Acting and Deliberating using Golog in Robotic Soccer — A Hybrid Architecture —

Frank Dylla, Alexander Ferrein, and Gerhard Lakemeyer
Department of Computer Science V
Aachen University of Technology
52056 Aachen, Germany
{dylla,ferrein,gerhard}@cs.rwth-aachen.de

Abstract

Existing approaches to high-level robot control which support deliberation in one form or another usually do not view the time spent on reasoning as critical, which is true in typical applications like office delivery. This is not so, however, in domains like robotic soccer where a team of robots must cooperate in a highly dynamic environment and where actions need to be chosen under tight resource constraints. For this reason, most existing systems for such domains rely on purely reactive architectures without a reasoning component. Our aim is to build robotic soccer agents which are capable of limited forms of deliberation using the action language Golog. In order to meet the real-time constraints, we propose to integrate Golog into a hybrid architecture, which enables the robot to react to the environment very fast as well as to choose actions proposed by the reasoning component. We feel that Golog is particularly well-suited for the reasoning component because it allows to limit the search space by programming complex actions and because of recent advances in extending the expressiveness of the language to deal with issues like continuous change, event-driven behavior, and uncertainty, all of which are important in a domain like robotic soccer.

Introduction

Autonomous mobile robots have been successfully deployed in a number of realistic domains like office delivery or museum tour guides (Simmons *et al.* 1997; Burgard *et al.* 2000; Thrun *et al.* 1999). Typically, such robots exhibit a high-level controller which selects the next action to be executed, which is then handed to other, low-level routines which oversee the actual execution of the action such as navigating to a certain location. High-level controllers can, roughly, be divided into those which do not make use of (logic-based) deliberation and which make up the majority of existing systems (Murphy 2000), and those which do such as (Myers 1996; Thielscher 2002; Burgard *et al.* 2000; Boutilier *et al.* 2000).

Being able to reason about one's actions clearly has advantages in that a) the user is able to specify a task at an abstract level and b) the robot can project the outcome of its actions and choose the best course of actions according to some measure like cost or utility. There is a price to pay, however. Reasoning does not come for free and is indeed intractable in general settings such as planning. Action languages like Golog (Levesque *et al.* 1997) alleviate this problem to some extent by offering programming constructs to limit the search space, but even there the problem does not go away. Hence it is not surprising that the applications where reasoning mechanisms have been employed are usually not time-critical. For example, it does not matter much if the robot spends seconds or even a minute or more to figure out an optimal schedule to deliver mail. But what about highly dynamic domains like robotic soccer where a timely response is critical? Indeed, with few exceptions such as (Jensen & Veloso 1998), most existing soccer playing systems do not use a principled (logic-based) reasoning component at all or only to a limited extent as in (Murray, Obst, & Stolzenburg 2001; Burkhard 2001).

Challenges and opportunities for deliberation in robotic soccer present themselves as follows: On one hand, a high degree of reactivity is necessary to cope with the ever changing environment, which often presents unexpected opportunities like a ball rolling into close range. Even if the robot has the ball, it should not just stand around and muse about what to do next. Chances are that an opponent will intercept and the ball is lost to the other team. On the other hand, deliberation clearly seems useful for intelligent game play such as playing a double-pass to out-play the opponents or to force an off-side by coordinating with the other defenders. The question then is how to allow for some deliberation, which may involve comparing several candidate plans, without losing the advantage of quick responses.

In this paper, we want to argue that a recent extension of Golog is a good candidate for the deliberative component of a robotic soccer agent, but, compared to previous uses of Golog, it needs to be integrated into the system in a hybrid fashion, similar in spirit to (Jensen & Veloso 1998). (We will discuss the connection to this work at the end of the paper.)

Our proposal is to adopt a hybrid approach to how the next action to be executed is chosen. To guarantee a fast response, we use a reactive component which suggests an action solely based on the current state together with some heuristics to determine the utility of the possible actions. At the same time, a Golog reasoning system runs asynchronously, which takes the current world model and projects a number of candidate plans, chooses the best one and returns it for execution. A separate action selection module chooses among the

actions returned by the reactive component and the next action of the plan returned by Golog and initiates its execution. Note that by running Golog asynchronously, it does not have direct control over the robot and it may take some time to produce a plan. If Golog does not return a plan in time, then there is always an action to choose from the reactive component, which may not be optimal but is likely much better than doing nothing.

