculate the fitness function for each of these set of parameters, keep only the best sets, generate new mutated sets based on the values of the best sets, and repeat, starting with calculating the fitness function for the new mutated sets [Han16].

2.1.4 Contextual Learning

In reinforcement learning, we assume the objective is constant: the task to perform does not change. However, this is far from the truth in RoboCup: in learning low-level skills, such as kicking, it may be desirable to kick the ball to many different positions, and from many different starting positions, as shown in Figure 2.1, which requires a different reward function. However, learning all possible kicking tasks is unfeasible, especially in a continuous environment. Thus, to solve this problem, we use contextual learning, or multi-task learning, which consists of assigning a context to each task, such as were we wish to kick the ball, and then learning multiple tasks at the same time, based on that context. Although RL algorithms generally do not take contextual learning into account, it is possible to extend them to include this functionality.

Abdolmaleki [APL+17] has extended the algorithm presented in the previous section, CMA-ES, to support contextual learning.

2.2 Multi-Robot Learning

Up to now, we've discussed learning within the perspective of a single robot, in a static (that does not change, not necessarily immobile) environment. However, in RoboCup, collaboration is a

requirement, and multiple robots are necessarily present in the environment and they might all learn at the same time.

2.2.1 Multi-Agent System

A Multi-Agent System (MAS) is a system of interacting agents, who make autonomous decisions based on the information they receive and cooperate (or compete) to achieve their goals [Woo09].

2.2.2 High-Level Learning

There are 2 main categories of skills in RoboCup, as defined by Stone et al. [SV99]: low-level skills and high-level skills. On one hand, low-level skills consist of the basic soccer robot skills such as walking, kicking, getting up after falling, or even interception of a moving ball, without which no coordination between robots is possible. On the other hand, high-level skills consist of coordination between robots and encompasses everything between passing the ball to a team mate to multi-robot formations [SV98].

For learning this wide variety of skills, many variables must be taken in consideration. For low-level skills, there are low-level parameters such as the position of the ball, the position we wish to move to, or the position of the goal, while for higher level skills there are higher level variables, such as which team mate to pass to, or the current formation.

This concept can be generalized to multiple learning layers [MS18].

2.2.3 Multi-Agent Learning

Multi-agent learning is the application of ML, which has been discussed in the previous section, to MAS. Multi-agent learning builds upon ML by increasing the number of agents, from 1 to N, resulting in the problem of an increase of the search space by a large factor. Another problem is the high-level behavior that emerges from the low-level interactions between groups of agents. Solutions to this problem tend to be based on 2 main ideas: reinforcement learning and stochastic search, but both can use evolutionary strategies [PL05].

There are 2 kinds of multi-agent learning approaches, team learning and concurrent learning [PL05]. The former is considered to be the easiest kind, and uses a single agent, who learns for the entire team, being similar to the classic ML methodology, but has problems when the number of agents is increased. By contrast, the latter uses a team of agents, who are all concurrently learning, which may reduce the state-action space. However, concurrent learning has a key issue, adaptive dynamics: the environment is not static, and agents must learn how to adapt to a changing environment, as the other agents are also learning and changing their policies.

In regards to how the agents are organized together, there are 3 kinds of teams: homogenous teams, heterogenous teams, and hybrid teams [PL05]. Homogenous teams consist of a single behavior encoded into each agent, and are applicable to problems where the number of agents is very large, by reducing the search space. With that said, agent specialization is not possible, as all agent will act the same, which is crucial in RoboCup. Heterogenous teams consist of a different

behavior encoded into each agent, allowing specialization, such as a soccer robot behaving as the goal keeper and another as a defender, at the cost of increasing the state-action space. The last kind of team, is a generalization of the previous two: a team has S squads and each squad has S_n members, and all members of a squad share the same behavior. For $S = 1$, the team is homogenous, for $S_n = 1$ the team is heterogenous and for values in between, the team is a hybrid team. This hybrid generalization can be used to acquire the advantages of both homogenous and heterogenous teams.

Another problem is the credit assignment problem: which agents should the reward go to [PL05]? The most natural solution is to assign the same reward to all agents, effectively dividing the reward between all. However, this is non-ideal, as an agent could have performed a less than ideal action that would ideally be punished, and received a reward instead, amplifying the negative actions. Another solution, albeit more difficult to design, is to give a higher percentage of the reward to the agent who performed the bulk of the work and give no reward or even a punishment to agents who did not perform meaningful work.

Adding onto that, we come onto the issue of modelling the behavior of team mates [PL05]. For an agent to perform optimally, it must know the future behavior of their team mates. If both agents are aware of the future behavior of each other, they are also aware that the other agent is aware of them, resulting in an infinite recursion. This must be solved within the time constraints of RoboCup, as effectively as possible. One approach is to consider N-level agents, where N represents the number of awareness relations between the agents: a 0-level agent would only be aware of itself, a 1-level agent would be aware of the behavior of other agents, and a 2-level agent would be aware of the awareness of other agents of themselves.

Furthermore, communication is considered to never result in poorer performance [PL05]: if communication would result in lower performance, agents would learn to ignore it, resulting in no loss from communication. From this, there are 2 kinds of communication: direct communication and indirect communication. The former consists of sending messages directly between agents, in a external system disconnected from the environment, such as by using a shared blackboard and message-passing. The latter consists of using the interactions with the environment to send messages, and is related to how we communicate in real life with hand gestures, for instance. In theory, robots in RoboCup could communicate in the same way, by using arm movements to communicate their intentions. In practice, however, they use a direct means of communication, since RoboCup's 3D Simulation Sub-League allows sending messages.

Finally, there are 3 main open problems to be solve in the field of multi-agent learning: scalability, adaptive dynamics and problem decomposition [PL05]. Regarding the first, scalability, the size of the state-action space increases quickly with an increase in the number of agent or of the complexity of their behaviors. Adaptive dynamics is related to what was discussed earlier: unlike in traditional RL, the environment is not static: while our agent learns, the other agents also learn other policies, which might not be what our agent is expecting. The last issue is problem decomposition, which consists of how to divide the complex behaviors of each agents in a way that allows learning, as learning all behaviors at the same time would converge very slowly to a

result.

2.2.4 Collaborative actions

Collaboration between robots is said to have many advantages, including increases in performance over using a single robot, the fact that there are multiple robots in multiple locations, permitting simultaneous actions. Other advantages include a lower cost and better overall system properties. Another advantage is an increase in fault tolerance, if more than one robot can perform a function, such as being a defender in RoboCup, one can be replaced by another in case it falls down [YJC13].

Communication, whether explicit or implicit, is essential for coordination, so that robots can share what they know with others. There are 2 kinds of coordination: static and dynamic. The former is found throughout life in rules such as right-of-way at intersections: it consists of a fixed rule which robots follow and which results in coordination. The latter has 2 other types of dynamic coordination: explicit, which consists of communicating directly with other robots and forming a plan, and implicitly, by communicating via the environment what we wish to do, and the plan being an emergent behavior of the system [YJC13].

One application of collaborative actions in RoboCup is setplays, as defined by the FCPortugal3D and CAMBADA teams for general use by other teams in RoboCup [Mot07] [MLR10] [MRL11]. This allows defining when and how a specific plan will be executed, such as when executing free-kicks near the opposing goal. In this framework, robots have roles such as receiver or initiator, and act according to them, which means that robots could be heterogenous and be able to cooperate, by following their function in the setplay. Furthermore, it is suggested that the robot in charge of the setplay should be the one in possession of the ball, as it is the one who is in control of the play. Finally, it may be possible to acquire new setplays from successful plays in previous RoboCup games.

2.3 Learning in RoboCup

All teams in RoboCup must develop at least a set of low-level skills that allow players to effectively move to target positions, control the position of the ball, and escape unwanted conditions such as falling. The first can be accomplished by walking, the action of a player moving itself to a target position, possibly avoiding obstacles along the way, such as other players or even the ball. In the specific case of the 3D Simulation Sub-League, biped locomation, a popular approach is using the Three-Dimensional Linear Inverted Pendulum Model (3D-LIPM). For controlling the position of the ball, a player can kick it, which is often implemented by using key framed based approaches. To escape unwanted situations, such as when the player falls, a get up behavior can be implemented, using the same technique used for the kicks: one set of key frames for getting up when a player falls forward, and another set for the instance in which a player falls backwards.

In contrast, teams also require high-level skills, which make use of low-level skills to achieve high-level goals, such as ensuring ball possession, advantageous positioning and, ultimately, scoring goals for the team. This can be achieved in numerous ways: we can employ real soccer

concepts, such as tactics and formations or we could optimize specific setplays, such as a kick-off at the beginning of the game.

Although there are many teams participating in RoboCup 3D Simulation Sub-League every year, we will focus our analysis on 3 teams that have been in development for many years, with sufficiently developed low-level and high-level skills. We will perform an in-depth analysis of both low-level and high-level skills of the FCPortugal3D team, and of the high-level skills of magmaOffenburg and UT Austin Villa.

2.3.1 FCPortugal3D

Starting with the FCPortugal3D team, many low-level skills have been, and continue being, developed: walking, running, many different kinds of kicks and getting up. Additionally, there are high-level skills allowing decisions and coordination, such as flexible tactics, based on the Situation Based Strategic Positioning (SBSP) algorithm [LRS+13], and the Dynamic Positioning and Role Exchange (DPRE) system, which allows players to change roles dynamically as the match progresses.

2.3.1.1 Low-level Skills

In terms of walking, this team has many types available for use, which are implemented using the skills package. The first type is an omnidirectional walk [LRS+13, ALR+16], which uses a model based on the 3D-LIPM model to perform this skill, which is subdivided is various modules, such as various trajectory generator modules (which plan the trajectories of the movements that the robot will perform), a Foot Planner modules, which decides where to place the foot in order to achieve a constant center of mass and an Active Balance module, which attempts to guarantee that the robot will not fall while performing the biped walk. The second type of walking that was investigated uses truncated Fourier series formulations, optimized by a genetic algorithm, for instance [RLR+13, YCPZ06]. This allows varying the step length, the frequency of steps and the walking pattern as required.

Regarding running, the first walking model can be extended into a running model by not considering the center of mass as a fixed constant [LRS+13]. While enabling running, this also allows robots to walk more efficiently. Using an evolutionary optimization algorithm, an omnidirectional walking and running skill was developed. Furthermore, a much improved running skill that is around three times as fast as the current state of the art in running, which was shown in the Robotica 2019 festival, is being developed and integrated into the team's overall high-level strategy.

In terms of kicking, there have been many kicking models developed:

- Front Kick: a simple key frame based kick that can achieve a distance of around 6 meters when maximizing for distance, and 5 meters when robustness is preferred [RLR+13]. It requires a long setup time, to place the robot in position to execute the key frames, and is limited to a specific distance. Initially, a baseline model was established, which was then optimized to the final result.

- Omnidirectional Kick: an improved kick from a simpler static, key frame based, kick [LRS+13, FRM12]. This excludes the setup phase for the simpler kick, and instead calculates how the robot should perform the kick in order to efficiently kick the ball to the desired location. Consequently, kicks can be performed faster, and in any direction. It consists of 3 main modules: an Inverse Kinematics module, which calculates the value for each joint, a Path Planning module, which handles the trajectory, and a Stability module, for stabilizing the robot. It has been shown in our experiments to be able to reliably perform kicks in a few key directions. Finally, this model has 8 parameters, which can be tuned manually for good results, but for the best results should be optimized automatically.

- Controlled Kick: a kick that is optimized for flexibility, being able to kick the ball successfully to a variety of distances in a variety of different conditions, such as the robot's pose or the position and angle of the robot relative to the ball [ASL+16]. Involves contextual policy search, and generalizes the kick for multiple distances. Although slower than the kick presented above, and requiring a setup time, it is able to successfully kick the ball to a target location very accurately. It includes a Policy Function, responsible for finding the optimal parameters θ, and a Kick Controller, responsible for controlling the joints of the robot to achieve the kick indicated by the parameters of the Policy Function.

- Long-Distance Kick: a kick that is optimized for long distances. Its main use is in scoring goals from a distance, or kicking a ball to the opponent's side of the field from our field. It requires a relatively long setup time, and is only somewhat accurate.

