Robocup SLL Strategy Group 1

Adrian Swande*, Oskar Frej†, Gustav Samuelson‡, Lukas Bonkowski§, Ivan Blazanovic¶
School of Innovation, Design and Engineering, M.Sc.Eng Robotics
Mälardalens University, Västerås, Sweden
Email: *ase22003@student.mdu.se, †ofj22001@student.mdu.se, ‡gsn22003@student.mdu.se,
§lbi25001@student.mdu.se, ¶ibc24003@student.mdu.se

***Abstract*—**

***Index Terms*—AI, Autonomous Robots, RoboCup, Soccer**

I. Introduction

The RoboCup [**?**] is a tournament where different teams compete against each other with soccer playing robots. The RoboCup Federation arranges several types of leagues where every league uses different types of robots in different shapes and sizes. Overall, this tournament aims to advance in the scientific field of mobile robots. This project will focus on the Small Size League (SSL), division B in particular. In the SSL division B teams compete in 6 vs 6 matches of two halves where each half is five minutes long with a five-minute pause in between. The robots are constrained to certain physical dimensions according to the rules (the robots need to fit inside a cylinder of 0.18 meters width and 0.15 meters height) and the robots are built by the members of each team. The playing field is 10.4 times 7.4 meters with a playing area of 9 times 6 meters and the game is played with an orange golf ball. The rules of this league are similar to regular soccer but with several modifications. For example the rules include yellow and red cards, freekicks and penalties but also rules like maximum shooting speed and maximum dribbling length. The aim of this project is to develop a system that works well in simulation. That will be done by creating an AI system that can coordinate all six robots, handle the ball, score goals and defend against the opponents. In the long term the models we develop could be further developed and used in other works related to both RoboCup and other areas. In this paper we aim to answer the question of how transferable policies trained in simpler, more abstracted simulators like VMAS are to more accurate simulators such as grSim, and at which level of the hierarchal AI model.

II. Background

A. Autonomous Mobile Robots

According to Kate Brush[**?**], an (Autonomous) Mobile Robot is a robot that is capable of moving around and navigating through its surroundings with the help of for example software, sensors and cameras. The robots are mainly fitted with legs, wheels or tracks that are used to transport itself around, but they are also used in aerial and nautical environments. They are mainly driven by an automated AI system that is in charge of decision-making. Mobile robots have surged in popularity over the recent years (partly) due to their ability to operate in areas that humans can not/should not be in.

B. Reinforcement Learning

Reinforcement learning: According to Jacob Murel and Eda Kavlakoglu[**?**], reinforcement learning is a machine learning algorithm that is used to develop independent decision-making in autonomous agents. Agents train by repeating similar tasks over a period of time or repetitions, where they learn independently through trial and error. A popular implementation of the learning algorithm is Q-learning. According to GeeksForGeeks[**?**], Q-learning is a model-free RL algorithm that is used for training independent agents to make the best decision possible in each possible situation. It learns through a trial and error system, where it interacts with the environment to find the best method. A state-action-reward system is utilized, where the result of an action taken in a state is rewarded or penalized depending on the outcome. After a training iteration it writes its Q-values to a Q-table, where the values represent the best known expected reward for taking a given action in a given state. It updates the table using the Temporal Difference rule

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right).$$

For each state, the agent can either choose to explore or to exploit. Using the Epsilon-Greedy Policy (ϵ-greedy policy), the agent decides whether to take the best current known action (exploit), where the agent picks the best action with the highest Q-value based on the probability of $1-\epsilon$. Else it will try to find a new best possible action (explore), where the probability to explore is based simply on the ϵ-value. This is what allows the model to independently over time find the best possible outcomes for each state.

C. Deep Reinforcement Learning

Given a finite amount of actions and states, Q-learning can, thereofore, learn the optimal action to take at each state to ensure the maximum total reward according to some time horizon. In 2013, Mnih[**?**] proposed a variant of Q-learning called Deep Q Network (DQN), in which a neural network is used to approximate the optimal action-value function

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_i + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \,\middle|\, s_t = s,\, a_t = a,\, \pi\right]$$

(which is the maximum sum of rewards r_t discounted by the time horizon γ at each timestep t, achievable by a behavior

policy $\pi = P(a|s)$, after making an observation s and taking an action a), by means of gradient decent of the loss function

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\ U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta) \right)^2 \right],$$

where the quaternion (s, a, r, s') represents a so-called "experience replay" of a past action a at a certain state s, the received reward r and the next states' following the action. With this method, then – unlike with regular Q-learning – an action policy for a continuous state space (like in the scenario of soccer robots in a simulation) can be learned.

D. Other teams

The CMDragons[**?**] team won all six games they played during the RoboCup 2015 competition. In this paper they describe how they used algorithms to divide their robots into defense and offense subteams to suit the state of the game. They switched between the amount of robots depending on parameters such as ball possession, field region and the aggressivness of the other team. In offense, they used algorithms to both estimate the most optimal place to move for robots without the ball as well as the best action for the robot in possession of the ball. In defensive situations, algorithms were used to evaluate the threats. Both first-level and second-level threats were computed in order to stop the robot with the ball to score directly and to stop threatening passing options. Using these methods, the CMDragons were able to win the competition without conceding a single goal.

