The Dainamite 2007 Team Description

Holger Endert[1], Robert Wetzker[1], Thomas Karbe[2],
Axel Heßler[1], and Felix Brossmann[1]

[1] DAI-Labor, TU Berlin
Faculty of Electrical Engineering
and Computer Science
{holger.endert|robert.wetzker|axel.hessler|
felix.brossmann}@dai-labor.de
[2] Technische Universität Berlin
Faculty of Electrical Engineering
and Computer Science
thomaskarbe@gmx.de

Abstract. This paper gives an overview about the structure of the dainamite agent and highlights recent improvements of the world-model and the tactic, which were integrated into our framework. To this end, we present effective and robust methods for dynamic role assignment, movement and reachability analysis and opponent modelling. Finally we conclude by assessing our approaches and presenting current and future work.

1 Introduction

The *Dainamite* robocup team has been implemented and improved during student projects since october 2004 [3]. Much of the work done was influenced by [1], which provided a good guideline for developing the basic skills, the world-model and the synchronization with the server. However, since the completion of the basic framework, most of the parts were refined or even replaced, such that the majority of the given team was built upon own ideas.
In this paper we will give an overview about the current state of the *Dainamite* robocup team architecture, and highlight some recent developments, which improved the performance of the team since the last championship in 2006. First, we provide a brief summary of the structure of an agent in Section 2. A more detailed description can be found in [2] and [3]. Thereafter, we address our concepts of dynamic role assignment, the calculation of reachability and agent reflection in Section 3, 4 and 5, which are novel within our team. Finally, we conclude in Section 6 by evaluating our framework and discussing planned and currently existing extensions.

[3] Dainamite is implemented in Java, because it was more common to participating students, accepting the loss of performance compared to C/C++

2 The Agent Architecture

The behaviour of a *dainamite* agent results mainly from the interaction of its constituting components. These are the world-model, the perception, the synchronization (short: synchro), the action, the tactic and the planning devices. In Figure 1 (taken from [2]), their structure and control flow within the agent is visualized. Due to the reactive nature of the simulation, acting is done only after sensing, i.e. after receiving a message from the server that contains sensor information. The first step is to translate the perceived message into usable objects within the perception, and update the world model accordingly. Next, the synchro is informed about the type of the message that has arrived and whether a new server-cycle has begun. The synchro than decides, whether an action has to be computed. Generally, as visual information arrives last, the computation starts with their arrival. However, if this information is expected to arrive late in the current cycle, an action might also be determined based on the body-sense information. The action calculation takes place at different stages. First the tactic modul, called from the synchro, selects the current agent's state. The selection is done by first filtering all states that don't suit the current situation and than comparing the remaining states using planning and/or estimated value functions. The complete mechanism is described in [2]. Once the best state was found, it is executed by calling the appropriate skill inside the action component, which maps a high-level action (state, i.e. dribble) to a low-level action (dash, turn, kick, etc.). Finally, the resulting low-level actions are forwarded to the synchro and sent to the server.

3 Dynamic Role Assignment

One of the most important factors for a good team performance in Robocup 2D is an efficient energy(stamina) management. Each player possesses a given amount of stamina at the beginning of each game half which decreases when he accelerates (dashing) and recovers over time. Without stamina players tend to be much slower. Therefore it is crucial to avoid unnecessary movements as much as possible. One way of doing so is the dynamic assignment of player roles.
The *dainamite* framework knows 11 different player roles. The behaviour of each *dainamite* player only depends on it's role and not on the player's number[4]. These roles are strongly related to a player's position inside the team's formation and there exist roles like 'attacker-left' or 'defender-middle-left'. The initial assignment of these roles is static as each role refers to exactly one player. However, due to ball-interception, low stamina or tactical behaviour the relative positions of two players may change during the game. Therefore, in order to save stamina, we decided to dynamically swap roles whenever two players change positions i.e. when the left attacker finds himself right of the middle attacker. This way we intend to avoid unnecessary movements resulting in a better stamina usage and a faster positioning behaviour.

[4] The goalie never changes his role.