Likewise, the getting up skill was solved for 2 distinct instances: when the player falls forward and when it falls backwards. A simple key frame based model was manually tuned, which took 2 seconds for the forward get up, and 1.1 seconds after optimizing, and achieved good results for the backwards getting up [RLR+13]. Furthermore, like with running, a better model is being developed which improves the current state of the key frame based get up for the backwards instance.

2.3.1.2 High-Level Skills

The high-level skills that are part of FCPortugal3D were conceptualized much earlier, when the 3D Simulation Sub-League did not exist, and there was just the 2D Simulation Sub-League, beginning in the FC Portugal team [RL01b]. At this time, the simulation was much simpler than the 3D Simulation Sub-League, consisting of circular robots, that could move and kick the ball, with direct, non-jointed, movement [LR07]. This permitted research efforts to be more focused on the high-level skills of the teams rather than the low-level skills, although some optimization of the low-level skills was possible. The FC Portugal team was a winning team: in the 2001 European and World championships it scored 86 and 94 goals, respectively, conceding none.

One of the three key cornerstones of the strategy used in the precursor to FCPortugal3D was the player type: although players were homogeneous and possessed the same abilities, each player

was assigned a specific role, so that certain players would play more aggressively and others more defensively. This included the overall positioning of the players, strategic behaviors, and actions to take when the team is in possession of the ball, ball possession behaviors, or wishes to recover it from the opponent, ball recovery behaviors. The player type system was set up so that players switched roles dynamically in real time as the game's situation deemed it fit, using the DPRE system.

Another cornerstone are the tactics, composed of formations, which allow the overall team strategy to adapt to certain game conditions, requiring a variety of player types. These are used for many game situations, such as attacking, defense and transitions, where each player is assigned an ideal position and their role. The relationship between player types, tactics, behaviors, formations and situations is illustrated in Figure 2.2.

The last cornerstone is the SBSP system [RLO00]: in the FC Portugal team there were 2 main types of situations: a strategic situation, dictated by the strategic behaviors, and an active situation, dictated by the ball possession and ball recovery behaviors. This distinction allows players to purposefully position themselves in advantageous ways the action to take is not clear, and to quickly enter a purposeful action when the players are in possession of the ball or wish to recover the ball from the opponents. In order for the strategic behavior to calculate the target strategic position of the player many factors are taken in consideration, such as the tactic and formation in use and the role of the player. This position is then adjusted depending on many other factors including the ball position and velocity, situation and the positions of other players and their player types. The end result is that the team is capable of occupying the field evenly, covering the most important positions, and positioning each player of the team in an advantageous state for more active states.

Another topic with implications to the 2D and 3D Simulation Sub-Leagues, and wider implications to RoboCup, Robotics and Artificial Intelligence in general, is coaching: an agent can gather information about its opponents, namely game statistics, such as ball possession, and model the opponents' behaviors [RL01a]. Consequently, the team can exploit the weaknesses in the strategy of the other team, such as regions of the field that are not covered by opponents. This allows the FC Portugal team to be more flexible in how it handles its high-level strategy, by not having a fixed ruleset that is followed every time, but one that is adapted to each opponent team. A general coaching language - Coach Unilang was developed to enable coaching for the FC Portugal team, and possibly other teams.

Furthermore, interpretability is crucial to understanding what happens in a high-level strategy: without understanding the perceptions and actions that each player takes in response to the opponent, it is impossible to fully understand how our players react to the opponent. [LRC07, LR07]. For this, the FC Portugal team has developed many ways of viewing debug data, such as a visual debugger [RL01b], which allows debugging the textual log files that are output by the simulator in a visual way, enhancing the conclusions that can be taken from the often too detailed logs. Additionally, it is possible for agents to draw geometric shapes onto the field, which the visual debugger can show, aiding in understanding what the agent is seeing and thinking. Another tool

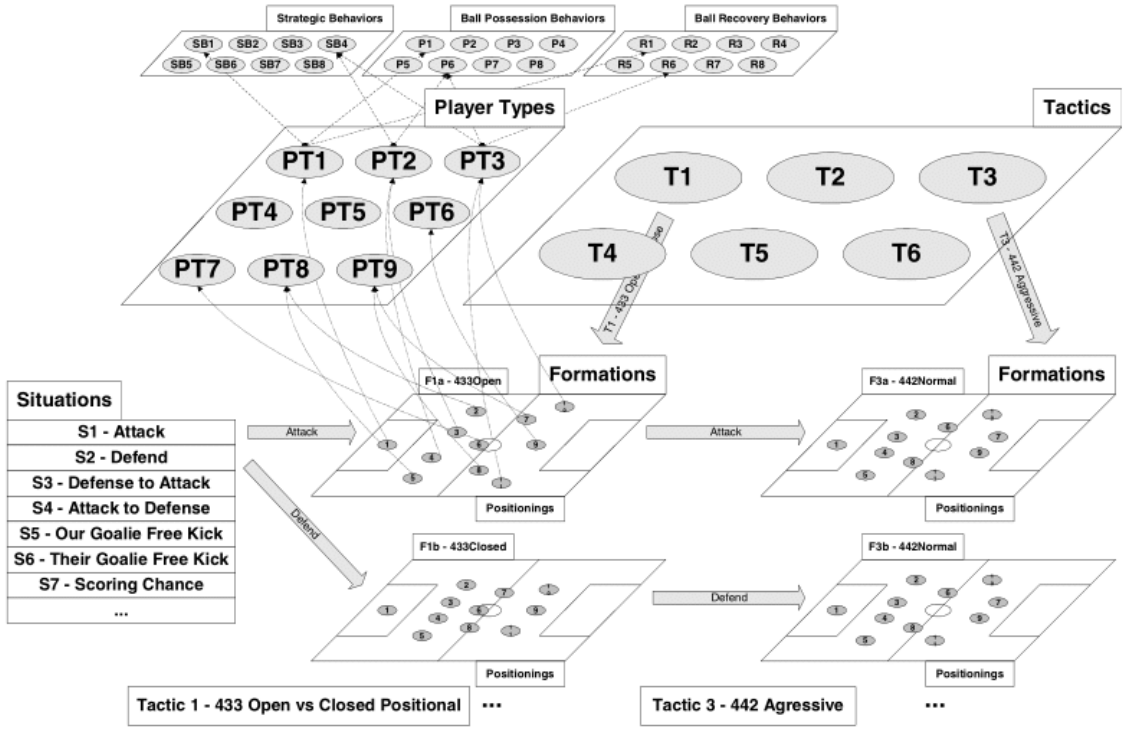

Figure 2.2: Overview of the high-level skills used by the 2D Simulation Sub-League's FC Portugal team [RL01b].

that was developed was an offline client [RL01b], which allows stepping through the logic of the team in a debugger, such as gdb, and viewing at a very low-level how variables in the code change and which functions are executed. This allows understanding exactly what happened and which conditional branches were taken at any level, both low-level and high-level of a simulated soccer team.

To conclude the beginnings of the implemented high-level skills that were introduced in the 2D Simulation Sub-League, there is communication, using the ADVCOM system, which enables agents to communicate the most important information between themselves effectively [RL01b]. The communication in the Simulation League in general is single channel, low bandwidth and unreliable, which requires a suitable and efficient method for handling communication between robots. The ADVCOM system ensures that the world state of each agent is kept up to date, even for objects that might not be in their field of vision, and improves the communication of certain events completed by the team, such as ball passes or switching player types. In order to maintain the aforementioned constraints, the system communicates the complete world state and high-level events when an agent determines that it is in the best position to send this information. The best position is determined by the importance of the information of the agent when compared with the perceived importance of the information of the other agents. When our information is perceived to be the most important one, it is communicated to others. Afterwards, there is a delay before repeating the information, to avoid sending similar information multiple times.

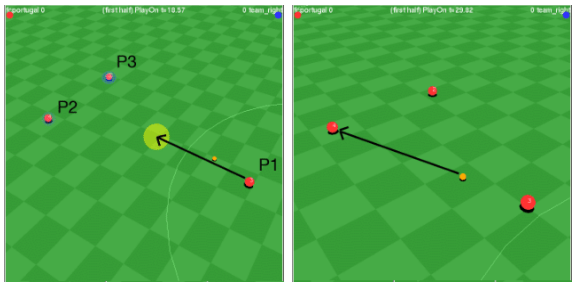

Figure 2.3: Example of the older 3D Simulation Sub-League [Mot07].

In previous years, the 3D Simulation Sub-League was also much simpler, as shown in Figure 2.3: each player consisted of a simple sphere, and they could move around the field also in a very direct way, which again gives more relevance to the high-level skills of a team, rather than the low-level ones. It is in this context that the team began working on setplays, a coordinated previously defined plan, for use in any general soccer team in robotic soccer, both simulated and with real robots [Mot07]. A language to define a general setplay was created, with concepts such as the players that take part in the setplay, whether they be specific players or players with certain roles. A setplay consists of steps, small actions, such as a pass or a kick to the goal, that may be performed by a single player, depending on the conditions specified by the user. Between each step there is a transition, which may be triggered by user-defined conditions. To end the setplay, there is an abort transition, which aborts the setplay early, if it is thought that the setplay will not be able to be completed, or a finish transition, which signals that the setplay was completed successfully.

These setplays may be defined directly in the domain specific language, which must be understood by the user, which is time consuming. A graphical solution, Strategy Planner (SPlanner) [CAA+14], which can be seen in Figure 2.4, was developed to aid users in constructing setplays by intuitively using a graphical user interface to define the intended behavior. This can be achieved easily by using the mouse and keyboard and filling the required fields, such as the type of the setplay and what triggers it. Additionally, SPlanner allows viewing a graph of the steps and transitions that are part of a setplay, as seen in Figure 2.5, enhancing comprehension of more complex setplays with many transitions or steps. The tool also performs basic logical checks that the bare language cannot perform, such as ensuring that there is a limit of eleven players, and that the player positions are within the boundaries of the field.

The aforementioned high-level skills were included into the current 3D Simulation Sub-League, albeit with the necessary modifications that are necessary for them to work with humanoid robots and the different low-level skills.

2.3.2 magmaOffenburg

In general, the magmaOffenburg team has excellent tooling, including a simplified 2D simulator that estimates the movement speed of agents to quickly test out new strategies [BDF+], but also magmaDeveloper, a tool that permits debugging the behavior of the agents [DG16].

Figure 2.4: Main screen of the SPlanner tool [CAA+14].

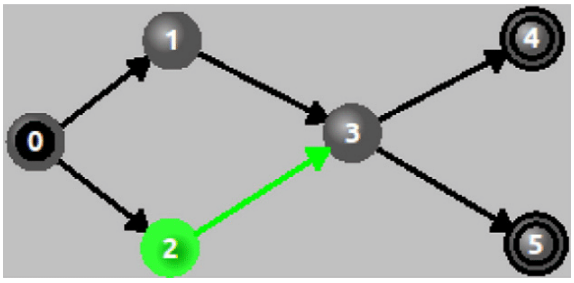

Figure 2.5: Setplay graph generated by SPlanner [CAA+14].

