Team Description
Mainz Rolling Brains 2005

Felix Flentge, Christoph Schneider, Tobias Jung, Sascha Metz, Robert Deusser

Department of Computer Science
Johannes Gutenberg-University Mainz
D-55099 Mainz, Germany

1 Introduction

This year we will introduce a whole new decision layer design based on general principles for coordinated multiagent decision making. In last year's Mainz Rolling Brains 3D agent [1] we used the same agent design as in our 2D agent [2–4]. On the decision layer we had several modules for different tasks as passing or dribbling. These modules rated the usefulness of their particular actions depending on the current situation. Then the module with the highest rating was called to execute its action.

We identified two main problems with this approach: First, the coordination between these modules turned out to be quite difficult. To obtain a different agent behavior, changes in several modules were necessary. Also, it was not very clear which module is used in which situation. Second, the switch from 2D to 3D simulation means a considerable increase in the complexity of the agent control. In particular, the handling of the ball got a lot more difficult than before and so now an additional couple of steps is necessary to accomplish the basic tasks as passing or dribbling. This has far reaching consequences. For example, in 2D it was sufficient to think about the pass partner when the agent was in the position to kick the ball because it could kick in every direction quickly. In 3D the agent has to choose its pass partner many simulation cycles before it reaches the ball because it can only kick in the direction from the agent towards the ball. So, in order to play a pass towards a teammate, it has to approach the ball in a certain way. While approaching the ball, the pass partner may also change its position which would make further adaptations necessary.

Our answer to these problems is to introduce new methods for making decisions and executing actions. The basic idea is to use *team actions* which are valid for the whole team and facilitate the coordination of the agents. Based on these team actions we will have a clear structure for choosing the appropriate *individual actions* using explicit *goal functions* which rate the usefulness of the outcomes of the team actions. This general approach to coordinated multiagent decision making is described in the next section. In section 3 we will give a short overview of the overall agent design and give some hints how the general approach is realized in the decision layer. Section 4 deals with the learning of a simple skill: a simulated two-wheeled robot learns to approach a moving ball in order to kick it towards a given target. We conclude with section 5 where we sum up the current status and describe our plans for the future.

2 Coordinated Multiagent Decision Making

We paid special attention to the problem of how to achieve a coordinated team behavior in the absence of explicit communication. The basic idea of our approach is to choose some *team action* based on a clearly structured situation evaluation first and to deduce the appropriate *individual actions* after that. In the following paragraphs we will first describe the different types of actions and types of situations (i.e. state descriptions) we use. Then we show how we arrive at the individual action to be executed by the agent to realize specific goals.

2.1 Actions

In order to achieve a coordinated team behavior, we introduce *team actions* a^T which are chosen from a small set of predefined team actions A^T. Each team action a^T consists of ten *individual actions* $a^P \in A^P$, i.e. one for each player except the goalie. For example, one team action could be "pass from player 5 to player 7". It would consist of the following individual actions:

- player 5: kick the ball towards player 7
- player 7: prepare to receive the ball from player 5
- every other player: move to an appropriate position

The individual actions are invoked by calling the agent's behavioral *skills*. The respective skill is called every time step to realize the action using a least commitment strategy [5], i.e. taking the latest information into account.

2.2 Situations

Ideally, at each time step all players in the team should participate in the same team action and follow the inferred individual actions to realize this team action. To increase the probability that all agents arrive at the same team action, we use a clearly structured and systematic decision process. Therefore, we differentiate between three different kinds of situation descriptions:

- The *world situation* s^W is a complete state description of the full world (estimated by the world model), i.e. a large vector consisting of positions and velocities of all players and the ball.
- The *tactical situation* $s^T \in S^T$ describes the situation of the whole team on an abstract level. Examples are: "ball possession and ball in front of the opponent's goal" or "not in ball possession and ball somewhere in central midfield". Thus tactical situations are from a finite (and very small) set. A hand-coded decision tree classifies world situations into tactical situations (see Section 3).
- The *resulting situation* s^R is obtained by simulating the effects of team actions. It contains all necessary information that allow an evaluation by means of goal functions (see next paragraph).

2.3 Goal Functions

To evaluate team actions we introduce *goal functions* $g_i, i = 1, \ldots, N_g$. A goal function returns a value between 0 and 1 which describes the degree of goal realization in a certain resulting situation s^R. For example, goal functions may evaluate the distance between the ball and the opponent's goal, the number of possible pass partners or the number of players in certain areas. Thus goal functions correspond to "features" in the sense of usual position evaluations. To obtain the value of a certain state we just compute the weighted sum of its individual features. The notable difference between goal functions and features is that goal functions are not defined uniformly on the domain of world situations but operate on the level of the resulting situations.

