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. This specific league uses a centralised vision system that allows all robots to get information about the position of the other robots and the ball at all times. In that way, developers can focus all efforts on the strategical side of the game which makes the SSL perfect for newcomers in the RoboCup. In the SSL division B teams compete in mathces with 6 against 6 robots and the matches consist 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 in width and 0.15 meters in height) and the robots are built by the members of each team. The playing field is 10.4 by 7.4 meters with a playing area of 9 by 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, free kicks and penalties just like in real soccer but also rules like maximum shooting speed and maximum dribbling length[**?**].

The aim of this project is to develop a system of robots 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. To create this AI system two different approaches will be tested, one of them being a type of reinforcement learning and the other one being a genetic algorithm. 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 following research questions:

- Why is it difficult for RL agents to learn complex skills (like shooting and passing) beyond simple navigation?
- How does skills developed with GA perform vs hard-coded skills?
- How do simulation inaccuracies and bugs affect learning and agent performance?

II. Background

A. Autonomous Mobile Robots

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. Behavior Trees

Behaviour trees (BT) are a way to structure the switching between different tasks in autonomous agents. This kind of structure was developed for controlling NPCs (non-player characters) in games and they are both modular and reactive. Modular meaning that the system consists of components that are independent and reusable, e.i. the components can be tested individually or removed without changing the whole tree. Reactive, on the other hand, means that the system adapts to changes in the surrounding space and can for example change its behaviour based on what is happening. The structure of a BT is like a directed rooted tree with internal nodes called control flow nodes and leaf nodes called execution nodes. Each connected node are most often called parent and child where the root is the node without parents. The execution of the tree starts with the root that sends signals to its children to start executing. The cild then returns running when the execution is under way and then success or failure depending on whether the process could complete or not. In this way the flow of tasks can be controlled[**?**].

C. Reinforcement Learning

Reinforcement learning (RL) 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[**?**]. 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 stores its Q-values in 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[**?**].

D. Deep Reinforcement Learning

Given a finite amount of actions and states, Q-learning can, therefore, 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.

E. 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 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.

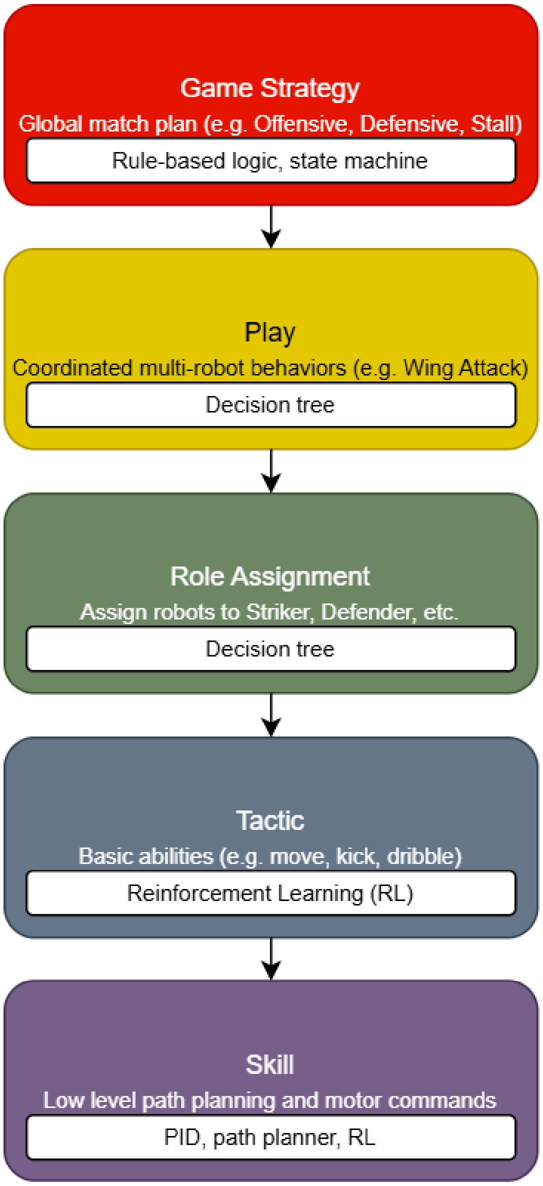
Game Strategy
Global match plan (e.g. Offensive, Defensive, Stall)
Rule-based logic, state machine
Play
Coordinated multi-robot behaviors (e.g. Wing Attack)
Decision tree
Role Assignment
Assign robots to Striker, Defender, etc.
Decision tree
Tactic
Basic abilities (e.g. move, kick, dribble)
Reinforcement Learning (RL)
Skill
Low level path planning and motor commands
PID, path planner, RL

Figure 1. Hierarchical team behavior structure

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.