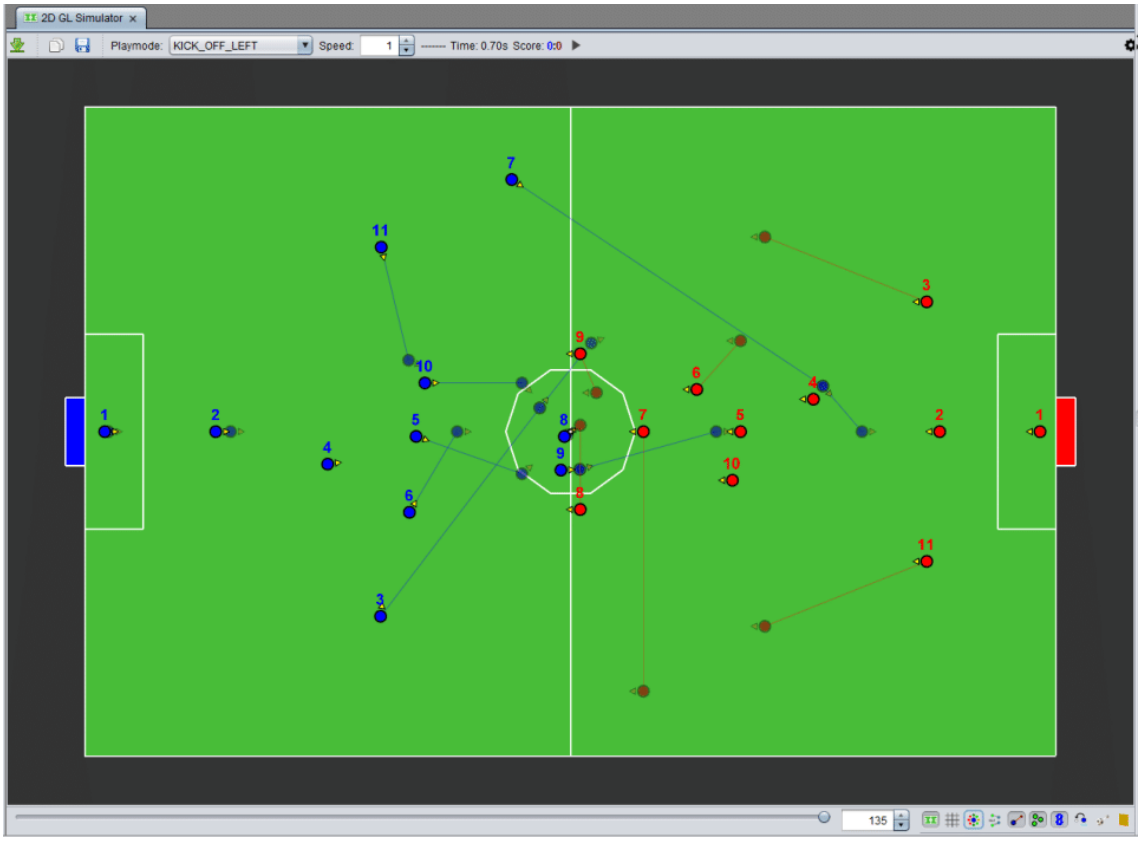

Figure 2.6: Main view of the magmaOffenburg simulator [BDF+].

The magmaOffenburg team tackled a critical question in its early years: why develop a new simulator if there already is a 3D simulator used by all 3D Simulation Sub-League teams [BDF+]? The two main justifications for this were that the 3D simulator is quite slow, requiring lots of CPU time for simulating all the individual joints of each agent, and that it is non-deterministic, causing difficulties in reproducing specific kinds of bugs, such as ones that only occur in rare instances. This simulator, as shown in Figure 2.6, includes all the benefits that are part of the standard 3D simulator with RoboViz, such as dragging and dropping the ball or players, and drawing geometrical shapes into the soccer field. Additionally, since the agents and the simulator are running in the same process, it is possible to debug the code that is being executed in real time, permitting live debugging. Other features are the possibility of controlling the speed of the simulation at any time, and being able to change the code being run on the agent, without restarting the agent nor the simulator.

The simulator is intended to mimic the high-level behavior of the 3D simulator, even if not perfectly, in order to allow testing new strategies very quickly, without considering the already implemented low-level behaviors. This allows abstracting away the notion of the low-level behaviors and only computing the necessary high-level calculations, speeding up the time taken to run simulations. Low-level behaviors are simulated: for walking, there is a WalkEstimator components that measures how quickly an agent walks across the field, and for kicking, each low-level kick has an associated probability distribution in terms of kick distance. In their results, they note a speed up of 40 times in the command line simulator, from 22 hours to simulate 100 games in the 3D simulator, to around half an hour to simulate the same amount of games in the 2D simulator. Although their idea is promising, they also note that the results differ a lot from the 3D simulator, and more work must be put into ensuring that both simulators behave identically.

In terms of high-level skills, they include strategy definitions that define the position of their agents based in the match situation and the role of the player. This is a similar idea to the SBPS system used by the FCPortugal3D team: when players are not in an active situation, they position themselves in the most advantageous locations. A graphical interface for defining and simulating these strategies was included in their developer tooling.

The magmaOffenburg team found that many non-integrated tools were developed over the years and thus opted to integrate them all into a single, full integrated, developer tool, named magmaDeveloper. This tool allows defining other tools as plugins, which can be used with the team, providing an easily extendable interface, aiding in the creation and integration of new tools. One example of an integrated tool is the Debugger plugin, which allows, as the name implies, debugging a specific agent. Another plugin is the Strategy Developer plugin which can be seen in Figure 2.7, and, as mentioned before, allows defining new strategies.

2.3.3 UT Austin Villa

The UT Austin Villa team is a winning team, scoring 136 goals in RoboCup 2011, and 52 goals in RoboCup 2014, while conceding none in both instances [MUB+12, MDS15]. Although they have well performing low-level skills, including an omnidirectional walk engine and its optimization

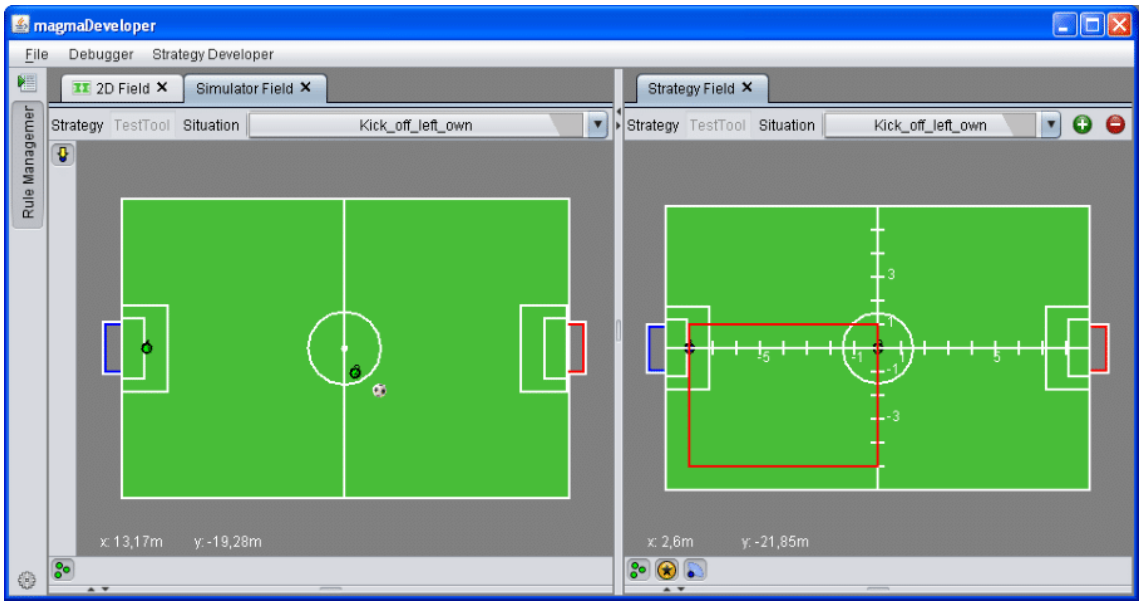

Figure 2.7: Strategy Developer plugin in magmaDeveloper [BDF+].

framework, and an inverse kinematics kicking engine, our main focus in this subsection are their high-level skills: the dynamic role assignment and positioning system [MBS12].

The UT Austin Villa team divides the high-level strategy in 3 main steps:

1. Define the formation to choose based on the position of the ball.

2. For each player, assign its role depending on its position in the field in relation to the formation selected in 1.

3. Communicate amongst all players to select the resulting role assignments.

Step 2 is a non-trivial problem, since players cannot be fixed to certain roles, as their positions will vary during the game, thus a more dynamic assignment is required. This involves defining a role assignment function which defines a role assignment that takes the least time to put the players in the positions defined by the formation, and that does not change abruptly as long as the formation does not change. One possible algorithm for the role assignment involves the use of dynamic programming, as the naive method would be too costly in terms of time.

The search space for learning skills can be divided into layers, some of them interacting directly with the environment, such as walking skills, and other, higher-level skills, making use of these low-level skills. The most usual way to learn skills A and B, in which B depends on A, is learning skill A, freezing its parameters, then learning skill B, called Sequential Layered Learning. This might not be optimal, since the optimized skill A might not be in the best state to execute skill B. Another option is first learning skill A, then learning the parameters of skill A and B at the same time, called Concurrent Layered Learning. However, this greatly increases the search space, inducing long computation times, which might be unacceptable.

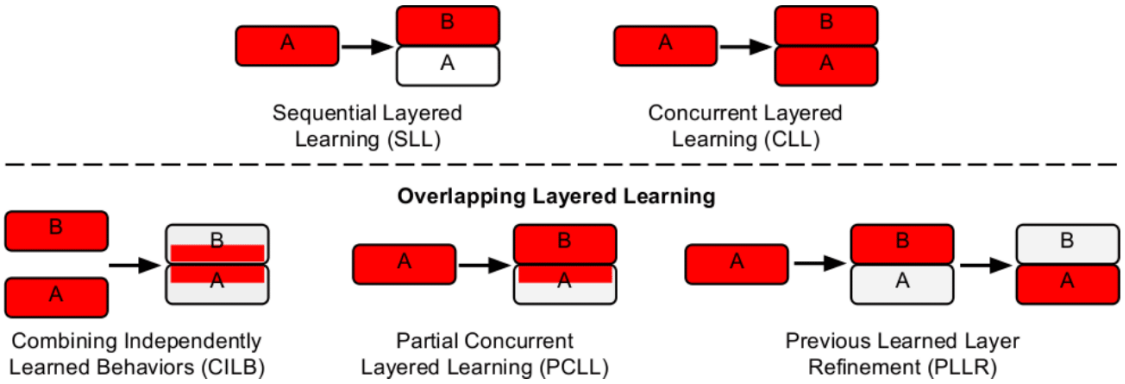

Figure 2.8: Overlapped Layered Learning [MS18].

One possible solution is Overlapping Layered Learning [MS18], as shown in Figure 2.8, which proposes learning skills independently, and then learning only the parts that influence the dependent skill. There are three possible ways to perform learning in an overlapping layered fashion:

1. Combining Independently Learned Behaviors: if A and B are not dependent on each other, they can be learned independently, and then only the parameters relating to the transition between them can be optimized. For instance, a dribbling skill, consisting of moving the ball along with the player, can be combined with a kicking skill to quickly move the ball to a certain position and kick it. These skills can be learned independently and then the transition between the dribble and the kick can be learned, with as few parameters as necessary.

2. Partial Concurrent Layered Learning: assuming B is dependent on A, A can be learned first, and then, instead of making A fully frozen, we can partially freeze it, allowing the relevant parameters to be learned while concurrently learning B.

3. Previous Learned Layer Refinement: assuming B is dependent on A, we can learn A first, then freeze A and learn B, and after that relearn A while freezing the parameters of B.

2.4 Conclusions

There are a lot of ML methods, RL algorithms and parameters of those same algorithms, from the simplistic Q-learning algorithm to the more complex TRPO algorithm. Choosing the proper set of algorithms and parameters is no small task, and requires careful performance evaluations. Additionally, multi-robot learning has many key factors to take in consideration when designing a high-level skills engine, from handling the large state-action space to team mate behavior modeling.

There is a large amount of work done specifically in RoboCup, with various teams contributing to the state of the art. We analyzed the contributions of the FCPortugal3D team in both the low-level skill and high-level skill domains, with special focus given to the latter, including topics such as strategies, SBSP, DPRE, coaching, interpretability and debugging, inter-robot communication,

setplays and graphical definition of setplays. In addition, we looked at the high-level skills and support tools of the magmaOffenburg and UT Austin Villa teams. Having an exhaustive list of potential methods to use is essential for choosing which ones are most suitable for the problem.

Chapter 3

Distributed Optimization Framework

After introducing the main concepts behind learning in RoboCup, it is important to note that optimizing skills takes time, and high-level skills doubly so, since they have more agents to simulate. The naive approach is running a single simulation at a time, which would take weeks to perform any useful optimization. In this chapter, we examine how we can speed up the optimization process by taking advantage of multiple networked computers, in order to parallelize the simulation tasks.

3.1 Motivation

A framework was developed for optimizing low-level behaviors and high-level strategies, using arbitrary fitness-based optimizers, and adaptable to any RoboCup team. This was necessary because of the long optimization times involved: for instance, calculating a single sample of a kick-off (involving 90 trials), as explained in the next chapter, took around 5 minutes, and tens of thousands of samples would have to be calculated for proper optimization. We have focused on the CMA-ES [Han16] algorithm and the FCPortugal3D [RLR+13] team. The framework supports distributed computing, to speed up the optimization process, and scales linearly with computational power. In other words, the time taken to evaluate an entire generation of samples is inversely proportional to the amount of workers available. This permits, in the extreme case, evaluating a CMA-ES sample in the time it takes to evaluate a single trial, assuming the number of workers is large enough.

