A testbed for intelligent soccer strategies for the RoboCup.be team

Tim Vermeulen Katja Verbeeck

CODeS, KaHo Sint-Lieven (KULeuven), Gebroeders Desmetstraat 1, 9000 Gent, Belgium

Abstract

Robocup.be is part of an international project to encourage AI and robotic research. Guided by DSP Valley and in association with other colleges and universities KaHo Sint-Lieven cooperates in a Belgian team of small, driving soccer robots to take part in international RoboCup competitions in the future.

This demonstration paper will present an environment to simulate and test intelligent strategies for the RoboCup.be soccer robots. The testbed developed is the result of the first author's master thesis. A physical simulation environment is created, using a physical engine. Next this is extended with a multi-agent system to get a testbed in which intelligent multi-agent soccer strategies can be developed and tested. We focus on incorporating reinforcement learning in our RoboCup team. The ultimate goal is to hybridize the testbed by replacing the physical engine with the real-life robotic soccer game.

1 RoboCup

Robocup.be is part of an international project to encourage AI and robotic research. Guided by DSP Valley [1] and in association with other colleges and universities (Erasmus University College, Lessius University College, Campus De Nayer and Vrije Universiteit Brussel) KaHo Sint-Lieven cooperates in a Belgian team of small, driving soccer robots to take part in international RoboCup competitions in the future. The Belgian team aims for the RoboCup Small Size League which is an international competition for small soccer robots, the height for instance is limited to 15 cm. In those annual competitions teams from many different countries show their best, in June 2010 18 teams participated the Small Size League. In July 2011 a new competition is held. More information can be found on www.robocup.be.

2 The simulation environment

The demonstrated simulation environment exists of two parts, shown in Figure 1, and an interface layer in between. One part is based on a physical engine while another part is the multi-agent environment. The interface layer connects those two and makes the necessary conversions.

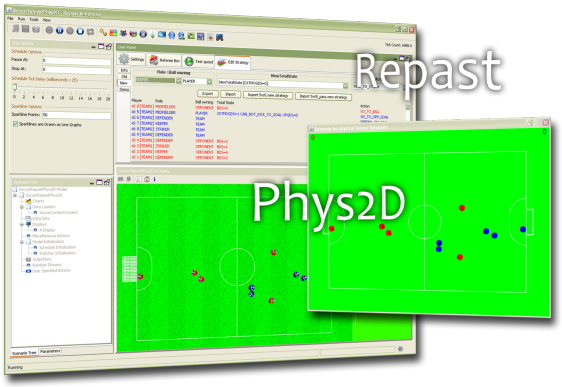

Figure 1: Simulation environment based on Repast Symphony and Phys2D.

Physical engine

The robots, the field and the ball have many physical properties. To simulate the robots and the ball we specify them as bodies in a physical engine. A body can have physical properties like a position, a rotation, a velocity, a rotation speed, an acceleration, a mass, friction. All this is simulated with the physical engine, Phys2D [1], which is written in Java and based on the Game Developers Conference presentation of Erin Catto in 2006. Currently a three dimensional engine is being added.

[1] DSP Valley is a technology network focusing on embedded software and micro-electronics. For more info see www.dspvalley.com.

Multi-agent environment
The controlling of the robots is done by a multi-agent system. Based on practical tests and criteria like the possibility of continuous coordinates, a three-dimensional representation and the possibilities to append the physical engine, Repast Symphony [2] was chosen as the underlying MAS platform. Within this environment the ball and players are defined as agents, each with their own behavior. They update their location every time step to match with the simulated physical world. After updating its location a player uses its strategy, like described in the next section, to calculate the action to take.

3 Strategies

Every player uses a strategy to determine what action to take. A strategy exists of three parts: a state representation, a set of actions the player can take and a policy to select an action in the current state. In our demonstrated test- and simulation environment different strategies can be tested and new ones can be added.

The state representation
The total state of the player is a dynamic set of partial player states: the role, the ball owning, the distance to the own goal, the distance to the goal of the opponent, the distance to the ball, the possibilities to shoot to the goal and the pass possibilities. In some situations some parts of the total state are ignored. This reduces the calculation time per step resulting in a higher update frequency and thus better response times.

The actions
To make a varied player behavior, one need an extensive set of actions a player can take like 'go to the ball', 'go to the goal of the opponent', 'defend the own goal', 'run clear'. Each action corresponds with three skills, one of each type. The move skills will adjust the position of a player, the turn skills will adjust the rotation of a player and the kick skills will kick the ball if necessary. Some examples of implemented skills are: 'run clear', 'move to defender position', 'turn to ball', 'kick to teammate'.

The policy
The only thing left is selecting a proper action to take in the current state. This is perhaps the most important part that specifies some rules to select an action in the current state. This policy can be a simple static policy, hard-coded or specified in a file, or it can be a smarter learning policy based on reinforcement learning [3]. In our demonstrated testbed an extensive static policy is used and tests were done with learning policies.

Local intelligence
Another simulated feature is local intelligence. Some parts of the behavior are implemented on the robot itself, for instance avoiding collisions. This can give better results because the reaction times are smaller and local sensors can be used, but only limited calculation power is available on small size league robots, so implementing the complete behavior local is unfeasible.

4 Demonstration

In the demo we first run a simple match in the physical engine. In this case, the players clearly lack intelligent team behavior. Next we demonstrate how intelligent multi-agent strategies can simply be incorporated in our simulated robot soccer testbed. Different strategies can easily be created and activated for each team. We show manually created strategies versus strategies based on reinforcement learning [3]. The interested listener can play with the tool and at runtime alter the strategy of a team. On-line performance of the used strategies is plotted in a life-updated chart.

References

[1] Kevin Glass and Gideon Smeding. Phys2d - the 2d game physics engine in java. http://www.cokeandcode.com/phys2d.

[2] Argonne National Laboratory. Repast - recursive porous agent simulation toolkit. http://repast.sourceforge.net, 2008.

[3] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: an introduction*. Mit Pr, Mei 1998.