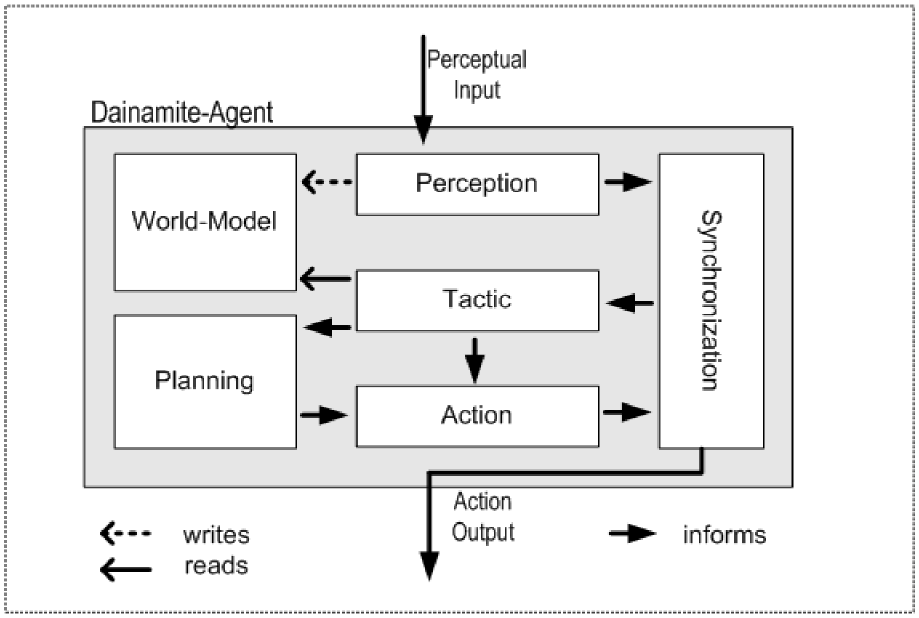

Fig. 1. Interaction of the agents internal components

The role assigment is done based on a predefined ‘perfect’ team formation. The ‘perfect’ team formation describes where all players (roles) should be positioned in the given game situation. Based on this information we find the combination of players to roles with the smallest average distance to the perfect positions using a *weighted least squares algorithm*. The weights are determined by the importance of the role which may differ i.e. based on the distance to the ball.
As a soccer team is a distributed system and the knowledge of each entity may differ we have to make sure that the mapping of roles to players is consistent for all teammates. However, we don’t communicate the different mappings in order to achieve accordance. Instead we try to communicate as much positioning informations as possible thus avoiding most of the knowledge discrepancy and therefore leading to a more consistent mapping behaviour. In practice this measure seems to allow only a very limited number of inconsistencies.
The dynamic role assigment used by our team leads to a better stamina management and a faster overall movement. However, it is still unclear how this mechanism should be adapted best when we explicitly consider heterogenous players i.e. if you want your center attacker to be the fastest player on the field then this constraint has to be considered during the mapping process. We plan to investigate this behaviour in our future research activities.

4 Reachable Area

In a robocup game, the players and the ball are the only moving objects on the field. A players movement is composed of dash actions, turn actions and the drift that results from its current speed. The player can influence this movement in almost each cycle. On the other hand, the ball movement can only be influnced by kick and tackle actions of players. Thus, if the ball is not close to a player, its future movement is predictable (up to some noise that will be ignored here for reasons of simplification).

In almost all situations of a robocup game, an agent needs knowledge about possible movements of players and the ball. Due to the large action space, possible movements are very numerous, and it is infeasible to calculate all of them. A good example is the problem of finding a good pass to a teammate. Here one could choose a subset of all possible pass directions and kick powers and calculate the best of them. Using this method involves two problems. The first is the expensive calculation and the second is, that we don't know if the optimal pass is contained within the chosen subset. Another way of calculating possible movements of ball and player is to use geometric objects to represent reachable areas of an object instead of taking some examples of its movement. In the pass calculation example, one could find out, which passes can be intercepted by opponents and which will be received by a teammate.

In our calculation, all geometric objects are represented by lines and parts of circles. In Figure 2 (a) we present the reachable area of a player. This area shows the positions a player can reach in a given number of cycles. It includes the kick distance of the player (important for interception). As can be seen, the player can cover the longest distance by only dashing (forward or backward). If he uses one or two turns, the distance that he can pass decreases. The ball movement can be analyzed accordingly.

Our approach is to use this method for determining the applicatability of complex actions, which depend on the movement properties of the players and the ball. In order to reduce complexity, we therefore analyze distinct cases. For example, since area intersection is often expensive to calculate, we simplified the problem by assuming the kick power or the kick direction to be given. The resulting calculation can be done much easier with only one variable part of a kick command. In Figure 2 (b) the possible positions of a ball that was tackled is visualized. In such cases, the kick direction is given by the body direction of the player. So he can control the next position of the ball only by choosing the tackle power. The possible positions are marked as blue line. This geometric form is a result of the ball speed limitations. On the other hand, in Figure 2 (c) the possible positions of a ball that was kicked with a given kick power and variable kick direction is shown. The form is the result of intersecting two circles. The first has the actual ball position in its center, and defines the maximum coverable distance (2.7m). The other circle has its center at the next ball position, resulting from adding its current speed. It defines the area, to which the ball can be kicked with the given kick power.