In order to optimize behaviors or strategies, fitness functions must be defined for both. These are problem dependent, and take into consideration factors relevant to the scenario, such as the time it takes an agent to stand up, or whether a goal was scored with a given formation.

Due to a noisy environment, the fitness of a given sample must be averaged over a number of trials n. Depending on the value of n, we can trade-off between accuracy of our evaluation, which increases with the amount of trials, and the speed at which each sample is evaluated.

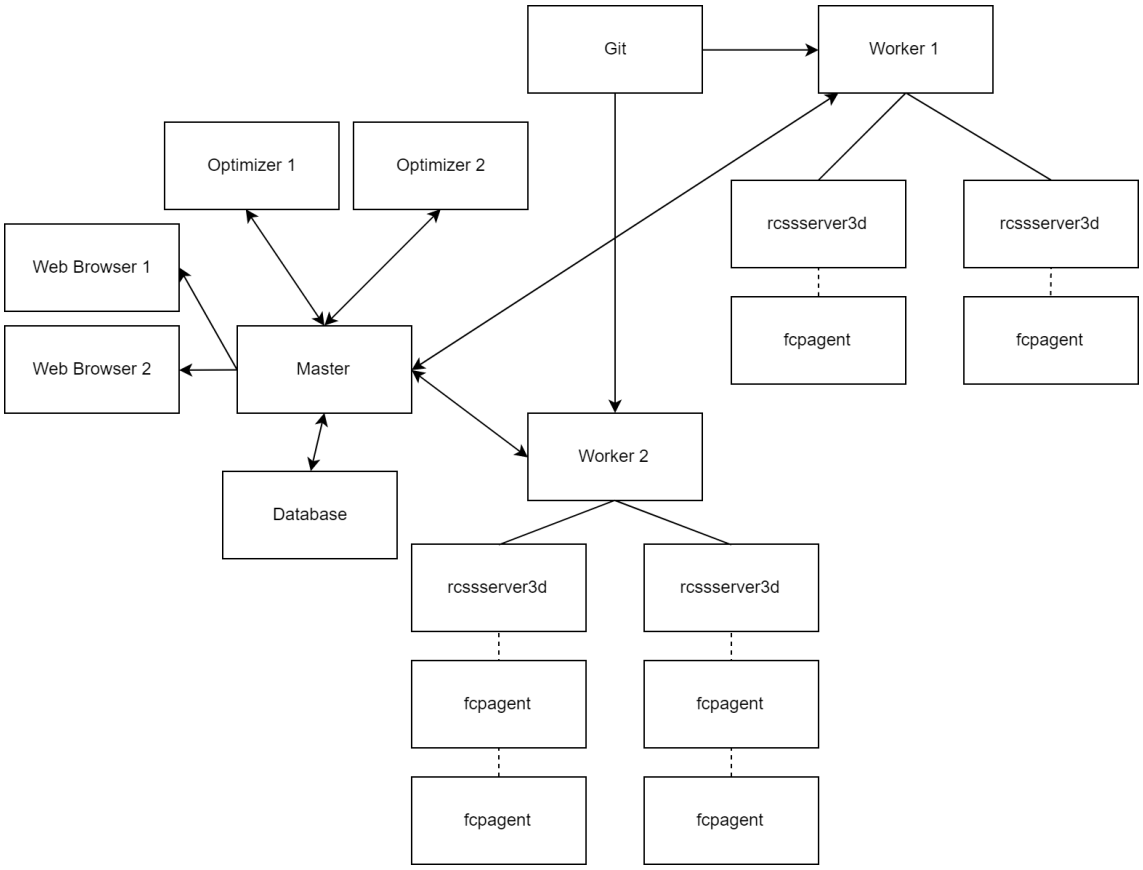

Figure 3.1: Example diagram of a possible configuration of the distributed framework.

3.2 Architecture

The architecture of the distributed system is divided into 3 main components: a single, or more optimizers running concurrently, a master, and various workers. The optimizer component is responsible for running the optimization algorithm, such as, but not necessarily, CMA-ES, and gives an instance of the model that is being optimized to the master. The master then forwards it to the workers, who run simulations in order to determine the fitness function for a given trial. This continues until n trials are done and the fitness of the sample is sufficiently averaged. Having multiple workers not only speeds up computation times, but also results in redundancy: workers can fail at any time, and the framework will seamlessly continue the work, without loss of progress.

An example of a possible configuration of optimizers, master and workers is shown in Figure 3.1, where lines with arrows represent the flow of useful information, lines without arrows represent processes and dashed lines represent both running processes and a TCP connection between the agent and rcssserver3d (RoboCup Soccer Simulation Server 3D). The full details of each component are further explained in this section.

The framework was improved iteratively, and a total of 3 methods were tested. The first is a trivial centralized implementation, which we will refer henceforth as single core, where a single worker evaluates each sample sequentially. This results in low computer utilization and slow

```
<?xml version="1.0"?>
<SetplayDeclaration>
  <Name>Pass and Kick</Name>
  <Parameters>
    <Parameter Name="PositionPasser" Count="2"/>
    <Parameter Name="PositionKicker" Count="2"/>
    <Parameter Name="PositionBallPass" Count="2"/>
    <Parameter Name="PositionBallKick" Count="2"/>
    <Parameter Name="KickParametersPasser" Count="8"/>
    <Parameter Name="KickParametersKicker" Count="8"/>
  </Parameters>
  <Reward>Goal</Reward>
  <RewardParameters>
    <RewardParameter Name="SelfGoals" Value="-1"/>
    <RewardParameter Name="Goals" Value="1"/>
    <RewardParameter Name="TimeStartLoss" Value="10"/>
    <RewardParameter Name="LossPerSecond" Value="0.05"/>
    <RewardParameter Name="MaxZMultiplier" Value="1"/>
  </RewardParameters>
  <Players>
    <FCPortugal3D Number="11" Type="4"/>
    <FCPortugal3D Number="6" Type="2"/>
    <MagmaOffenburg Number="1"/>
    <MagmaOffenburg Number="2"/>
  </Players>
  <Optimization SyncMode="true" RealTime="false" SoccerRules="true" Command="(
      kickOff Left)" MaxTime="30"/>
</SetplayDeclaration>
```

Listing 3.1: An example declaration.

computation times. The second is a local distributed implementation, named multithreaded, where a single worker runs several instances of the simulation software. This solution takes advantage of the many cores a modern computer has available, and speeds up the convergence process. The third solution is a distributed one, over the network: an optimizer generates samples and requests other workers to evaluate them. This allows the framework to scale horizontally, and is the fastest method.

In the framework there are two types of optimization information: declarations and definitions, and they're both defined in XML files. Regarding the former, declarations, as exemplified in Listing 3.1, are metadata about the parameters of an optimization process, defining a configuration for the simulations to be performed. They include data such as the names and number of parameters, the reward function and parameters for the reward function to be used, the players to be used in the simulation, rcssserver3d configuration information, commands to execute and the maximum allowed time for the simulation to run. In regards to the latter, definitions, as exemplified in Listing 3.2 are an instance of a declaration, comprising all the parameters of an instanced model.

Communication between the three main components is achieved using HTTP messages with JSON payloads: master exposes a web API which is hosted on a domain, and both optimizers

```
<?xml version="1.0"?>
<SetplayDefinition>
  <Name>Pass and Kick</Name>
  <Parameters>
    <Parameter Name="PositionPasser">
      <Value>-0.6573746531115268</Value>
      <Value>-0.216089926892083</Value>
    </Parameter>
    <Parameter Name="PositionKicker">
      <Value>-1.8025015782342935</Value>
      <Value>-0.5381189784440512</Value>
    </Parameter>
    <Parameter Name="PositionBallPass">
      <Value>1.6921101129979252</Value>
      <Value>-0.49332295187032854</Value>
    </Parameter>
    <Parameter Name="PositionBallKick">
      <Value>-1.5651529985751247</Value>
      <Value>0.11737443279825248</Value>
    </Parameter>
    <Parameter Name="KickParametersPasser">
      <Value>-0.7861737813317902</Value>
      <Value>-1.5623204347995028</Value>
      <Value>0.1637967344608503</Value>
      <Value>0.44901990488772225</Value>
      <Value>-0.27592294829775343</Value>
      <Value>-0.6085872680793574</Value>
      <Value>0.3336864073461102</Value>
      <Value>1.2021622951384716</Value>
    </Parameter>
    <Parameter Name="KickParametersKicker">
      <Value>0.5154665409086537</Value>
      <Value>-1.8477995282685895</Value>
      <Value>0.6842266574821053</Value>
      <Value>-0.23976978539010443</Value>
      <Value>-0.284719476619423</Value>
      <Value>1.163877221869012</Value>
      <Value>0.44139181060268073</Value>
      <Value>1.9920030203961043</Value>
    </Parameter>
  </Parameters>
</SetplayDefinition>
```

Listing 3.2: An example definition.

and workers communicate with it that way. A database is used to store information regarding the optimization process, including the fitness value of every single reward. There are two kinds of endpoints: the ones specific to optimizers, and the ones specific to workers. The former are:

- /optimize/work: submits a new sample to be calculated.
- /optimize/stop: stops a sample from being calculated (in case the optimization process is halted.
- /optimize/results: polls for the results of previously submitted samples.

The endpoints specific to workers are:

- /compute/get: acquires a new sample to calculate.
- /compute/post: posts the result of a single trials.

Additionally, the master exposes a web statistics interface, which can be used to view statistics about the optimization process conveniently, on any device, such as mobile phones. For example, this allows users to know when the optimization process has found new parameters that are worth investigating, without forcing the user to stay near their computer. There are 4 kinds of distinct information presented:

- Performance of the optimization process, as seen in Figure 3.2: gives a general indication of the overall performance of the system, showing trials done by each worker, which are differentiated by colors. Each column corresponds to a timeframe of 10 minutes, and each pixel corresponds to 10 trials performed. It is not intended to be accurate, but give an overview of how the framework is doing. For instance, in the case of the shown figure, there was a problem with the blue worker, which suddenly began performing badly, as seen in the middle of the graph. The issue was quickly identified by using this graph and, with some debugging, was solved. This type of performance issue checking would not have been possible without the use of an overall performance graph.
- Listing of all different fitness functions.
- Listing of all different configurations per fitness function, ordered by fitness, as seen in Figure 3.3: this allows finding the configuration that has achieved the highest fitness for a given fitness function, or the number of trials executed for each configuration. A high fitness function does not correspond necessarily to the best configuration, as the configuration could be more restrictive, such as a configuration that includes opponents in a kick-off versus one that does not. The histogram is an automatically generated histogram from all the trials taken and is sensitive to outliers: in the case of the image, a single negative outlier with around -5.83 fitness stretched the range of the histogram.

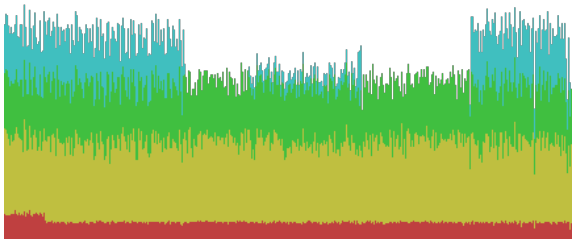

Figure 3.2: Statistics about the performance of the optimization process.

- Listing of the best samples for each configuration, as seen in Figure 3.4. This allows viewing an ordered list of the best samples evaluated so far, with the average fitness value for each sample, and including a fitness histogram for each sample that allows easily understanding how a given sample performs. For example, the histogram allows quickly evaluating whether a kick-off performs many average goals, or a fewer number of good goals, which is a lot more information than a simple average fitness.

- Viewing the parameters of a given sample: this allows copying the parameter definition to test the corresponding model instance in a local environment.

