AllemaniACs 2004 Team Description

Alexander Ferrein, Christian Fritz, and Gerhard Lakemeyer

Knowledge Based Systems Group
RWTH Aachen,
Aachen, Germany
{ferrein, fritz, gerhard}@cs.rwth-aachen.de

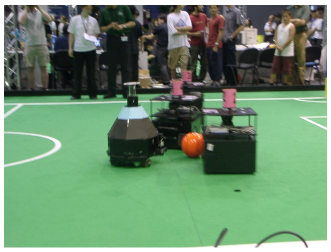
Abstract. This paper describes the scientific goals of the ALLEMANIACS 2004 MID-SIZE soccer team. We present our robot platform and sketch out our software system. In an extended example we show some features of the high-level robot programming language GOLOG which we use for modeling the behavior of our robot .

1 Introduction

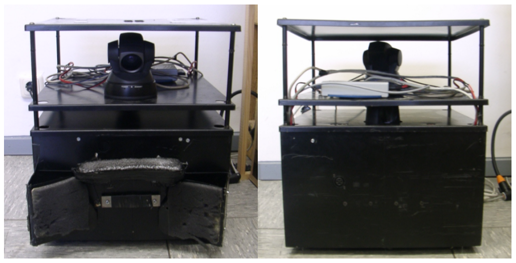
ROBOCUP's MID-SIZE league has high requirements to system designers. Researchers focus on the fields of sensors and actuators, computer vision, communication, and system integration, just to mention a few, under the strict real-time constraints given. Another issue is the decision making process of a robot and how a team of robots can be coordinated, how cooperative team-play can be achieved. Thus, the robot should not only think about its own actions but also take teammates into account when making plans for future actions. Therefore, the decision module of the robot must be able to reason about actions and their effects and project or simulate them into the future. For our decision making module we use a variant of GOLOG, a logic-based programming language which combines explicit agent programming as in imperative languages with the possibility to reason about actions and their effects. For making optimal decisions we use an decision-theoretic planning extension of GOLOG for the decision making in our high-level control.

2 Robot Hardware

Regarding the experiences of other teams that "off-the-shelf" robots must be completely re-engineered for ROBOCUP's MID-SIZE to be competitive, we decided to develop the platform on our own [1]. The intention was to develop robots competitive in ROBOCUP which can also be used in the office domains of service robotics applications. The platform has a size of 39 cm $\times$ 39 cm $\times$ 40 cm (Fig. 1). For power supply we have two 12 V lead-gel accumulators with 15 Ah each on-board. The battery power lasts for approximately one hour at full charge. The robot has a differential drive, the motors have a total power of



(a) A scene from the Championships in Padova; AllemaniACs in the defense (right-hand side)



(b) The front and rear view of an AllemaniACs player

Fig. 1. The AllemaniACs team

2.4 kW. This power provides us with a top speed of 3 m/s and 1000°/s by a total weight of approximately 50 kg.

On-board we have two Pentium III PC's at 933 MHz running Linux, one equipped with a frame-grabber for a Sony EVI-D100P camera mounted on a pan/tilt unit. Our other sensor is a 360° laser range finder with a resolution of 0.75 degree at a frequency of 20 Hz. For communication a WLAN adapter based on IEEE 802.11b is installed.

3 Robot Control Software

In this section we give a brief overview over the software components we use to run our robots. We distinguish between the software running locally on each robot and the control software on an external computer to control the whole team.

3.1 On-board Software

For our software architecture we are using a three layered architecture consisting of a low-level layer where sensory inputs and actuator outputs are controlled, a mid-level where modules like collision avoidance and localization are located and a high-level controller for making plans about future courses of actions.

Fig. 2 shows our software architecture. For the low-level control task there exist the modules *motor* for motor control, *laser* which receives the data from the laser scanner, *camera* for supplying the video stream and controlling the pan/tilt unit, and *kicker* which controls the kicking device.