In the remainder of this paper we will describe the proposed hybrid architecture which combines reactive behavior and deliberation in more detail. All examples are drawn from robotic soccer, which has been the driving force behind developing this architecture. However, we feel that the principles involved go beyond soccer playing robots and apply more generally to robots acting in highly dynamic environments. This is very much a report on work in progress and it probably leaves open more questions than it answers. On the other hand, we feel we are heading in the right direction of showing that cognitive robotics can have significant impact in areas where so far it has played little role, if any.

The rest of the paper is organized as follows. In the next section, we briefly introduce the RoboCup soccer domain. We then present our hybrid architecture for combining deliberation with reactivity, illustrated by examples and issues from robotic soccer. We finish with a brief discussion and give an outlook on future work.

The RoboCup Domain

The efforts around robotic soccer are coordinated by the RoboCup organization (RoboCup 2002). RoboCup provides a number of benchmarks to test and compare ideas in mobile robotics, in particular, soccer playing robots.[1] Soccer naturally lends itself to comparing the effectiveness of ideas in competitions, which are held annually at various national and international events. Due to the wide area of different problems, five soccer leagues are distinguished. The simulation league is the only one using software-agents. The small- and mid-size league, the Sony four-legged and humanoid-robot league are played with real hardware. We concentrate on the simulation- and mid-size league, mainly for two reasons. For one, these are the leagues we are experimenting with and competing in ourselves. For another, these two leagues provide rather complementary challenges. The simulation league requires the coordination of a fairly large team of agents, yet without the burden of dealing with actual sensors and effectors. The mid-size league, on the other hand, features small teams of robots, where each robot can be equipped with rather sophisticated sensors and effectors.

In a little more detail, we have the following: In the mid-size league, four robots with a maximum extension of $40\times40\times80$cm are on a team. They have kickers to hold and kick the ball. A model of the world — so a model of the situation on the playground — has to be obtained through sensors like laser range finders and video-cameras. In the simulation league, there are 11 agents on each team. The world model needed for computing the next actions are given by the simulation-server, called SoccerServer (Noda *et al.* 1998).

[1]There is also *RoboCup Rescue*, which deals with robots rescuing people from disaster zones.

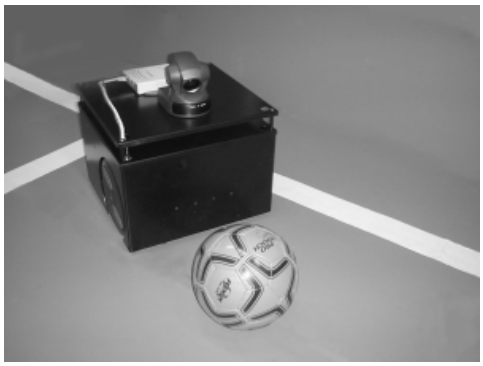


Figure 1: One of our mid-size robots

Despite the differences in the two leagues, we believe that the architecture presented in this paper is appropriate for both leagues.

An architecture for RoboCup

Overview

Figure 2 summarizes our architecture which enables us to use deliberation as well as benefit from reactive behavior.

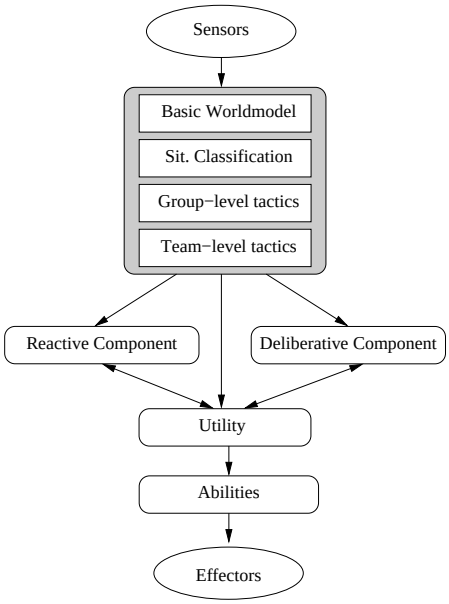


Figure 2: Architecture

Using sensor data as input the world model (surrounded by the gray box) keeps track of the positions of the players and the ball. Simple facts about the game like the starting positions of the players are stored here as well. The world model also provides a (simple) classification mechanism which, roughly, labels a given situation as "good" or "bad." Finally, the world model stores information regard-

ing the tactic currently played by the team as a whole or by a group of players like the defenders.