Another crucial feature of the framework is that it is partly continuously integrated in 2 different levels. This allows for quick iterations without constantly deploying code running on workers. The first level is the agent code, which is published online, on a git repository. When a trial is to be done, the master notifies workers of specific code versions to execute. Each worker then fetches that code, compiles it when necessary, and evaluates the samples using multiple threads. Results are sent back to the master, and this process is repeated until convergence has been achieved. The second level of continuous integration is that the core code of the worker itself is continuously integrated, every time a new version is sent to the git repository, workers fetch the new version, shutdown the previous run, and begin executing with the most up to date version. This allows multiple workers to be deployed and updated with ease, requiring only updating a single git repository.

3.3 Experiments and Results

We will now evaluate the performance of our framework, noting any advantages over other methods. Firstly, we will evaluate the performance of the three different optimization methods: single core, multithreaded and distributed. Afterwards, we will compare the expected time taken for each method and the actual time taken for the distributed one in a real world workload. Finally, we will share our experience in using the framework in a time-constrained scenario.

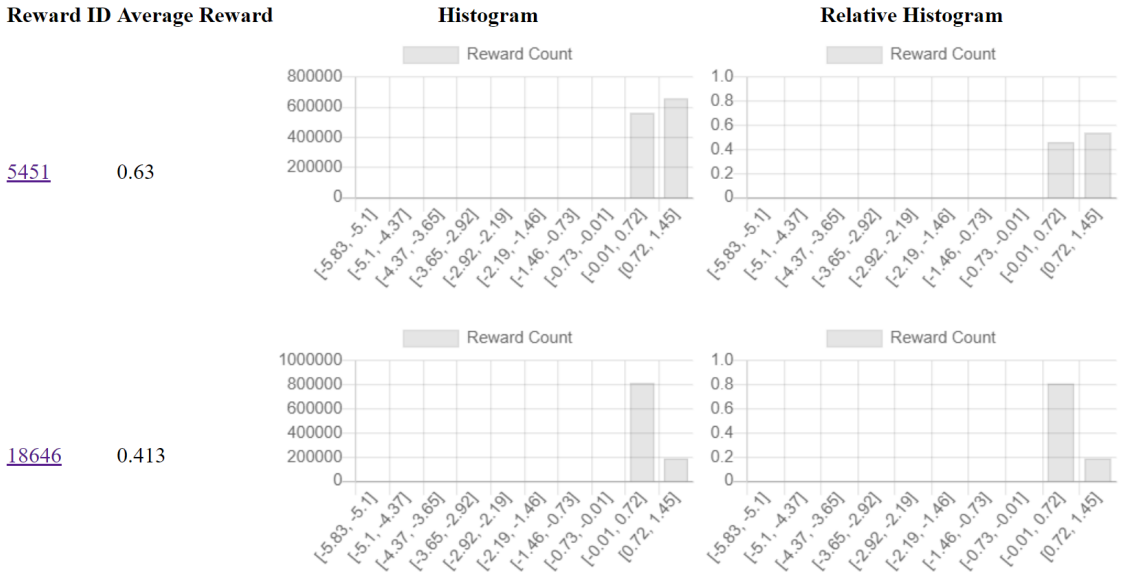

Figure 3.3: Listing of two configurations for a given fitness function.

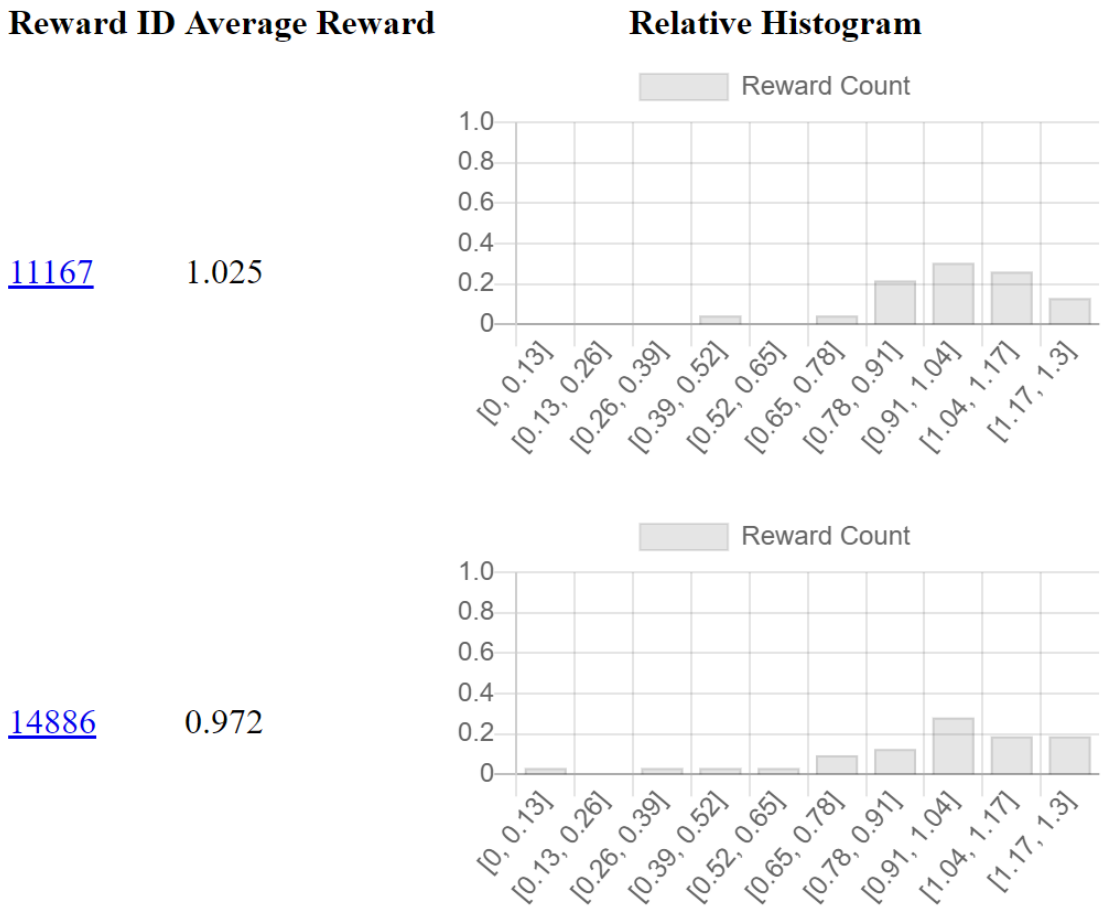

Figure 3.4: Listing of the two best samples for a given configuration.

Table 3.1: Hardware specifications of the machines used by the distributed optimization framework.

Name	CPU	Memory
Laptop1	i7-7700HQ 3.8GHz	16GB DDR4
Laptop2	i5-M460 2.53GHz	4GB DDR3
Desktop1	i7-4770K 4.3GHz	24GB DDR3
Server1	Dual X5650 2.66GHz	8GB DDR3

For each of the three optimization methods, we measured the average, minimum, and maximum amount of trials evaluated over six ten minute intervals. Table 3.1 shows the hardware specifications of the worker computers used for these runs. *Laptop1* was used for the first two methods, while all the computers together were used for the distributed runs. The first method used a single core, and the second used four physical cores, with hyper-threading enabled, for a total of eight logical cores.

The results of our evaluation are shown in Figure 3.5. Memory usage never exceeded 4GB in any machine, and CPU usage was at 100% in all four computers, showing that the optimization process is CPU bound. It is quite clear that the distributed system greatly outperforms the other methods, as expected, since it utilizes all the computer resources available. We can also see a much greater variation in the minimum and maximum values in the distributed system, due to synchronization between threads and networking.

The second experiment consists of running a real world workload, an optimization problem, using the framework. We optimized the High-Level Kick-Off and the Mixed-Level Kick-Off, which are described in the next chapter, and took performance measurements. The relevant information regarding the performance of the framework is that for the former kick-off it took $440k$ trials to optimize, while the latter took $1200k$ trials. The performance information given by Figure 3.5 is comparable to the workload that is being executed in this experiment. Thus, we can use the average trials per 10 minutes to calculate the expected time to compute all the trials for both optimizations. Along with this information, we also have included the actual time taken in the real life workload, in Table 3.2. The distributed framework improves the time taken to optimize by 1400%, when comparing the single core and distributed performance of optimizing the high-level kick-off. The actual time taken is larger than the expected time for the distributed platform for many reasons, including *Laptop1* not being continuously on-line during the optimization, and failures related to networking and power.

Finally, we had to do a last minute optimization for Robotica 2019 (Festival Nacional de

Table 3.2: Expected time according to measurements for a real world workload to finish, as well as the actual time taken.

	Single core	Multithreaded	Distributed	Actual Time
High-Level Kick-Off	628 hours	148 hours	43 hours	70 hours
Mixed-Level Kick-Off	1770 hours	417 hours	123 hours	175 hours

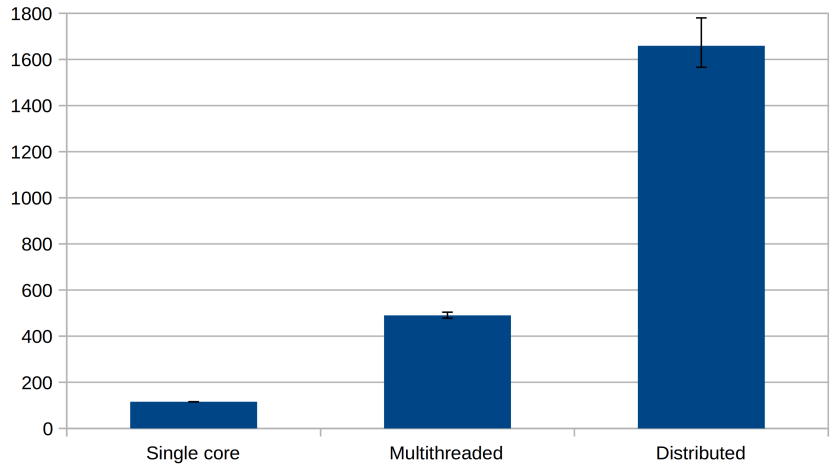

Figure 3.5: The performance of the three framework architectures.

Robótica 2019): we had just finished implementing a skill, and we had around 24 hours to optimize it. Using the computers in Table 3.1, as well as 2 servers graciously provided by Universidade de Aveiro, we were able to accelerate the optimization process greatly, simulating a kick-off with two of our players, and 2 magmaOffenburg opponents. Although CMA-ES did not converge, we achieved excellent results, as seen in the Experiments and Results chapter of this dissertation. This would not have been possible with a simple single core or multithreaded solution. Thus, the framework is useful for speeding up operations when time is a constraint.

Chapter 4

Multi-Robot Learning of High-Level Kick-Off Skill

We examine how we used the distributed optimization framework to optimize a high-level skill that we implemented, the kick-off. Having a performant kick-off is essential for playing a match well, independently of whether the soccer is simulated or in real life, as it enables an early goal right when the game begins, or to recover from an opponent's goal.

4.1 Concept and Approach

In soccer, a setplay is a set of studied movements, which can be practiced, that allows a team to achieve an advantageous situation. One such setplay is a kick-off, which is performed either when the game begins, when teams switch sides, or when a goal is scored by the opposite team.

The FCPortugal3D codebase already had a framework for defining kick-off setplays, with a few basic kick-offs already implemented. Thus, we began by working using the already defined framework, expanding upon one of these kick-offs, developing a new skill that is executed by two players, a passer and a kicker. The low-level skills that were used are some of the ones present in the FCPortugal3D team:

- Omnidirectional walk, which allows the players participating in the setplay to quickly walk anywhere, from any starting position, and is used by both the passer and the kicker.
- Controlled kick, which allows the passer to pass the football to the kicker at the beginning of the setplay.
- Long-distance kick, allowing the kicker to kick the football to the goal.
- Get up skill, in case the players fall and need to get back up.

The passer is positioned near the middle of the field, next to the football, and the kicker is positioned behind the passer. The passer uses the controlled kick to pass the ball to the kicker, moves out of the way, and the kicker then tries to score a goal.

This skill by itself, which we will call the base model, underperforms: parameters, such as the starting locations of the players and the kick targets, were handpicked with no foresight, the skill sometimes works, but not reliably. To solve this issue, we will parameterize the model, adding small variations to the model's parameters, in the form of deltas from their original values.