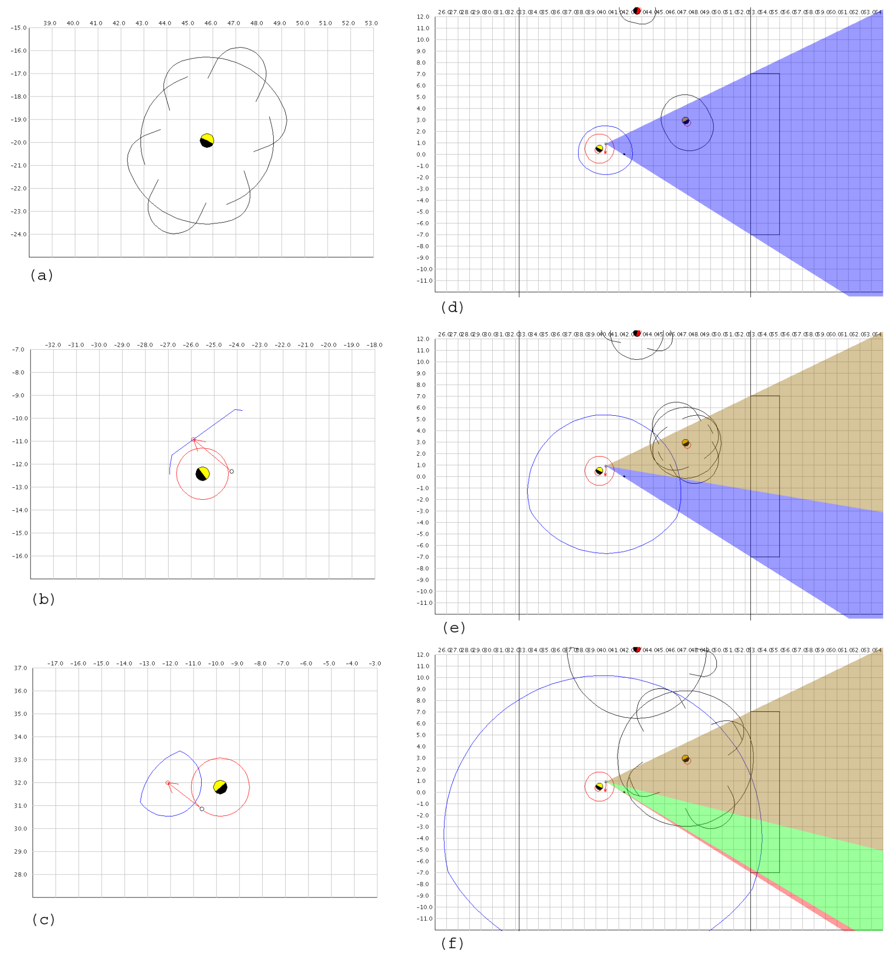

Fig. 2. Geometric calculation of a reachability. (a) Reachable area of a player, (b) Possible positions of the ball after a kick with given direction and variable power, (c) Possible positions of the ball after a kick with given power and variable direction (d) - (f) Geometric calculation of the Scoring

In our current version of the dainamite team, the three geometric analyses are used for the scoring calculation. The best kick is calculated by using a given power of 100 and a variable direction. The best tackling is calculated by using a given direction and variable power. In 2 (d) - (f) we present an example of this calculation. The player is in a good position to attack the goal. Only the keeper is in front of him. The blue area marks all the angles in that the ball could be shot towards teh goal. In the next picture (e), the keeper can catch some of these balls. These angles are henceforth marked red. In the third picture (f), the angles that lead to a successful scoring are marked green. The lower part is marked red because of some security distance to not hit the post.
We remark, that an advantage of this method is the good visualization and hence testing possibility. In order to improve its applicability in distinct situations, further work will be spent on approximating the real reachable areas, which are often different due to noise or suboptimal actions taken by players.

5 Agent Reflection and Opponent Modelling

In some cases agents should not only reason about their local actions, but also about the actions of other agents as well. The reasons therefore are manifold. For instance, if agents are teammates, they have to coordinate their actions, and if they are opponents, they can estimate the actions in order to react optimally with respect to the opponents behaviour. Another example is improving the accuracy of the world-model. Knowing the actions of other agents allows to infer information about parts of the world which weren't seen lately, for instance by applying the results of movement analysis as done in [4].