The world model is used by the reactive component (RC), the deliberative component (DC), and action selection component (ASC). The RC computes from the given situation a "best" next action, whereas the DC (Golog) projects several candidate plans selects one together with a first action. The ASC then chooses an action from those suggested by the RC and DC and hands it to the module Abilities, which handles the actual execution, which eventually translates into control signals to the effectors such as the motors or the kicker of the robot. Here we list the main abilities, which correspond to the actions available to RC and DC:

1.	go	move to a certain position
2.	$attack$	attack an opponent player
3.	$dribble$	dribble to a certain point
4.	$stop\_ball$	stopping the ball
5.	$pass$	pass the ball to a certain position or player
6.	$goal\_shot$	shoot towards the goal
7.	$blocking$	blocking an opponent
8:	$pass\_intercept$	intercept the ball

We will not go into any details of how these abilities are implemented except to note that, in the case of the simulation league, these are simply given. We also ignore low-level issues like sensor interpretation, collision avoidance, or self-localization, which are important but would lead us too far afield. Finally, we completely ignore the issue of communication between robots for the purpose of coordination and the fact that there are adversaries out there with their own choice of actions. While these are very important issues, we simply do not have not much to report here yet, but hopefully will have in the future.

In the following, we describe in more detail the four components central to our architecture: the world model, the reactive, deliberative, and action selection component.

The World Model

The world model is built from the sensory inputs and static information about the game. It is divided into four sections. In the basic world model, information such as the position of the players and the ball is stored. These values can be computed directly from the sensory input. Keeping track of this information over the time, one can compute the movement vectors of the opponent players and the ball. With that one can make prediction of future positions of objects.

The task of the module *Situation Classification* is to first map the information of current player positions relative to the ball to some more abstract game description, where the position of the ball is taken as the point of reference. This information is then used as a starting position for the deliberative component to find a preferred game situation. Figure 3 illustrates the basic concentric grid representation with the position of the ball at the center. This representation is adapted from (Stone, Riley, & Veloso 2000).

The gray dots indicate teammates and the black ones adversaries. We classify situations by building groups going

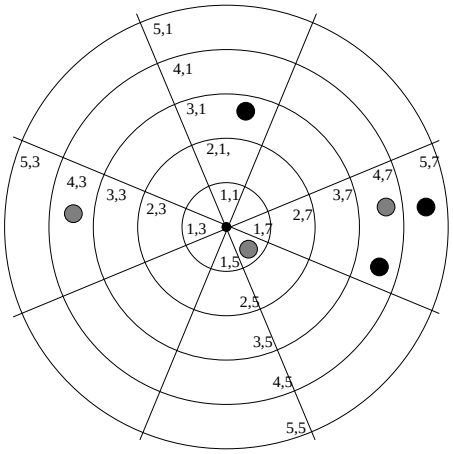


Figure 3: Model for situation classification with the ball centered

from the ball through the cones from the inside to the outside. If the ball is at position $(5.5, 12.1)$, which is somewhere in the right mid-field, the situation can then be encoded as:

$$[\{ball : midfield, right\}\ \{[r_0 : 1, 6]\}\ \{[o_0 : 3, 1]\}\ \{[r_1 : 4, 3]\}\ \{[o_1 : 4, 7][r_2 : 4, 7][o_2 op, 5, 7]\}].$$

The r_i denote teammates and o_k opponents. Furthermore we define events like $lost\_ball$ or $in\_good\_goalshot\_position(r_0)$. The $lost\_ball$-event for example is generated if our team was in $ball\_possession$, for no agent of our team $ball\_in\_kickable\_area(r_i)$ is valid and for at least one opponent this is true.

To what extent we have to take direction and speed information into account during classification is currently under investigation. In (Stone, Riley, & Veloso 2000) the number of players per cone or the relative directions and distances per player are mentioned as good approximations for situations.

Another role of classification is to evaluate a given situation as "good" or "bad." Consider, for example, the following situation from the mid-size league:

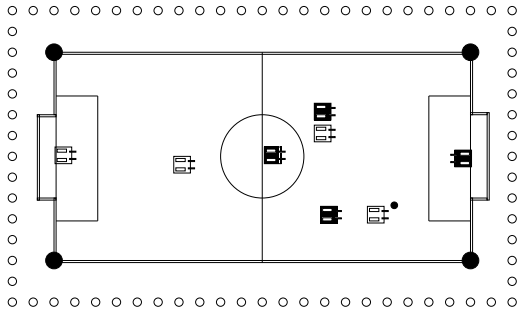


Figure 4: A worthwhile game situation for the white team

Clearly, this situation should be classified as "good" by the white team and, by symmetry, as "bad" by the black

team. We are currently working on developing simple evaluation functions for this kind of classification.