We define two different kick-offs: one where we optimize the initial positions of the agents and the kick targets, which we call high-level kick-off; and one where we further optimize the low-level parameters of the kicks used in the strategy, called mixed-level kick-off. We will compare the optimized models amongst themselves, and with the base model.

In order to perform the optimizations, a fitness function will be defined to quantitatively describe how the parameters for the model perform, in order to optimize the model with CMA-ES. The models will firstly be optimized with just the passer and the kicker, and then with simulated opponents. The results of the trained models will be shown in the next chapter.

4.2 Fitness function selection

The first fitness function for this strategy took into account several factors, such as whether there was effectively a goal and how quickly it was performed. The initial fitness value f_i was given by:

$$f_i = \begin{cases} g & t \leq 10 \\ g - \frac{t-10}{20} & t > 10 \end{cases}, \tag{4.1}$$

where $g \in \{-1, 0, 1\}$ represents whether there was a goal (1), no goal (0), or a self goal (-1) and t the time taken to achieve the goal. This function was defined in order to reward goals within ten seconds, and penalize strategies up to thirty seconds, at which moment the opponents will have overrun the kicker.

The final version of the fitness function also takes into account the height of the football after it was kicked to the goal, in order to make it impossible for opponents to block it, and heavily penalizing kicks which do not leave the ground. The final fitness value f is obtained by multiplying each line in f_i by h:

$$f = \begin{cases} gh & t \leq 10 \\ (g - \frac{t-10}{20})h & t > 10 \end{cases}, \tag{4.2}$$

where h represents the maximum vertical position achieved by the ball.

We averaged the fitness value over 90 trials, to reduce the variance as much as possible without compromising the computational cost of the learning process.

4.3 High-level Kick-Off

The high-level kick-off is a rather simple model, with only eight parameters, that focuses solely on the high-level strategy. Rather than focusing on all the possible parameters, it focuses instead on the positions that the players should start the setplay at, and where they should kick the football. This results in optimizing only the high-level parameters of the model, resulting in a sequential layered learning approach as described in Overlapped Layered Learning [MS18].

A starting position and kick target was handpicked for the passer and the kicker, which is then modified by the parameters of the model, consisting of delta vectors.

The eight parameters are:

- *P_POSX* and *P_POSY*, passer initial position delta vector.
- *K_POSX* and *K_POSY*, kicker initial position delta vector.
- *P_SHOOTX* and *P_SHOOTY*, passer kick target delta vector.
- *K_SHOOTX* and *K_SHOOTY*, kicker kick target delta vector.

4.4 Mixed-level Kick-Off

The mixed-level kick-off is a more complex model, in comparison with the high-level kick-off. It includes 24 parameters and focuses on both the low-level and high-level skills simultaneously. It optimizes all the skill layers simultaneously, including the parameters of the previous skill, but also the low-level parameters for the kick executed by the passer and the kicker. This results in optimizing not only the high-level parameters of the model, but also the low-level parameters, resulting in a concurrent layered learning approach, as described in Overlapped Layered Learning [MS18]. The 16 extra parameters, in addition to the ones shown in for the high-level kick-off, are:

- *P_DELTA_DIST_TO_BALL* and *K_DELTA_DIST_TO_BALL*: takes into consideration the distance to the ball.
- *P_DELTA_DIST_TO_BALL_MINX* and *K_DELTA_DIST_TO_BALL_MINX*: the minimum distance to the ball in the X axis.
- *P_DELTA_DIST_TO_BALL_MAXX* and *K_DELTA_DIST_TO_BALL_MAXX*: the maximum distance to the ball in the X axis.
- *P_DELTA_DIST_TO_BALL_MINY* and *K_DELTA_DIST_TO_BALL_MINY*: the minimum distance to the ball in the Y axis.
- *P_DELTA_DIST_TO_BALL_MAXY* and *K_DELTA_DIST_TO_BALL_MAXY*: the maximum distance to the ball in the Y axis.

- *P_DELTA_REL_DESIRED_POSY* and *K_DELTA_REL_DESIRED_POSY*: the desired relative position in the Y axis.

- *P_DELTA_KICK_ANG_ERROR* and *K_DELTA_KICK_ANG_ERROR*: the maximum angular error.

- *P_DELTA_KICK_ANG_REGION_ERROR* and *K_DELTA_KICK_ANG_REGION_ERROR*, the maximum angular region error.

4.5 Optimization

Although the initial untrained model can score goals, its performance can be improved. Both models will be optimized using the framework described in Chapter 3. Initially, in the first optimization runs, for performance reasons, only the passer and the kicker will be present. This ensures that the set of optimized parameters, although not ideal for real world soccer games, is as good as it can be, to have a good starting point for more costly optimizations.

Using the CMA-ES algorithm, we will optimize the model without early-stopping, to give time for the algorithm to explore the full parameter space. This will be done for both the high-level and the mixed-level kick-offs. We should obtain parameters for the model that allow it to substantially improve the amount of goals it performs on a field with no opponents.

After optimizing, we will select the high-level and mixed-level kick-off parameters that have achieved the highest average fitness over the 90 trials. These are not necessarily the best parameters, as there is a factor of luck involved, but they are the ones most likely to be. The resulting models for these 2 sets of parameters will also be compared with the base model.

We optimized the high-level kick-off until it reached convergence in approximately 370 generations, whereas the mixed-level kick-off took approximately 1000 generations to achieve convergence. Both kick-offs used the default population size of 10. The maximum fitness function of each generation, for each model, can be seen in Figure 4.1.

We also compared the baseline performance and the performance of the high-level and mixed-level kick-offs. The metric used was the average, minimum, and maximum fitness f, over 1000 tests, of the best samples obtained. A fitness $f = 0$ is a failed kick-off, while higher values represent increasingly better goals. The optimized models can be viewed at `https://youtu.be/RILWVmaSETk` for the baseline model and `https://youtu.be/bO61mAD7rig` for the mixed-level kick-off.

The results are outstanding, as seen in Figure 4.2, with a fitness average of 0.249 for the base model, which increases to 0.388 for the high-level kick-off, and to 0.701 for the mixed-level kick-off. Both scenarios show an increased performance with respect to the base model of 55% and 182%, respectively. It is interesting to note that, while the high-level kick-off achieved the best goals, the mixed-level kick-off achieved a larger amount of mediocre goals, and so achieved overall better performance.

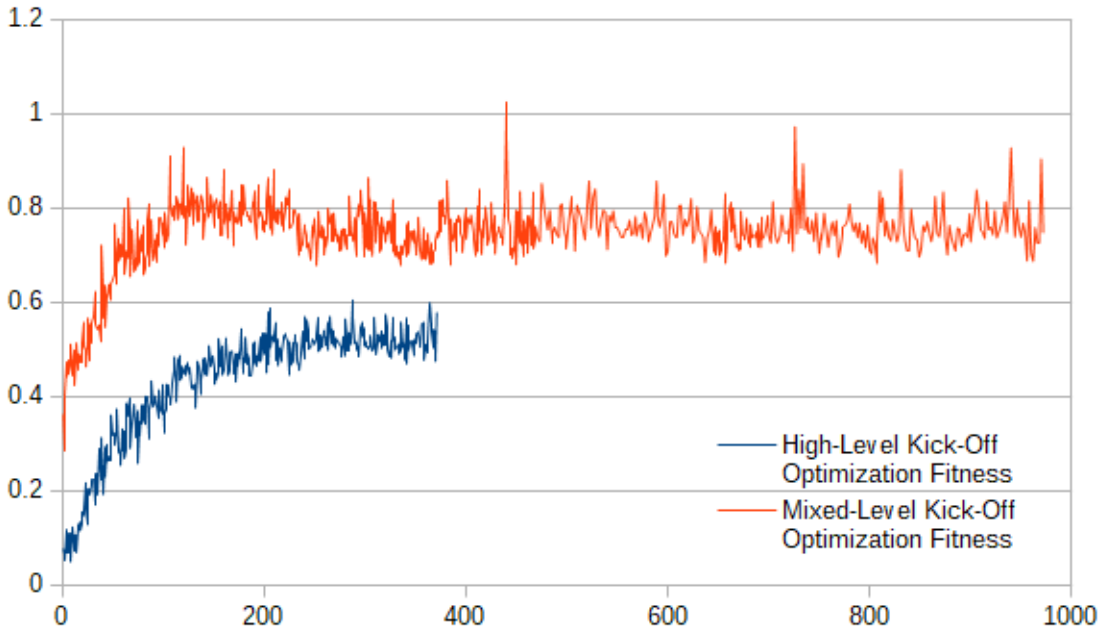

Figure 4.1: Comparison of the evolution of the two kick-off skills.

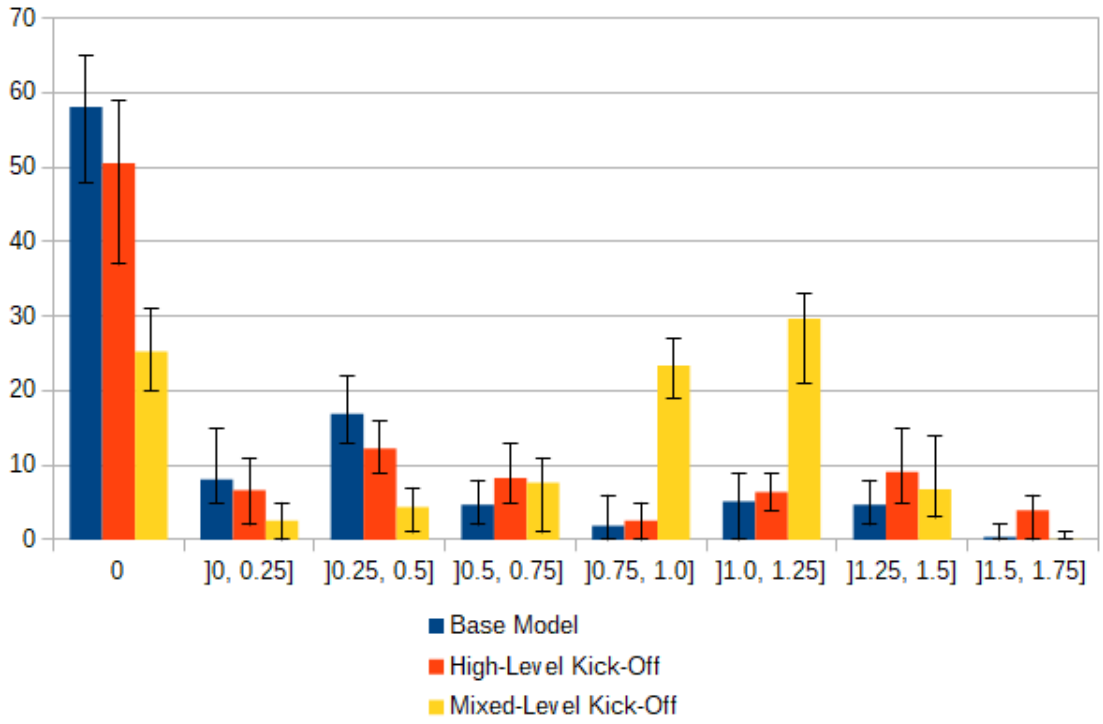

Figure 4.2: Comparison of results between the two optimized kick-off skills and the baseline.

Figure 4.3: Initial position of the players in the blocking wall. Taken from the livestream of a real world RoboCup match.

4.6 Optimization against simulated opponents

To have a more realistic simulated situation for the kick-offs, 2 opponents from the magmaOffenburg team were used, the goalkeeper and the midfielder, numbered, at the time of writing, 1 and 2, respectively. These opponents were selected for the next batch of optimizations as they were in the direct path of the football, causing the most incovenience for the kicker. For this case, we optimized the mixed-level kick-off only.

Having magmaOffenburg opponents resulted in the final parameter set being optimized specifically against the magmaOffenburg goalkeeper and midfielder. Other optimizations could be done against the binaries of other teams, likely resulting in differently optimized parameters for each team.

4.7 Blocking Wall

Upon testing the basic optimized model, with two players, passer and kicker, against the magmaOffenburg team, we stumbled upon a problem: the players did not have enough time to performs the kicks, and they would be quickly overrun by their opponents. To solve this issue, we used 5 players in a line, between the middle of the field and the passer, to act as a wall that would block the opponents, as shown in Figure 4.3. After the passer passed the ball to the kicker, all the players in the wall would crouch, to prevent opponents from interfering in the setplay, as shown in Figure 4.4.