In order to let agents reason about the actions of others, we equipped each player object in the world-model of an agent with a tactical component (state evaluation) together with a few higher level actions (states), which are easy and inexpensive to use in state-evaluation. A list of the states used for reflection is given in Table 1. From this extension, we let the agents infer new information, by determining the expected action of an agent using the knowledge of the estimator, and applying the effects of that action to the player. The kind of information we derive is called *expected players* and *next players*, both holding predicted player states. The former refers to the players assumed state (i.e. its position, speed, ...) in the current cycle, the latter to the assumed players state in the next cycle. Both information types are important enhancements, that are used as detailed next.

5.1 Usage of Expected Players

This information is primarily an improvement of the world-model, und subsequently enhances the determination of own actions that depend on them. For example, the action of each intercepting agent is always calculated within usual ball interception[5], and thus the prediction is already given and usable by the

[5] It is assumed, that each agent intercepts as fast as possible.

State	**Teammate**	**Opponent**	**Comment**
InterceptBall	yes	yes	State that lets an agent intercept the ball.
AttackState	yes	no	State that defines the movement of an offensive or midfield player, when the team is in ball-possession.
MoveBackState	no	yes	Defines the movement of an opponent back, when its team is not in ball-possession.

Table 1. List of states currently used for agent reflection.

reflecting state. The *MoveBackState* is far more imprecise, but also important, as can be seen in Figure 3. There, opponents are moved towards their own goal, if they are not visible and our team is in ball-possession. Especially when attacking the opponents goal, some players are often in an agents back, and thus can't be seen. In the figure, the world-model of the yellow agent is displayed. In the upper part, he can see the red player (visualized as red opaque player), which is indicated by the yellow area. Then he turns towards the ball in order to intercept and during the following 10 cycles he can't see the red player anymore. After 5 cycles, the expected position is quite good, as can be seen in the middle part of the figure. In another 5 cycles, the expected position becomes more incorrect, but is even more precise than the last seen position, which is displayed as black spot (lowest part of the figure). Table 2 shows the average improvement of an agents world-model accuracy for the expected players, compared to last seen positions, whereas Δ_{Dist} **Game** is the average distance error in a complete game, and Δ_{Dist} **Attack** the average distance error only in attacking situations. As can bee seen, the expected positions are better in average, though only little when averaging over the complete game. When considering the attacking cases separately, the difference on average error is higher, following from the impact of the *MoveBackState*.

	Δ_{Dist} **Game**	Δ_{Dist} **Attack**
Seen Player	2.48m	3.43m
Expected Player	2.36m	3.05m
Improvement	0.12m	0.38m

Table 2. Average error in player-position knowledge of attacking agents.

Certainly, the overall performance of the prediction depends on the behaviour of the opponent team. If this behaves completely different than assumed, the error will increase instead. In either case the prediction is a rather pessimistic one, such that action selection will be more cautious. Hence, the team performance should not suffer, if the opponent behaves different. Finally note that our

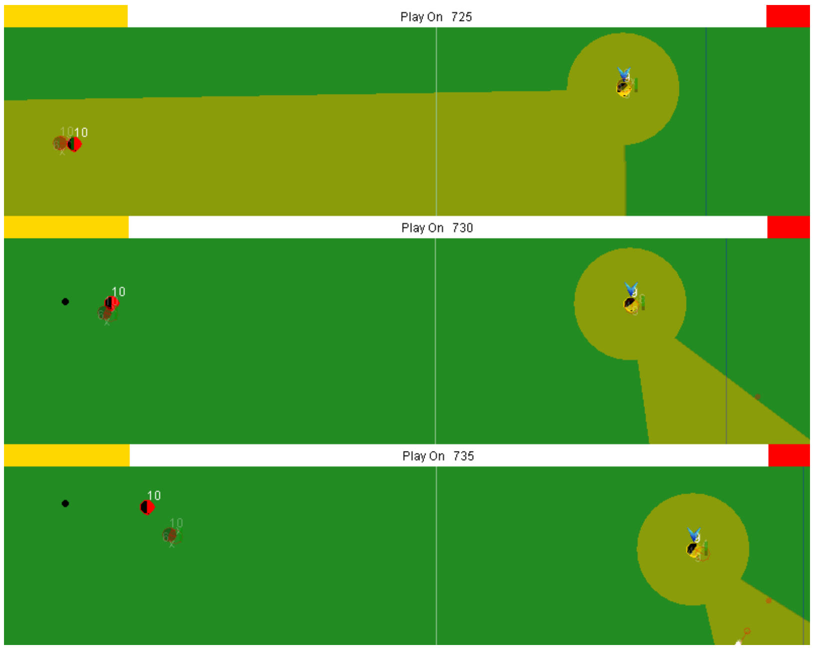

Fig. 3. Difference between expected and last seen players.

world-model is also improved via communication that was deactivated for the prediction evaluation.