The *Group-Level* and *Team-Level Tactics* can be viewed as analogues of the directives of the coach in real soccer. The module *Group-Level Tactics* defines the tactical parameters for the different player groups defense, mid-field and offense. A good tactic for the defense in real soccer, if playing with four defenders (see Team-Level Tactics), might be forming a string parallel to the goal-line, as so to head towards an off-side. The module *Team-Level Tactics* sets the general tactical conditions of the whole team. Parameters as basic formation, e.g. 4 defenders—4 mid-fielders—2 attackers, offensive or defensive play, wide or narrow play, i.e. playing over the out-fields or not, are determined here. This tactical information is simply stored as parameter settings which the player will use for decision making. A defender must behave in a different way if she has four other teammates playing in the defense than if there are only three other defenders. The role affiliation is stored in the basic world model. This information can then be used by any component.

Utilities

All methods selecting or reasoning about actions, which we discuss in the following subsections, use a measure of utility to rank different possible actions. Here we briefly illustrate what we have in mind.

Consider the situation in Figure 6(a). Here the labels correspond to utilities, which we assume to range between 0 and 1. We assume that utilities are computable in a simple manner. For example, a reasonable utility value for passing a ball to a teammate can be obtained as a function of the trajectory of the ball, its average velocity, the distance of the closest opponent from the ball trajectory, and an average speed of the opponent (see Figure 5). The white player has

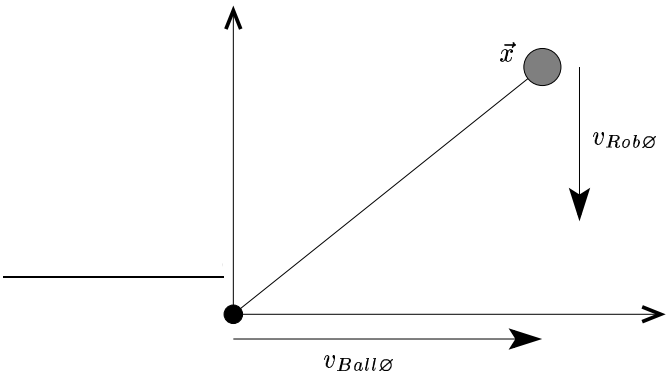
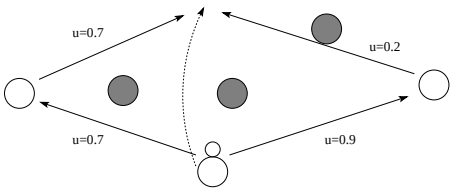


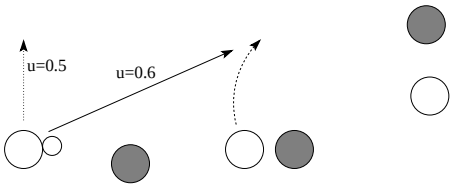
Figure 5: Parameters to compute the utility of a Pass

the opportunity to play a pass (solid line) to the teammate on the left or on the right. If only the next action is considered, then the pass to the right is chosen because the utility value is higher (0.9 vs. 0.8). If several actions are considered, say, to play a double-pass with the white player moving along the dashed line, then passing to the left first seems better because the combined utility of the two passes is higher ($0.7\times0.7>0.9\times0.2$). Now consider two possible situations after the ball was kicked to the left player. In Figure 6(b) the opponent players did not move that much. The recalculated utility based on the present setting of the world is still high enough to play the planned pass. In contrast to this, in Figure 6(c) a player moved into the pass-way, so the utility decreases significantly. Thus the double-pass is abandoned and another action, say, a dribble (dotted line) is selected.

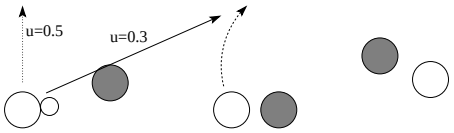
In general, in order to select actions with the highest utility at a given situation, techniques from Markov Decision Processes will be employed (L.Puterman 1994). These have recently been adapted to symbolic reasoning using Golog as well (Boutilier *et al.* 2000).



(a) An initial decision situation



(b) A possible situation after one action has been executed



(c) Another possible situation after one action has been executed

Figure 6: Selecting an action based on utilities

The Reactive Component

The idea for the reactive component is to quickly settle on the next basic ability to execute.

In the Simulation League a new world model is generated every 0.1 seconds. At the end of a cycle an action has to be given to the simulation server, otherwise the player will not do anything. For the mid-size league, this cycle is likely longer, but the right amount still needs to be determined experimentally.