Figure 4.4: Crouching positions of the players in the blocking wall. Taken from the livestream of a real world RoboCup match.

Another topic is blocking the opponents, by standing in front of them, to the crouching behavior, which was originally intended to give our players an advantage in case the setplay is unsuccessful. This later turned out to be essential, as the synergy of the blocking wall and the kick-off proved so successful, that crouching in front of opponents in the way the blocking wall is performed was heavily penalized, if not outright banned. To solve this topic and successfully block other opponents, our players would have to assign to agree between themselves who blocks who (an assignment problem), intercept the opponents and block them, standing in their way. Additional work that could be performed would be lowering the number of players needed to successfully block the opponent, resulting in more free players to perform other tasks.

Chapter 5

Experiments and Results

After developing a well optimized kick-off skill, it is time to test it against other teams. Firstly, we will compare its performance in simulated soccer matches against other teams and afterwards in real world matches against other teams during a live event.

5.1 Experimental Setup

After optimizing the mixed-level kick-off with simulated opponents, during Robotica 2019 (Festival Nacional de Robotica 2019), there were a large amount of sets of parameters that we could choose from, with varying fitness values. For the first experiment, rather than choosing the one with the best fitness value, we opted to choose the best 10 sets of parameters according to the fitness value, and test their effectiveness in 10 fully simulated kick-offs, including every single player of both FCPortugal3D and magmaOffenburg in the kick-off. We then picked the best 3 and ran 30 more fully simulated matches for each.

Secondly, for Robotica 2019 the FCPortugal3D team played matches against other teams. Firstly, there was a round robin round zero of games, to allow teams to test their implementations. Afterwards, there were round robin matches, semi-finals with the 2nd versus the 3rd placed, and then the 1st versus the 4th placed. After the winner of each semi-final was ascertained, there was a final with both winners against each other, in order to determine who was 1st and 2nd.

5.2 Simulated kick-offs against other teams

As can be seen in Figure 5.1, it is quite clear that the performance in the optimization is dissimilar from the performance in simulated matches. This can be due to two main factors: the optimization does not accurately simulate the kick-off, since it does not simulate the blocking wall, nor the entirety of the opposing team; and a factor of luck: over thousands of optimization rounds, some very high fitness scores are bound to be obtained by mere coincidence.

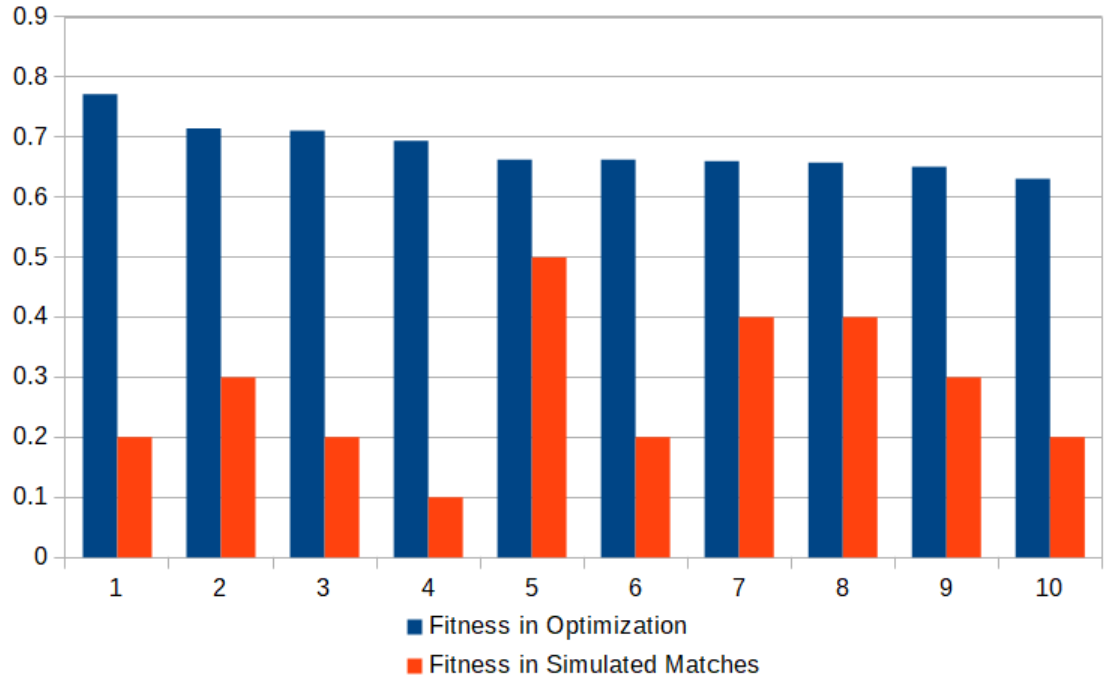

Figure 5.1: Simulated kick-offs

From there, we picked the best 3 sets of parameters amongst the highest simulated kick-off fitness, models 5, 7 and 8. Amongst these models, we simulated 30 more kick-offs, to determine which we should use for real world matches. The resulting fitness value of these simulations can be seen in Figure 5.2.

There was a very close match between sets of parameters 5, 7 and 8: 5 had a fitness of 0.400, while 7 and 8 tied for 0.367. We have concluded that these models are very alike when run against the magmaOffenburg team, and further kick-off simulations against other teams would be required to discover which model is more effective. Thus, with simplicity in mind, we chose the set of parameters with the highest fitness, model 5.

5.3 Real world performance

In regards to the real world results in Robotica 2019, the final positions of the various teams were as follows:

1. magmaOffenburg
2. AIUT3D
3. FCPortugal3D
4. BahiaRT

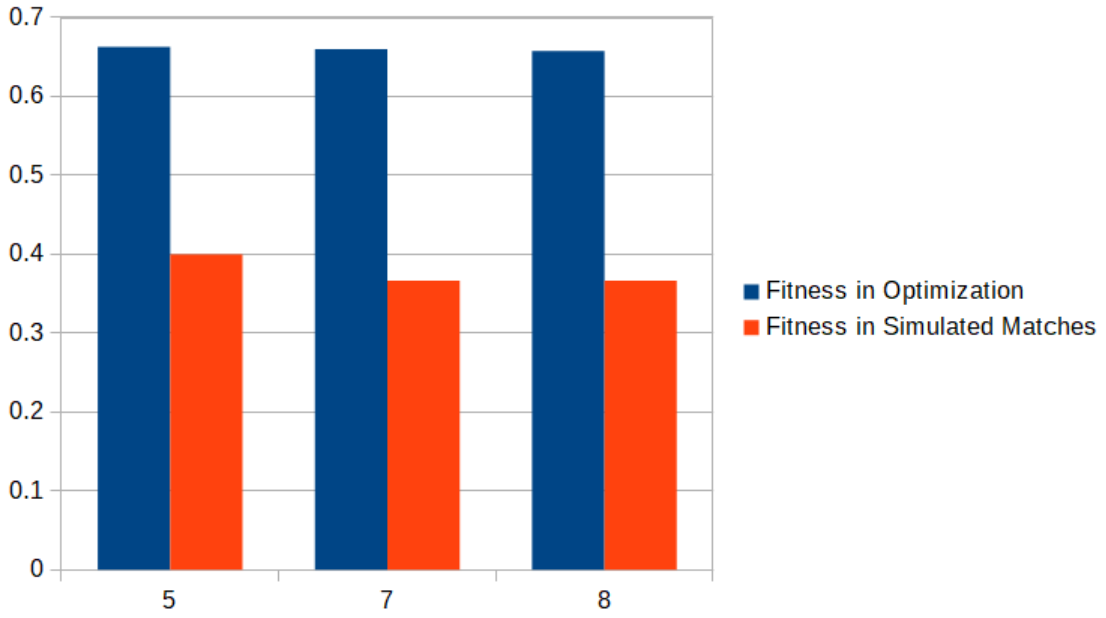

Figure 5.2: Comparison between the best 3 sets of parameters found in simulated kick-offs

Thus, there were a total of 3 round robin groups of matches played, 2 semi-final matches and 1 final match. The round robin groups of matches each had had the following matches:

1. AIUT3D versus ITAndroids
2. FCPortugal3D versus AIUT3D
3. BahiaRT versus FCPortugal3D
4. magmaOffenburg versus BahiaRT
5. magmaOffenburg versus ITAndroids
6. FCPortugal3D versus ITAndroids
7. BahiaRT versus AIUT3D
8. magmaOffenburg versus FCPortugal3D
9. BahiaRT versus ITAndroids
10. magmaOffenburg versus AIUT3D

The semi-finals had 2 matches:

1. FCPortugal3D versus AIUT3D
2. magmaOffenburg versus BahiaRT

Table 5.1: All the different types of ways kick-offs failed during the matches.

Failure Type	Description
1	Football hit the wall when passed
2	Football was kicked not far enough to reach the goal
3	Football was passed; failed to be kicked
4	Football was passed; missed the goal
5	Football was kicked to the goal; defended by goalkeeper
6	Football was kicked to the goal; defended by player other than goalkeeper
7	Kick-off failed due to a bug in handling tie-breaker matches.

The final was a match between magmaOffenburg and AIUT3D.

In this section we will analyze the kick-off skill that was performed in every match the FCPortugal3D team took place in. A kick-off can either succeed, resulting in a goal, or fail for a particular reason, as shown in Table 5.1.

Starting with round 0 (viewable at https://youtu.be/ch1Jr1uKRQk), we had the mixed-level kick-off skill implemented and optimized at this point, albeit without taking the other opponents in consideration. The performance for these matches is displayed in Table 5.2. There was 1 goal and 4 failures in total, and the kick-off was responsible for winning the BahiaRT versus FCPortugal3D match. From this data we conclude that what made the kick-off unsuccessful was its execution, since the failure resulted from the action of the passer, kicker, or the players participating in the wall (failure types 1, 2 and 3).

In round 1 (viewable at https://youtu.be/zma9tQuLLNE), we used exactly the same kick-off as round 0. The results of the matches in this round can be seen in Table 5.3. There were 3 failures and 1 goal in total, which puts the average goal rate of this particular kick-off over rounds 0 and 1 at $\frac{2}{7} = 0.29$ in real world conditions.

In round 2 (viewable at https://youtu.be/liCumHnE1QI), we improved the kick-off slightly: we changed the configuration of the blocking wall, namely the initial positions of the players, and we ran the optimization against the simulated opponents, picking the best set of parameters as detailed in the previous section.

In the semi-final (viewable at https://youtu.be/6HhWRyXsars), our kick-off was responsible for tying the match in the first part, after AIUT3D scored a goal, requiring a tie-breaker. For round 2 and the semi-final each kick-off resulted in an average of $\frac{4}{10} = 0.4$ goals, which is a fantastic result.

Table 5.2: Round 0 matches.

Match	Score	Goals	Failures	Failure Types
FCPortugal3D versus AIUT3D	1-0	0	1	1
BahiaRT versus FCPortugal3D	1-2	1	1	2
FCPortugal3D versus ITAndroids	1-0	0	1	1
magmaOffenburg versus FCPortugal3D	0-1	0	1	3

Table 5.3: Round 1 (first 4 matches) and round 2 matches (last 4 matches).

Match	**Score**	**Goals**	**Failures**	**Failure Types**
FCPortugal3D versus AIUT3D	2-0	0	1	1
BahiaRT versus FCPortugal3D	0-2	1	0	
FCPortugal3D versus ITAndroids	3-0	0	1	1
magmaOffenburg versus FCPortugal3D	0-0	0	1	1
FCPortugal3D versus AIUT3D	3-0	0	1	1
BahiaRT versus FCPortugal3D	0-3	1	0	
FCPortugal3D versus ITAndroids	2-0	1	0	
magmaOffenburg versus FCPortugal3D	1-0	0	2	6,5

Table 5.4: Semi-final match.

Match	**Score**	**Goals**	**Failures**	**Failure Types**
FCPortugal3D versus AIUT3D	1-1 (1-2)	2	3	4, 6, 7

Chapter 6

Conclusions and Future Work

RoboCup is a test bed for multi-robot learning, which is an open field, with many open topics that are yet unsolved. Each league presents its unique challenges, with the 3D Simulation Sub-League requiring an emphasis on both low-level skills and high-level skills. Low-level skills have been sufficiently developed in the 3D Simulation Sub-League, but high-level skills are still open to large improvements, and their development has been emphasized by the rules of the league. In this thesis, we proposed applying Machine Learning to the multi-robot domain, and implemented both a framework for optimizing high-level skills and a kick-off high-level skill.