III. Experimentation

A. Strategy Hierarchy

The following Hierarchy shows how we could organize Team behavior across 5 layers, from high-level strategy down to low-level execution. Each layer builds on the one above it, enabling modular, scalable control.

1) Game Strategy: The top-layer defines the overall team behavior based on the given game state. If we take Rule-based logic for example, we could look at time left to play and score. Then if we are winning and the time is lower than a specified threshold, we could set the Strategy to Stall. This will then provide high-level context for all other decisions made below.

2) Play: The play layer selects coordinated maneuvers such as setting up a wing attack or forming a defensive wall. Selecting plays could be done by a decision tree based on factors like ball position, team formation, and opponent layout. Each play then sets constraints or goals for roles and tactics.

3) Role Assignment: This layer will dynamically assign robots to specific roles (e.g. striker, defender, goalie) based on their position, proximity to the ball, or other factors. Optimization algorithms such as Hungarian matching have been used with great success.

4) Tactic: The tactic layer defines what action a robot should take in its current role. This could be whether the robot should pass, dribble, shoot, or intercept. This layer's decisions are highly context-sensitive and reinforcement learning is a good choice.

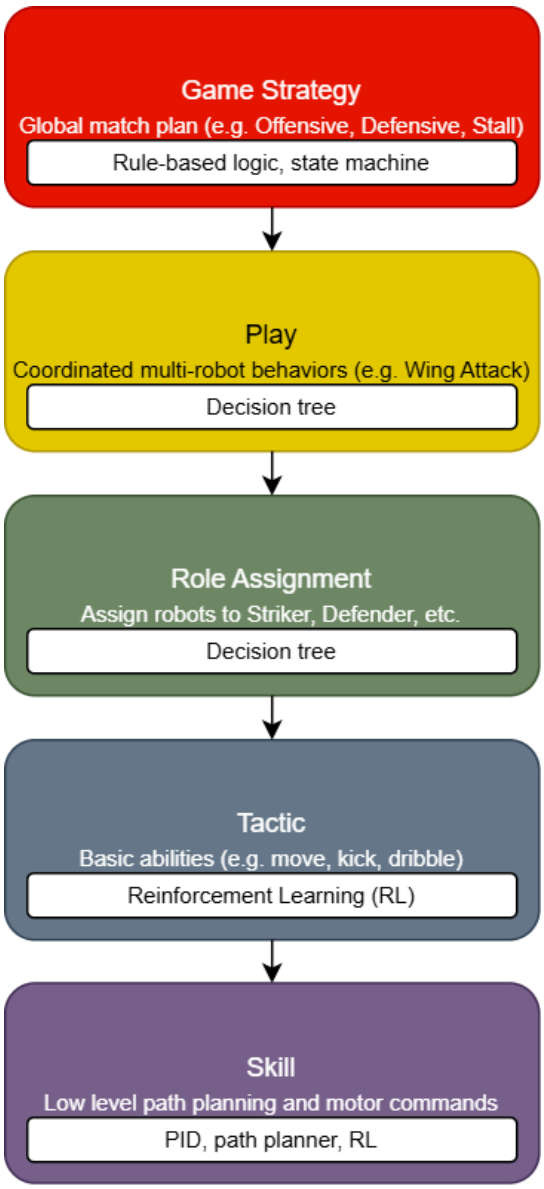

Figure 1. Hierarchical team behavior structure

5) Skill: The skill layer handles the low-level physical execution of actions. This could be moving to a position, kicking (how hard) or dribbling. Commonly used control methods are PID and path planning, but reinforcement learning can also be used to improve fine motor control, adaptability, or performance in unpredictable situations.

B. Training

To prepare our agents for RoboCup SSL, we are currently exploring VMAS (Vectorized Multi-Agent Simulator) as our training environment. VMAS is a lightweight and very customizable simulator to prototype and train multi-agent behaviors. We can set up custom scenarios that simulate different game situations. We can then use it to train both the Tactic and Skill layer behaviors using RL algorithms. In each scenario,

we define the agent's observations, actions, and reward signals to shape the behavior we want them to learn. VMAS also supports fast, parallelized simulation with GPU acceleration, which allows us to efficiently run many environments at once. At this stage, we are still experimenting with different training approaches. The goal is to eventually transfer the trained policies to a more realistic simulator like grSim, where we can evaluate their performance in a full game setting.

C. Agent Architecture: Single vs Multi-Agent

There are two considered approaches herein when creating AI for multiple agents: single-agent or a multi-agent architecture.

A **single-agent** approach would involve one central controller that receives the entire field state and outputs coordinated actions for all robots. This method simplifies coordination and is often easier to implement and train.

A **multi-agent** approach would assign each robot its own agent, possibly with limited field knowledge. This approach is more realistic and can model decentralized behavior, but introduces complexity in coordination and learning stability.

Our initial focus will likely lean toward the single-agent model to reduce complexity during development. However, we may transition to or experiment with a multi-agent setup depending on performance and scalability needs.

IV. Method

V. Results

VI. Discussion

VII. Conclusion

Acknowledgment