Many different reactive systems have been developed over the recent years. Applicable methods are for instance deci-

sion trees, neural networks or condition-action rules. Our aim is not reinventing reactive soccer play, so we build on the experience of other teams. We have to figure out in the future how useful classified situations in combination with one predefined reaction are. In our first implementation a decision tree will be implemented. A simple schema of such a tree is shown in Figure 7. While these tree will be handcrafted at first, our aim is to eventually learn them, following an approach like (Stone & Veloso 1998) in the simulation league, but adapting it to the mid-size league as well.

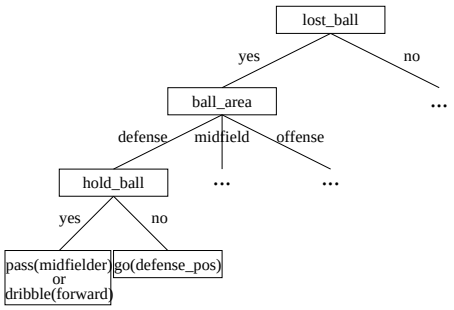


Figure 7: A simple Decision Tree for reactive decisions

All utility values and the world model are available to the RC and can be used to determine an immediate action. A decision tree may start with the question whether we $lost\_ball$ or not. If not we ask for the $ball\_area$ and get one of the answers defense, mid-field, or offense. If the ball is in the defense and I $hold\_ball$ actions like $pass(midfielder)$ or $dribble(forward)$ are defined, in the other cases something like $go(defense\_pos)$. If we just $lost\_ball$ similar questions have to be asked. The actions then would be more like shifting relative to the ball or attacking the opponents' ball-holder directly. The terms $midfielder$ or $defense\_pos$ stand for several players or positions available in the world model. We want to allow for the possibility of having several action choices at a leaf of the decision tree. The one with the highest utility will be given to the ASC.

The Deliberative Component

Golog

We use a variant of the action programming language Golog (Levesque *et al.* 1997; Giacomo, Lesperance, & Levesque 2000) for deliberation. We will not go into any details of the language except to note the following. Golog is based on the Situation Calculus (McCarthy 1963), in particular, the variant proposed by Reiter (Reiter 2001), which includes a solution to the frame problem. Starting with a pre-defined set of primitive actions, which in our case correspond to the abilities of a soccer agent as defined earlier, the user can combine them in Golog to obtain complex actions using control structures known from conventional programming languages such as sequence, if-then-else, while-loops, or recursive procedures. Some examples are given below. We believe that the ability to explicitly program complex actions provides an important advantage in highly dynamic environments compared to classical planning methods. This way it is possible for the designer of the system to provide plan skeletons for, say, different variants of a double pass. The robot can then instantiate these variants for a given situation, project them to assess the various outcomes, and then choose the most promising one for execution. Perhaps most importantly, the complexity of the projection task can be controlled by the user by carefully choosing the plan skeletons.

However, this alone is not enough. In particular, the original Golog lacks the expressiveness to adequately describe actions in continually changing, uncertain environments like RoboCup. Fortunately, recent extensions of Golog by (Grosskreutz & Lakemeyer 2000a; 2000b) have helped overcome this shortcoming. The most prominent novel features are the following:

For one, it is now possible to model *continuous change*, which is useful for spatial reasoning, in particular when reasoning about the trajectories of robots or the ball. Furthermore, the construct $waitFor(\phi)$ plays an important role. For example,

$$waitFor(ballPos = (10.1, 30.4))$$

may stand for the robot waiting until the ball has reached a certain position. This way we are able to express very conveniently *event-driven behavior*, a feature that so far was found only in nonlogic-based robot control languages such as RPL or Colbert (McDermott 1991; Konolige 1997).

Lastly, it is now possible to deal with *uncertainty* by using the probabilistic construct $prob(p, \sigma_1, \sigma_2)$. To see its use, consider the following example program

$$[kick(r_0, r_1),$$
$$prob(0.75, set\_kick\_success, set\_kick\_failed),$$
$$if(kick\_failed, goDefense, goOffense)].$$

The robot r_0 considering this plan is supposed to kick the ball to r_1. With a probability of 0.75 the kick is successful, otherwise the ball is lost. Depending on the outcome of the action the robot will react.

Applying Golog to the Soccer Domain

The task of our deliberative component is to find a plan consisting of only a few actions that lead to a known game situation (compare Figure 3)—that is, to find a transition from one game situation to another.[2] Since the Golog interpreter, a variant of IndiGolog (de Giacomo & Levesque 1999), is relatively slow compared to the reactive decision cycle, the DC has to run asynchronously. This has the advantage that the DC has not need to provide a new plan at every decision cycle.