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

Part of the training and testing process was conducted using the Virtual Multiagent Simulator. A new simulation model was developed to replicate a football match in the B Division of the Small Size League (SSL). The virtual environment included a field and agent-based robots, all scaled to match real-world dimensions. Key game mechanics such as out-of-bounds, goal-out, and corner detection were implemented. Basic functionalities—such as shooting, passing, positioning, dribbling, opening space, and throw-ins—were also incorporated. Each simulation session lasted 10 minutes, with two teams (red and blue), each consisting of six agents. The red team followed a hardcoded script that selected the first available action, while the blue team was trained to select optimal actions using various AI models, including rule-based systems and reinforcement learning techniques.

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.

D. Reinforcement Learning

Our main approach involved training a multi-agent reinforcement learning policy for our RoboCup SSL simulation using the VMAS framework, adapted to the RoboCup SSL specifications. We used Proximal Policy Optimization (PPO) with a centralized critic. We set out to train across multiple scenarios, including "defensive", "ball at centre", and "offensive" scenarios with further differentiation, varying the number of own players and enemy players, the distance to the ball, and the distance to the goal.

The main reason for not just training on full games is that the rewards are more sparse in a full game setup.

It's easier to train agents to shoot a goal when the players are in situations that don't require many actions to shoot a goal.

1) PPO Setup: **Disclaimer:** Due to ongoing difficulties with our simulator and unreliable shooting mechanics, most work was going into solving those issues and not optimizing parameters, reward function, or varying scenarios.

- PPO updates: [e.g., 4 epochs per update, batch size 32, learning rate $1\mathrm{e}{-3}$]
- Actions: high-level (move, shoot, pass, dribble), mapped to low-level continuous actions including (dribble direction and speed, shotpower, pass_scores for each teammate)
- Rewards: rewarded for scoring goals, getting closer to the enemy's goal, getting closer to the ball; punished if the other team scores a goal.

IV. Method

V. Results

A. Reinforcement Learning

Our experiments with reinforcement learning did not give us the results we hoped for. Even in basic scenarios, our agent was not able to score goals consistently. Training in a basic two-player setup did not show any coordinated and cooperative play between the two players. The players run towards the ball slowly and dribble towards the goal. Shooting was hard to replicate even in simplified scenarios. Possible reasons will be discussed in the Discussion section.

VI. Discussion

A. Reinforcement Learning

The results we achieved with applying PPO to our VMAS Robocup SSL setup were not as expected. In this section, I will lay out why I think the results were unsatisfactory and what we could do differently to train our agent to achieve coordinated, cooperative play and, most importantly, score goals and defend effectively.

First and foremost, we would have to create a simulator where the basic mechanics work reliably. Mostly, shooting and passing need updating. Debugging step-by-step revealed that shooting and passing were reliant on the robot somehow getting into the perfect position and having the perfect angle.

Once this is achieved, we also have lots of other ways we could improve our agent.

1) Changing our low level skills: Our agent selects from hard-coded low level skills. For the agent to behave as intended, these have to be carefully selected and implemented robustly.

2) Reward shaping: There is a lot of room for improvement through changing our reward function. One aspect we haven't considered yet in our reward function is passing as a reward. Previous work shows promise in rewarding successful passes, not being intercepted, and giving a negative reward for intercepted passes (Wei, R., Ma, W., Yu, Z., Huang, W., & Shan, S. SRC 2018 Team Description Paper).

With our centralized approach, we can design the reward function with the state and actions of all players, not just individuals. More strategic behaviour could therefore be achieved by maintaining good spacing, occupying key positions, and setting up opportunities for passing and teamwork. Rewards have to be considered carefully. Our shaped rewards must still align with our primary objectives:

- Scoring goals
- Effective defense

so that agents do not focus too much on subgoals at the expense of overall team performance.

Due to our inconsistent results stemming from our core simulation issues, we did not include systematic evaluation and visualizations of agent performance.

In future work, once the simulation inconsistencies are addressed, we plan to implement more systematic evaluation of agent performance. This would include tracking and visualizing learning curves, episode rewards, and success rates for key behaviours such as scoring and passing. These evaluation methods are necessary for diagnosing problems and refining progress on agent results.

VII. Conclusion

Acknowledgment