2.4 Action Selection

The heart of our action selection algorithm is a function that simulates the effects of a team action a^T on the current situation s^W. This function returns the resulting situation $s^R(a^T, s^W)$ and a value $p(a^T, s^W)$ between 0 and 1 that could be interpreted as the probability of applying the team action successfully[1]. We can calculate the utility $u(a^T)$ of a team action a^T by

$$u\left(a^T\right) = p\left(a^T, s^W\right) \sum_{i=1}^{N_g} w_{i,s^T} g_i\left(s^R\left(a^T, s^W\right)\right) \tag{1}$$

and choose the team action a^T with the highest utility. The weights w_{i,s^T} depend on the tactical situation s^T and the specific goal function g_i and describe the importance of the specific goal in the current tactical situation. For example, if the ball is near the opponent's goal, it may not be important to cover opponents, but to shoot towards the goal.

For action selection we first determine the tactical situation s^T and the applicable team actions a^T. These are evaluated using equation (1). To avoid changing team actions too often, there should also be a goal function for maintaining the current team action (e.g. one that rewards the old team action with 1). Finally, we deduce the individual action a^P from the chosen team action.

3 Architecture

Our agent is split into three layers (see Figure 1): the technical layer which handles the communication with the server, the transformation layer which provides the agent with a world model and simple skills, and the exchangeable decision layer which chooses the agent's actions. Last year the decision layer contained the same modular concept (c.f. [2], [3]) which we were already using in the 2D competition. This year the decision layer is completely rewritten from the scratch and implements the principles described above.

[1] A very natural enhancement of this principle would be to have many possible resulting situations $s_i^R(a^T, s^W)$ with corresponding values $p_i(a^T, s^W)$.

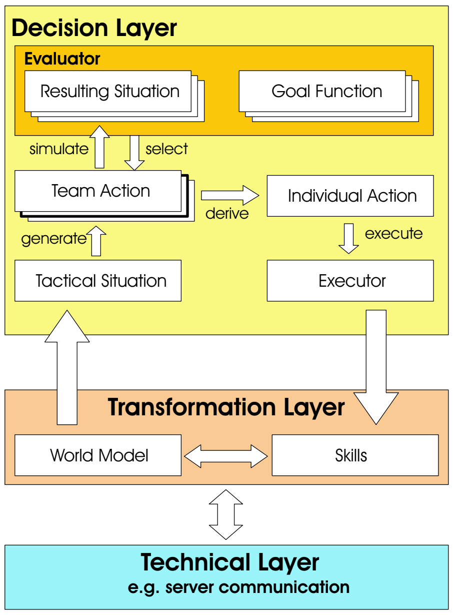


Fig. 1. Architecture of the MRB agent.

As the simulation kernel is based on SPADES, we use the SPADES agent library from P. Riley [6] to handle the communication with the server. We use the same communication scheme which was used by our 2D agent, i.e. the agent waits for a message from the server and updates its world model according to it. Then the player selects an action based on the new information, sends it to the server, and starts waiting for a new message from the simulator.

Our world model, which is part of the second layer, is still quite simple. It stores information about all world objects the agent has seen. Therefore, it is updated every time the player gets new sensor information. Using this information the word model calculates the agent's position with a "weighted flag" algorithm [3]. To reduce the error on the position, the velocity of the player and its old position is also considered. The world model also tries to calculate the velocity of the ball by simulating it forward from the last time it was kicked. As we know the agent's position quite well, we will concentrate on improving the knowledge of the velocities of the ball and the player.

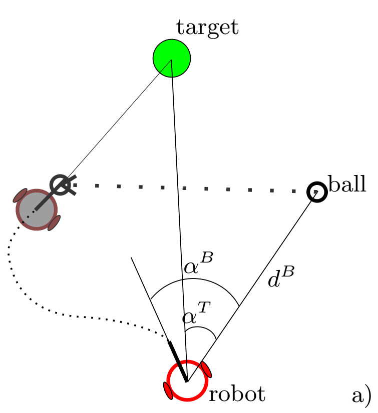
Another part of the second layer are our basic skills. These are an abstraction and extension of the commands provided by the server, as they contain routines for driving to a position or kicking the ball into a specified direction. So far, these routines are coded by hand but we plan to learn some of these skills as described in the next section.

The decision layer implements the concept of coordinated multiagent decision making as described in section 2. We determine the tactical situation based on the information in the world model. Depending on this tactical situation a number of candidate team actions are generated. These are evaluated using the *evaluator* which simulates the effects of the team actions and rates them using the goal functions. The team action with the highest rating is selected. Since the decision process is geared towards coordinated decision making, ideally all agents will arrive at the same team action. The agent's individual action is then derived and passed to the executor which is responsible for performing the respective action using the agent's skills.