The basic abilities and situation descriptions are the same as used for the reactive layer. Once the robot is in a known game situation, one can make use of pre-defined action scripts or schemata in the form of Golog- procedures. Given the situation being blocked by an opponent in front

$$[\{ball : midfield, right\}\{[r_0 : 1, 6]\}\ \{[o_0 : 3, 1]\}$$
$$\{[r_1 : 4, 3]\}]$$

[2] Due to the dynamics of the game and the uncertainties involved, it certainly does not make sense to plan from the kick-off to scoring a goal.

(compare Figure 6 or 3), the action sequence for a single robot supposed playing a double-pass might be something like $[stop\_ball,\ pass(r_1),\ go(pos\_behind\_opp, 90),\ waitFor(haveBall)]$.[3] Due to this plan the robot should stop the ball, pass it to robot r_1 defined by the situation classification and runs (with 90% of maximum strength) to a point behind the blocking opponent. Taking a closer look at the $WaitFor$-Construct we have to take the behavior of the other robots, especially the own ones, into account during planning. A predefined script for two involved robots might be like this:

$$[stop\_ball(r_0), pass(r_0, r_1),$$
$$if(lost\_ball, attack\_ballholder(closest\_player\_to\_ball)),$$
$$waitFor(r_1, haveBall)\ go(r_0, pos\_behind\_opp, 90),$$
$$stop\_ball(r_1), pass(r_1, pos(r_0)),$$
$$if(lost\_ball, attack\_ballholder(closest\_player\_to\_ball)),$$
$$waitFor(r_0, haveBall)].$$

The planner is now able to fill the variables of these scripts with legal values by projecting from the present into the future. Every atomic action is evaluated with respect to the assumed world model. Only $stop\_ball(r_0)$ is evaluated with respect to the actual situation. If more than one plan has been defined for one situation, the one with the best overall utility is passed to the ASC. With the help of Golog programming constructs the predefined scripts can be made more flexible. The plan above contains an if-statement whether the pass was successful or not. If the ball is reconquered immediately by $attack\_ballholder$, the plan execution can be continued, otherwise the game situation changed so significantly that the utility for the next action-situation pair will be decreased drastically and the plan execution will be stopped. If the plan succeeds completely another known game situation arises. Overall our approach can be seen as a transition function between known game situations. If one action fails in between the world evolved differently from the assumption. The utility decreases significantly if the present action of the plan is not applicable anymore. Therefore the plan execution will be stopped by the ASC. Further details will be described in the next paragraph.

One difficulty is modeling the behavior of teammates and the opposing team. The own teammates view the situation from the ball-holder's point of view. Thus the robot in area $[4, 3]$ in Figure 3 is able to notice being robot r_1 in the script. The ball-holder has to execute all primitive actions starting with parameter r_0 and the other one all actions starting with r_1. To receive a common decision for the same script the robots need to have a similar world model. Initial tests have shown that our situation abstraction mechanism supports this requirement. Modeling the behavior of opponents is more sophisticated. Several methods have to be evaluated here. Some possibilities are (1) extending the present speed and direction for the next decision loops, (2) the nearest opponents always try to attack our ball-holder or the ball itself or (3) they behave like our own robots.

Another open question is when to initiate planning. As we already pointed out, the time it takes to generate a plan is independent of the reactive decision cycle. One possibility is initiating a new planning cycle immediately after a plan was generated. The disadvantage is the great expense on futile planning work, the advantage is having always a quite up-to-date plan. The other option is starting planning only if something unexpected happens, e.g our team lost the ball.

[3] If this situation occurs in the defense or the offense other scripts may be defined.

Action Selection

An reactive action by the RC and a plan by the DC are given to the Action-Selection method. This module has to decide which action—reactive or the next in the plan—is selected for execution.

While the RC action has already been evaluated on the present world model, the planned action is based on approximated assumptions how the world evolves when certain actions take place. Therefore the utility of the next action from the plan has to be evaluated again on the basis of the current situation.

Several cases can arise. The simplest case is only having a reactive action (RA) available and no valid plan. As defined the RC has always to provide a basic ability for the next decision cycle. The RA will be chosen. If a plan is given as well, we have to take a closer look. If the world somehow evolved as expected during deliberation the plan is valid and the next action of the plan is chosen for execution. Valid in this case means that the utility is close to the one calculated during planning. If the world changed differently from what was expected and the next action of the plan is not applicable, the newly computed utility will decrease because the action-situation pair will be rated significantly lower than during the planning loop. Therefore the reactive action will be chosen. An example was already given in figure 6.