5.2 Usage of Next Players

This information is relevant for coordination purposes between teammates, especially for passing. In this case, the following situation is given: If the ball-controller decides to perform a pass, the receiver will notice that soonest in the next cycle, either through visual information or communication. Thus, the receiver has one action to perform, before deciding to intercept the pass. If the ball-controller can predict this action, and subsequently the next state of the teammate, its assessment of the pass-situation becomes more accurate, and its ability to perform critical passes increases. Hence the *AttackState* predicts the behaviour of teammates, if in ball-possession. Note, that this reasoning is done only by one party (i.e. the ball-controller), whereas the other acts based on its usual tactics. Otherwise, if both use reflection, their predictions have to include the reflections of the other party, which has to include the reflection of the original party, and so on. Figure 4 shows the players world-model with next players, displayed as black shadows. The ball-controlling player assumes, that player 9 is moving forward, facilitating a steep pass.

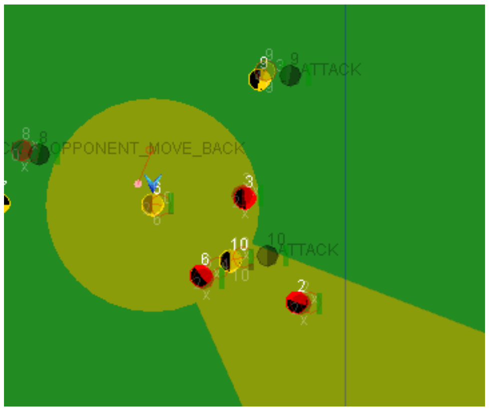

Fig. 4. Visualization of next players used for pass calculation.

5.3 Remarks

Despite our partial usage of the reflection framework, some promising results were already achieved. Since an accurate world-model influences the team performance, it seems worthy to spend effort on this task. In order to make this framework more effective, we have to create better estimators, without increasing the computational costs significantly. Therefore, we plan to use the coach for learning a positioning concept of the opponents. Then, *clang* [5] can be used to communicate rules, which can subsequently be evaluated by the reflection framework of each agent during simulation.

6 Conclusion

As we have shown, we spent a lot of work in creating an appropriate world-model that provides optimal information for situation assessment, that subsequently improved related tactical decisions. Additionally, we achieved a higher flexibility and saved resources by applying dynamic role assignments and increased our accuracy using the given reachability determination. In order to evaluate our new features, we also spent work on visualization of the world-model and the tactic. To this end, we extended the *Soccerscope* monitor from [6] with features for an efficient analysis, for creating statistics and for manual interaction when used within the dainamite framework.
Beside optimizing the basic framework and related components, we are currently working on the integration of learning capabilities for both, player agents and the coach. The former will be based on reinforcement learning in order to improve skills, using the piqle framework [7] as starting point. The latter will be based on CBR, and concentrates on optimal player type selection in a first step. Further work will be done within predicting opponents movements as described in [4]. Finally, skills and tactics in general will be improved, especially the offensive movement without ball when attacking the opponents goal.

References

1. de Boer, R., Kok, J.: The incremental development of a synthetic multi-agent system: The uva trilearn 2001 robotic soccer simulation team. (2002)
2. Endert, H.: The dainamite 2006 team description. In: RoboCup 2006 - Proceedings of the International Symposium. Lecture Notes in Artificial Intelligence, Springer (2006)
3. Endert, H., Wetzker, R., Karbe, T., Heßler, A., Brossmann, F., Büttner, P.: The dainamite agent framework. Technical report, DAI-Labor, Technische Universität Berlin, Germany (2006)
4. Kuhlmann, G., Knox, W.B., Stone, P.: Know thine enemy: A champion RoboCup coach agent. In: Proceedings of the Twenty-First National Conference on Artificial Intelligence. (2006) 1463–68
5. Cheny, M., Dorer, K., Foroughi, E., Heintz, F., Huangy, Z., Kapetanakis, S., Kostiadis, K., Kummeneje, J., Murray, J., Noda, I., Obst, O., Riley, P., Stevens, T., Wangy, Y., Yiny, X.: Robocup soccer server. (2003)
6. Takahashi, S.: (Yowai website: http://ne.cs.uec.ac.jp/ newone/soccerscope2003/)
7. Comite, F.D.: Piqle - a platform for implementation of q-learning experiments (2006)