In the mid-level this data is used for robot control. One central module is the collision avoidance module *colli*. This module uses the data provided by the

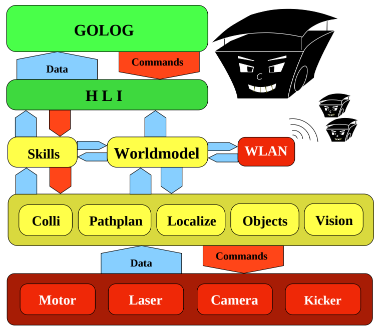


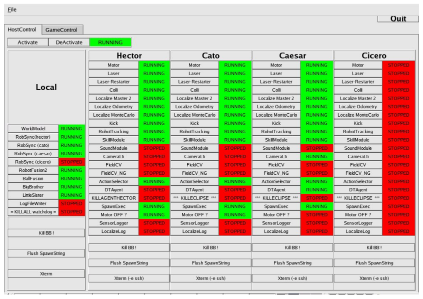
Fig. 2. AllemaniACs software system

laser scanner. With a frequency of 20 Hz *colli* creates an occupancy map from the laser reading. In the occupancy grid map an A^* search to a given target point is performed calculating a collision free path to that target point. The path-planner follows the same idea but, moreover, has world model information like the position of the teammates on the field.

For self-localization our software provides the *localize* module using the 360° laser range finder following a Monte-Carlo approach. As for RoboCup it is very important to have continuity in the position updates we use robot's odometry between two position estimates by the localization module. The *localize* module uses KLD sampling and cluster based sampling (e.g. cf. [2,3]). As for now we use laser and odometry as sensory inputs we will incorporate information from the vision module in RoboCup2004.

The module *objects* provides us with information about the objects on the field. The object recognition is done by the laser scanner. The *localize* module classifies dynamic objects. This information is used by the object recognition module to calculate the position of dynamic objects by clustering the appropriate laser beams. The disadvantage with this method is that it is not possible to estimate the orientation of the dynamic objects. In the future, the scan matching method will be used for providing also information about the orientation.

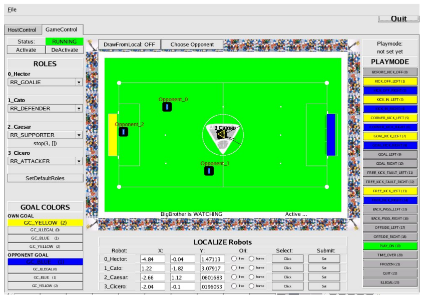
The ball, goals and flagposts are localised with the *vision* module using color segmentation and knowledge about the objects' shape. Colour segmentation is inexpensive, but the mapping between single objects and their colours or chrominances, if reduced to two dimensions, must be known. For each shape detected object, we compute the chrominance histogram. The histograms are combined based on the Bayes Theorem to obtain a lookup table [4], which is used for colour segmentation. To profit from the speed of colour segmentation, shape detection is only applied to regions containing the interesting object's colour. The circular ball is detected by a randomized hough transform. The goals and flagposts are modelled as quadrilaterals, which are bordered by straigt lines. Object detection takes about 50 ms on a Pentium III with 933 MHz. The entire vision system, including a segmentation and colour recalibration step, runs in 10 Hz.



Local	
WorldModel	RUNNING
RobSync(hector)	RUNNING
RobSync (cato)	RUNNING
RobSync (caesar)	RUNNING
RobSync (cicero)	STOPPED
RobotFusion2	RUNNING
BallFusion	RUNNING
BigBrother	RUNNING
LittleSister	RUNNING
LogFileWriter	STOPPED
= KILLALL watchdog =	STOPPED