Discussion and Future Work

In this paper we proposed a robot architecture which is intended for highly dynamic environments like robotic soccer and which combines reactive action selection and Golog-style planning. We are currently refining the architecture and implementing it both in the simulation league and on custom-built mid-size robots like the one in Figure 1, which we obtained very recently. Many issues still need to be resolved. We mentioned already that the question when and how robots should communicate is still open. Also, we only have preliminary ideas about how to classify a given game situation obtained from sensors and other world knowledge. Running actual experiments will likely help refining our ideas here. Similarly, experiments will help us develop better heuristics for the reactive component. Ultimately, machine learning techniques should be employed for this task. As for the deliberative component, we only have initial versions of Golog programs, which need to be tested and refined. Whether to run Golog concurrently at all times or whether to initiate plan selection on demand needs to be resolved as well. Finally, as one of the reviewers remarked, there seem to be interesting connections between the Golog programs we envision and MDPs or POMDPs, an issue that needs to be investigated more carefully in the future. (See also (Boutilier *et al.* 2000) for a version of Golog that explicitly integrates MDPs.)

Regarding related work, our architecture is an extension of the three-layered architectures proposed in (Firby, Propopowicz, & Swain 1998; Gat 1998). Sahota (Sahota 1994) proposes a kind of reactive deliberation, which on the surface looks very related. Even the application domain is the same. On closer inspection, however, the deliberative component used is actually much closer to our reactive component, since its sole purpose is to choose a single next ability such as "move to mid-field" or "defend goal." In other words, no plans of length greater than one are considered. In (Murray, Obst, & Stolzenburg 2001), logic programming is used for the implementation of soccer agents. Deliberation is mainly confined to a form of qualitative spatial reasoning.

Perhaps the most closely related approach to our work is (Jensen & Veloso 1998). Here, the simulation-league soccer agents also mix reactive and deliberative decision making. Among other things, the authors propose that an agent switches from deliberation to reactive control when an opponent moves too close to the agent. This fits well with our notion of dropping the deliberative plan once the world changes too much compared the world model used by Golog. Despite these similarities, there are significant differences as well. For one, Jensen and Veloso use PRODIGY (Veloso *et al.* 1995), a nonlinear planner, which runs as a central deliberative service and which derives a multi-agent plan for the whole team and then sends each agent its corresponding subplan. To make this work, severe restrictions in the expressiveness of the plan language are necessary. For example, it is assumed that every action takes the same unit of time, which seems to limit the usefulness of the plans derived. Besides, employing a full-scale planner like PRODIGY imposes a heavy burden computationally.

More recently, (Burkhard 2001) describes work in progress involving a so-called *double-pass architecture*, which features a *deliberator*, which produces partial plans using Case Based Reasoning techniques, and an *executor*, which asynchronously selects actions when needed, taking into account the current plan and information about the world. We intend to compare our approach to this one in more detail in the future.

Another related approach is taken in the WITAS project (Doherty *et al.* 2000). Here the control rests strictly with the reactive component, which is understandable given the unmanned aerial vehicle scenario. The deliberative component, which includes a planner, among other things, is only activated on demand when the reactive controller cannot achieve a goal. This is different from our architecture, since we never rely on the deliberative component producing a plan, mainly because the environment of a soccer playing robot requires constant vigilance on the part of the robot.[4]

On a final note, one may ask why there needs to be a strict separation between the reactive and deliberative component. After all, Golog, at least in principle, can be used to program strictly reactive behavior.[5] The short answer is that the Golog implementation, which is an interpreter running under Prolog, just does not have the performance needed for quick reactions. In other words, the reactive component is written in C, which also opens the way to use any special-purpose non-symbolic problem solver. Whether there are deeper reasons why reactive control should be separated from logic-based control remains to be seen.

Acknowledgments

We thank the anonymous referees for their valuable comments. This work was partly supported by Grant No. LA 747/9-1 by the German Science Foundation (DFG) and a grant by the NRW Ministry of Education and Research (MSWF).

References

Baral, C., and Son, T. 1998. Relating theories of actions and reactive control. In *Electronic Transactions on Artificial Intelligence*, volume 2(1998), Issue 3-4, 211–271. http://www.ep.liu.se/ej/etai/1998/008/: ETAI.

Boutilier, C.; Reiter, R.; Soutchanski, M.; and Thrun, S. 2000. Decision-theoretic, high-level agent programming in the situation calculus. In *AAAI'2000*, 355–362.

Burgard, W.; Cremers, A.; Fox, D.; Hähnel, D.; Lakemeyer, G.; Schulz, D.; Steiner, W.; and Thrun, S. 2000. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence* 114(1-2).