Firstly, we introduced Machine Learning in general terms, comparing supervised, unsupervised and Reinforcement Learning approaches, and how agents can learn in discrete or continuous action and state spaces. Afterwards, we delved further into Reinforcement Learning, introducing classical algorithms such as Q-learning and Gradient Descent, along with others such as TRPO and Deep Reinforcement Learning, in the context of a single robot. The classic algorithms have a major flaw: they only work in discrete domains, thus we delve in more detail on how algorithms can be discretized from a continuous domain or, ideally, thought out with continuous domain in mind. Finally, still regarding single-robot learning, we introduced the CMA-ES algorithm, which has been used extensively in this thesis, and contextual learning, which is required for agents to learn a slightly different task.

Regarding multi-robot learning, we introduced the concept of a Multi-Agent-System (MAS) and the definition of low-level skills and high-level skills as defined by Stone et al. in RoboCup. Furthermore, we delved in how multiple agents can learn how to perform actions collaboratively, multi-agent learning, and defined two different approaches: team learning and concurrent learning. Also, robots can organize themselves in homogenous, heterogenous or hybrid teams, and face the credit assignment problem: which robot should receive the reward from an action? We then discussed modelling the behavior of teammates and communication between agents. The final three topics regarding multi-agent learning were scalability, adaptive dynamics and problem

decomposition. We then studied collaborative actions, noting its advantages and disadvantages versus single-robot actions.

To close the chapter, we analyzed more specifically learning in RoboCup, in the context of the 3D Simulation Sub-League. First, we defined how teams generally implement their low-level behaviors, such as walking, kicking or getting up, and how high-level skills are implemented, by using tactics or setplays, for instance. We then performed an in-depth analysis of both low-level and high-level skills of the FCPortugal3D team, and the high-level skills of the magmaOffenburg and UT Austin Villa teams. In the case of FCPortugal3D, we surveyed their walking, running, four different types of kicks, and how the agent get up. Regarding their high-level skills, the full framework was studied: how each player is assigned a role, and how they form relationships with other systems, such as tactics, behaviors, formations and situations. The SBSP and DPRE systems were analyzed, along with topics such as coaching, interpretability, communication using the ADVCOM system and a language and graphical interface to define setplays. Regarding the magmaOffenburg team, we surveyed the tools they use for developing their high-level skills, including a faster simulator that mimics the default rcssserver3d simulator, and how they define the high-level skills of their team. In regards to UT Austin Villa, their dynamic role assignment and positioning system was introduced, along with their concept of Overlapped Layered Learning.

After that, we introduce the Distributed Optimization Framework, in order to speed up the optimization process by using more computers, as the time taken for optimizing high-level skills was a difficult constraint. Its architecture is outlined, and the three main optimization methods discussed, the different types of information the framework manages, and how each framework component communicates with each other, using HTTP endpoints. The interface that the framework presents to the user is explained, both in terms of the meta information presented, and the data regarding the optimization process. The framework is described to be continuously integrated on two levels, both on the skill level and the framework itself. Its distributed performance is then compared versus a single computer, with outstanding results, and the time improvement achieved by running a real workload is compared against the theoretical speeds.

Then we begin implementing, and then optimizing, a high-level skill: the kick-off. We introduce FCPortugal3D's setplay framework, and the four low-level skills that were used in implementing our kick-off. We then defined the high-level and the mixed-level kick off, along with the fitness function we'll use the evaluate their performance. The parameters of each kick-off are defined, along with how they will be optimized using CMA-ES. The resulting fitness histogram was analyzed for the two best set of parameters and the baseline model. The mixed-level kick-off was then optimized against simulated opponents, and a wall was added to prevent opponents from interfering with the kick-off, which was later banned for use in later RoboCup competitions, due to how effectively it synergized with the kick-off.

To conclude, we ran two sets of experiments: one where we found the best kick-off to use for Robotica 2019, by choosing the top 10 parameter sets, and picking the three best, and then choosing the best amongst the three, by running simulated kick-off with fully simulated teams. Afterwards, we analyzed the performance of our kick-offs during Robotica 2019: in rounds 0 and

1, there were big issues with the blocking wall, which were later fixed in round 2 and the semi-final. The results speak for themselves: for rounds 0 and 1 we achieved an average of 0.29 goals per kick-off, and in round 2 and the semi-finals this value increased to 0.4.

As future work, the blocking wall will have to be changed, to comply with the rules of the 3D Simulation Sub-League. More high-level skills, including corner kicks or general setplays, can also be implemented and optimized using the same distributed framework. The performance of the developed skills can also be measured in simulated matches against all other teams and the baseline FCPortugal3D, in terms of average goal difference, in order to ensure that the skill results in a net gain for the team.

References

[ALR+16] Abbas Abdolmaleki, Nuno Lau, Luis Paulo Reis, Jan Peters, and Gerhard Neumann. Contextual policy search for linear and nonlinear generalization of a humanoid walking controller. *Journal of Intelligent & Robotic Systems*, 83(3-4):393–408, 2016.

[APL+17] Abbas Abdolmaleki, Bob Price, Nuno Lau, Luis Paulo Reis, and Gerhard Neumann. Contextual covariance matrix adaptation evolutionary strategies. *IJCAI International Joint Conference on Artificial Intelligence*, pages 1378–1385, 2017.

[ASL+16] Abbas Abdolmaleki, David Simões, Nuno Lau, Luis Paulo Reis, and Gerhard Neumann. Learning a humanoid kick with controlled distance. In *Robot World Cup*, pages 45–57. Springer, 2016.

[BDF+] Martin Baur, Klaus Dorer, Jens Fischer, Duy Nguyen, Carmen Schmider, and David Weiler. The magmaOffenburg 2017 RoboCup 3D Simulation Team.

[CAA+14] João Cravo, Fernando Almeida, Pedro Henriques Abreu, Lu\'\is Paulo Reis, Nuno Lau, and Lu\'\is Mota. Strategy planner: Graphical definition of soccer set-plays. *Data & Knowledge Engineering*, 94:110–131, 2014.

[CM12] Jonathan H Connell and Sridhar Mahadevan. *Robot learning*, volume 233. Springer Science & Business Media, 2012.

[DG16] Klaus Dorer and Stefan Glaser. The magmaOffenburg 2012 RoboCup 3D Simulation Team. *Proceedings of RoboCup*, 2016, 2016.

[FLHI+18] V. François-Lavet, P. Henderson, R. Islam, M.G. Bellemare, and J. Pineau. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3-4):219–354, 2018.

[FRM12] R. Ferreira, L.P. Reis, and A.P. Moreira. Omnidirectional kick for a humanoid robot. In *Iberian Conference on Information Systems and Technologies, CISTI*, 2012.

[GLSL16] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

[Han16] Nikolaus Hansen. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[JM15] M.I. Jordan and T.M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[KAK+97] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. RoboCup: The Robot World Cup Initiative. In *Proceedings of the International Conference on Autonomous Agents*, pages 340–347, 1997.

[LHP+15] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[LR07] Nuno Lau and Luís Paulo Reis. High-level coordination methodologies in soccer robotics, robotic soccer. *Itech Education and Publishing*, pages 167–192, 2007.

[LR14] Nuno Lau and Luís Reis. *FC Portugal 3D Simulation Team: Team Description Paper*. nov 2014.

[LRC07] Nuno Lau, Luís Paulo Reis, and João Certo. Understanding Dynamic Agent's Reasoning. In *Portuguese Conference on Artificial Intelligence*, pages 542–551. Springer, 2007.

[LRS+13] Nuno Lau, Luís Reis, Nima Shafii, Rui Ferreira, and Abbas Abdolmaleki. Fc portugal 3d simulation team: Team description paper. *RoboCup 2013*, 2013.

[MBM+16] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[MBS12] Patrick Macalpine, Francisco Barrera, and Peter Stone. Positioning to Win: A Dynamic Role Assignment and Formation Positioning System. *AAAI Workshop - Technical Report*, 2012.

[MDS15] Patrick MacAlpine, Mike Depinet, and Peter Stone. UT Austin Villa 2014: RoboCup 3D Simulation League Champion via Overlapping Layered Learning. In *AAAI*, pages 2842–2848, 2015.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, and Others. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[MLR10] Luis Mota, Nuno Lau, and Luís Paulo Reis. Co-ordination in RoboCup's 2D simulation league: Setplays as flexible, multi-robot plans. In *de Robotics Automation and Mechatronics (RAM), 2010 IEEE Conference on, Singapore*, 2010.

[Mot07] Luís Paulo Mota, Luís Reis. Setplays: Achieving coordination by the appropriate use of arbitrary pre-defined flexible plans and inter-robot communication. In *Proceedings of the 1st international conference on Robot communication and coordination*, page 13. IEEE Press, 2007.

[MRL11] Luís Mota, Luís Paulo Reis, and Nuno Lau. Multi-robot coordination using setplays in the middle-size and simulation leagues. *Mechatronics*, 21(2):434–444, 2011.

[MS18] Patrick MacAlpine and Peter Stone. Overlapping Layered Learning. *Artificial Intelligence*, 254:21–43, jan 2018.

[MUB+12] Patrick MacAlpine, Daniel Urieli, Samuel Barrett, Shivaram Kalyanakrishnan, Francisco Barrera, Adrian Lopez-Mobilia, Nicolae \cStiurc\ua, Victor Vu, and Peter Stone. UT Austin Villa 2011: A champion agent in the RoboCup 3D soccer simulation competition. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 129–136. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[PG17] H.A. Pierson and M.S. Gashler. Deep learning in robotics: a review of recent research. *Advanced Robotics*, 31(16):821–835, 2017.

[PL05] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[RL01a] Luis Paulo Reis and Nuno Lau. Coach unilang-a standard language for coaching a (robo) soccer team. In *Robot Soccer World Cup*, pages 183–192. Springer, 2001.

[RL01b] Luís Paulo Reis and Nuno Lau. FC Portugal Team Description: RoboCup 2000 Simulation League Champion. In Peter Stone, Tucker Balch, and Gerhard Kraetzschmar, editors, *RoboCup 2000: Robot Soccer World Cup IV*, pages 29–40, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[RLO00] Luis Paulo Reis, Nuno Lau, and Eugénio Costa Oliveira. Situation based strategic positioning for coordinating a team of homogeneous agents. In *Workshop on Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, pages 175–197. Springer, 2000.

[RLR+13] Luís Reis, Nuno Lau, Luís Rei, Nima Shafii, and Bruno Pimentel. FC Portugal 3D Simulation Team: Team Description Paper. 2013.

[RN94] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, England, 1994.

[Rob18a] Robocup. Objective, 2018.

[Rob18b] RoboCup Simulation League Organization. Soccer Simulation League, 2018.

[SLM+15] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel. Trust region policy optimization. In *32nd International Conference on Machine Learning, ICML 2015*, volume 3, pages 1889–1897, 2015.

[SS96] Satinder P Singh and Richard S Sutton. Reinforcement learning with replacing eligibility traces. *Machine learning*, 22(1-3):123–158, 1996.

[SV98] Peter Stone and Manuela Veloso. Layered approach to learning client behaviors in the robocup soccer server. *Applied Artificial Intelligence*, 12(2-3):165–188, 1998.

[SV99] Peter Stone and Manuela Veloso. Layered learning and flexible teamwork in robocup simulation agents. In *Robot Soccer World Cup*, pages 495–508. Springer, 1999.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

REFERENCES

[Wat89] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

[WD92] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[Woo09] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[WRR+17] Théophane Weber, Sébastien Racanière, David P Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomenech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, and Others. Imagination-augmented agents for deep reinforcement learning. *arXiv preprint arXiv:1707.06203*, 2017.

[YCPZ06] Lin Yang, Chee-Meng Chew, Aun Neow Poo, and Teresa Zielinska. Adjustable bipedal gait generation using genetic algorithm optimized fourier series formulation. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4435–4440. IEEE, 2006.

[YJC13] Z. Yan, N. Jouandeau, and A.A. Cherif. A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, 10, 2013.