4 Learning Simple Skills

We want to apply learning methods to acquire some of the basic skills. Since we expect that two-wheeled robots are to be introduced in the Simulation League very soon, we chose the problem to intercept a moving ball with a simulated two-wheeled robot as a first testbed. The robot is controlled by setting the velocities of its wheels. The ball should be approached in such a way that it could be kicked towards a given target. Unfortunately, simulated two-wheeled robots were not available in the Simulation League at the time of our experiments, so we had to create our own simulation environment. We tried to choose all parameters meaningful with respect to the Simulation League, but we did not consider collisions or acceleration of the robot. Figure 2 a) shows the problem to be solved. In order to kick the ball, the distance d^B between the robot and the ball has to be smaller than 0.07 m and the angle α^B has to be smaller than 22.5°. Since the ball is always kicked in the direction from the robot towards the ball, the angle a^T between this direction and the desired kick direction has to be sufficiently small in order to reach the target.

To solve this problem we use a REINFORCE algorithm [7] in combination with an extended Growing Neural Gas [8] in an actor-critic architecture [9]. The actions we learn are the speed differences which should be applied to the wheels, i.e. one of the wheels is set to the maximum speed and the other one according to this difference. The REINFORCE algorithm is used to learn the situation dependent mean $\mu(x)$ and standard deviation $\sigma(x)$ of a normal distribution $N(\mu(x), \sigma(x))$ from which the real-valued actions are drawn. To approximate the parameters $\mu(x)$ and $\sigma(x)$ we use the extended Growing Neural Gas in a way similar to Radial Basis Function Networks. Figure 2 b) shows some of the learned trajectories. Details can be found in the accompanying RoboCup-Symposium paper which deals exclusively with this learning problem [10]. We are going to apply this learning method to the "real" Simulation League as soon as possible.



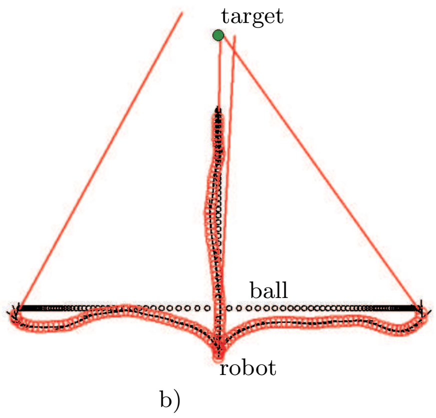


Fig. 2. a) Sketch of the problem to be solved with the relevant distance and angles and a possible solution trajectory. b) Four sample trajectories of the learned approach ball behavior. The robot is the bigger empty circle, the ball is the smaller empty circle and the target is the filled circle at the top. The ball moves in one of the four main directions. The objects were drawn every 100 milliseconds. The straight lines show the direction the ball is kicked in at the end of each episode (the ball moving towards the robot can hardly be seen because it is reached significantly faster than all other balls).

5 Conclusion

We presented a general approach for coordinated multiagent decision making. We currently work on integrating this method in the agent for this year's RoboCup. We also reported our experiments about learning to intercept a moving ball in order to kick it towards a given target. We want to apply the proposed learning algorithm in the Simulation League for learning some of the skills. We also plan to do similar experiments on a midsize-robot.

References

1. Arnold, A., Flentge, F., Schneider, C.: Mainz Rolling Brains 2004 - 3D. In: RoboCup 2004 Symposium papers and team description papers (CD-ROM). (2004)
2. Schappel, B., Schulz, F.: Mainz Rolling Brains 2000. In: RoboCup 2000: Robot Soccer. World Cup IV. (2001)
3. Arnold, A., Flentge, F., Schneider, C., Schwandtner, G., Uthmann, T., Wache, M.: Team description. Mainz Rolling Brains 2001. In: RoboCup-01. Robot Soccer World Cup V. (2002)
4. Arnold, A., Flentge, F.: Mainz Rolling Brains 2004 - 2D. In: RoboCup 2004 Symposium papers and team description papers (CD-ROM). (2004)
5. Burkhard, H., Bach, J., Berger, R., Brunswieck, B., Gollin, M.: Mental models for robot control. In: Advances in Plan-Based Control of Robotic Agents. Volume 2466 of Lecture Notes in Computer Science., Springer (2002)

6. Riley, P.and Riley, G.: SPADES – a distributed agent simulation environment with software-in-the-loop execution. In: Winter Simulation Conference Proceedings. (2003) 817–825
7. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning **8** (1992) 229–256
8. Fritzke, B.: A growing neural gas network learns topologies. In Tesauro, G., Touretzky, D., Leen, T., eds.: Advances in Neural Information Processing Systems 7, Cambridge MA (1995)
9. Sutton, R., Barto, A.: Reinforcement Learning. An Introduction. MIT Press, Cambridge, Massachusetts (2000)
10. Flentge, F.: Learning to approach a moving ball with a simulated two-wheeled robot. In: RoboCup International Symposium 2005, Osaka (submitted)