Burkhard, H.-D. 2001. Mental models for robot control. Dagstuhl Workshop on Plan-based Control of Robotic Agents.

de Giacomo, G., and Levesque, H. 1999. An incremental interpreter for high-level programs with sensing. In Levesque, H., and Pirri, F., eds., *Logical Foundations for Cognitive Agents*. Springer. 86–102.

Doherty, P.; Granlund, G.; Kuchcinski, K.; Sandewall, E.; Nordberg, K.; Skarman, E.; and Wiklund, J. 2000. The witas unmanned aerial vehicle project. In *ECAI'2000*.

Firby, R. J.; Propopowicz, P. N.; and Swain, M. J. 1998. The animate agent architecture. In Kortenkamp, D.; Bonasso, R.; and Murphy, R., eds., *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*. MIT Press.

Gat, E. 1998. Three-layered architectures. In Kortenkamp, D.; Bonasso, R.; and Murphy, R., eds., *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*. MIT Press.

Giacomo, G. D.; Lesperance, Y.; and Levesque, H. J. 2000. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121:109–169.

Grosskreutz, H., and Lakemeyer, G. 2000a. cc-golog: Towards more realistic logic-based robot controllers. In *AAAI/IAAI*, 476–482.

[4] Granted, a helicopter cannot stay idle either, but it can decide to hover while deliberation is in progress.

[5] See also (Baral & Son 1998) for a logical reconstruction of reactive behavior.

Grosskreutz, H., and Lakemeyer, G. 2000b. Turning high-level plans into robot programs in uncertain domains. In *ECAI'2000*.

Jensen, R., and Veloso, M. 1998. Interleaving deliberative and reactive planning in dynamic multi-agent domains. In *Proceedings of the AAAI Fall Symposium on on Integrated Planning for Autonomous Agent Architectures*. AAAI Press.

Konolige, K. 1997. Colbert: A language for reactive control in sapphira. In *KI'97*, volume 1303 of *LNAI*.

Levesque, H. J.; Reiter, R.; Lesprance, Y.; Lin, F.; and Scherl, R. 1997. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming* 31:59–84.

L.Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley.

McCarthy, J. 1963. Situations, actions and causal laws. Technical report, Stanford University. Reprinted 1968 in Semantic Information Processing (M.Minske ed.), MIT Press.

McDermott, D. 1991. A reactive plan language. Research Report YALEU/DCS/RR-864, Yale University.

Murphy, R. R. 2000. *Introduction to AI Robotics*. MIT Press.

Murray, J.; Obst, O.; and Stolzenburg, F. 2001. Towards a logical approach for soccer agents engineering. In Stone, P.; Balch, T.; and Kraetzschmar, G., eds., *Robot Soccer World Cup IV*, LNAI 2019, 199–208. Springer-Verlag.

Myers, K. L. 1996. A procedural knowledge approach to task-level control. In Drabble, B., ed., *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, 158–165. AAAI Press.

Noda, I.; Matsubara, H.; Hiraki, K.; and Frank, I. 1998. Soccer server: a tool for research on multi-agent systems. *Applied Artificial Intelligence* 12.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.

RoboCup. 2002. http://www.robocup.org.

Sahota, M. K. 1994. Reactive deliberation: An architecture for real-time intelligent control in dynamic environments. In *AAAI'94: Proceedings 12th National Conference on AI*, 1303–1308. American Association for Artificial Intelligence.

Simmons, R.; Goodwin, R.; Haigh, K.; Koenig, S.; O'Sullivan, J.; and Veloso, M. 1997. Xavier: Experience with a layered robot architecture. *ACM SIGART Bulletin* Intelligence 8 (1–4).

Stone, P., and Veloso, M. 1998. A layered approch to learning client behaviors in the robocup soccer server. *Applied Artificial Intelligence* 12:165–188.

Stone, P.; Riley, P.; and Veloso, M. 2000. The CMUnited-99 champion simulator team. In Veloso, M.; Pagello, E.; and Kitano, H., eds., *RoboCup-99: Robot Soccer World Cup III*. Berlin: Springer Verlag.

Thielscher, M. 2002. Programming of reasoning and planning agents with FLUX. In Fensel, D.; McGuinness, D.; and Williams, M.-A., eds., *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 435–446. Toulouse, France: Morgan Kaufmann.

Thrun, S.; Bennewitz, M.; Burgard, W.; Dellaert, F.; Fox, D.; Haehnel, D.; Rosenberg, C.; Roy, N.; Schulte, J.; and Schulz, D. 1999. Minerva: A second-generation museum tour-guide robot. In *ICRA'99*.

Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence* 7(1):81–120.