Hector		Cato		Caesar		Cicero	
Motor	RUNNING	Motor	RUNNING	Motor	RUNNING	Motor	STOPPED
Laser	RUNNING	Laser	RUNNING	Laser	RUNNING	Laser	STOPPED
Laser-Restarter	RUNNING	Laser-Restarter	RUNNING	Laser-Restarter	RUNNING	Laser-Restarter	STOPPED
Colli	RUNNING	Colli	RUNNING	Colli	RUNNING	Colli	STOPPED
Localize Master 2	RUNNING	Localize Master 2	RUNNING	Localize Master 2	RUNNING	Localize Master 2	STOPPED
Localize Odometry	RUNNING	Localize Odometry	RUNNING	Localize Odometry	RUNNING	Localize Odometry	STOPPED
Localize MonteCarlo	RUNNING	Localize MonteCarlo	RUNNING	Localize MonteCarlo	RUNNING	Localize MonteCarlo	STOPPED
Kick	RUNNING	Kick	RUNNING	Kick	RUNNING	Kick	STOPPED
RobotTracking	RUNNING	RobotTracking	RUNNING	RobotTracking	RUNNING	RobotTracking	STOPPED
SkillModule	RUNNING	SkillModule	RUNNING	SkillModule	RUNNING	SkillModule	STOPPED
SoundModule	STOPPED	SoundModule	STOPPED	SoundModule	STOPPED	SoundModule	STOPPED
CameraLti	STOPPED	CameraLti	STOPPED	CameraLti	RUNNING	CameraLti	STOPPED
FieldCV	STOPPED	FieldCV	STOPPED	FieldCV	STOPPED	FieldCV	STOPPED
FieldCV_NG	STOPPED	FieldCV_NG	STOPPED	FieldCV_NG	STOPPED	FieldCV_NG	STOPPED
ActionSelector	RUNNING	ActionSelector	RUNNING	ActionSelector	RUNNING	ActionSelector	STOPPED
DTAgent	STOPPED	DTAgent	STOPPED	DTAgent	RUNNING	DTAgent	STOPPED
KILLAGENTHECTOR	STOPPED	*** KILLECLIPSE ***	STOPPED	*** KILLECLIPSE ***	STOPPED	*** KILLECLIPSE ***	STOPPED
SpawnExec	RUNNING	SpawnExec	RUNNING	SpawnExec	RUNNING	SpawnExec	STOPPED
Motor OFF ?	RUNNING	Motor OFF ?	RUNNING	Motor OFF ?	STOPPED	Motor OFF ?	STOPPED
SensorLogger	STOPPED	SensorLogger	STOPPED	SensorLogger	STOPPED	SensorLogger	STOPPED
LocalizeLog	STOPPED	LocalizeLog	STOPPED	LocalizeLog	STOPPED	LocalizeLog	STOPPED

(a) Software Control Window



Status: RUNNING

ROLES

0_Hector: RR_GOALIE

1_Cato: RR_DEFENDER

2_Caesar: RR_SUPPORTER
stop(3, [])

3_Cicero: RR_ATTACKER

GOAL COLORS

OWN GOAL: GC_YELLOW (2)

OPPONENT GOAL: GC_BLUE (1)

Playmode: not set yet

PLAYMODE

LOCALIZE Robots

Robot:	X:		Y:	Ori:	Select:	Submit:
0_Hector:	-4.84	-0.04	1.47113	○ free ○ home	Click	Set
1_Cato:	1.22	-1.82	3.07917	○ free ○ home	Click	Set
2_Caesar:	-2.66	1.12	.0601683	○ free ○ home	Click	Set
3_Cicero:	-2.04	-0.1	.0196053	○ free ○ home	Click	Set

(b) Game State Window

Fig. 3. The RoboCup Control Center

The *skill* module encapsulates the robot behavior system. Here the basic behaviors like kicking and dribbling are implemented. This module has access to the modules which activate the actuators needed to perform the soccer behaviors as well as to the relevant world model information.

The *world model* consists of information like the ball and player position as well as some other internal state information. We provide two kinds of world model, a local one which is constructed from the own perception and a global one which is the result of a world model fusion on an external computer. The global world model is subject of the next section.

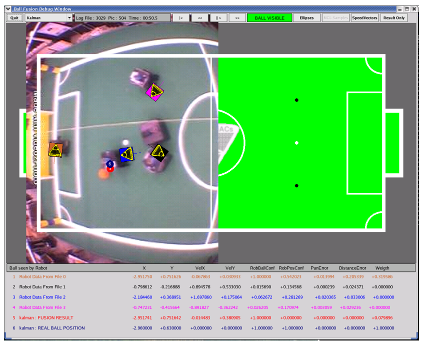
For interprocess communication we use a blackboard communicating via shared memory internally. For communication between different computers the blackboard uses UDP.

HLI stands for high-level interface and builds the interface between our high-level control and the rest of the software system. The name comes from the well-known Rhino software system [5] and we use this name for historical reasons. As our high-level control is implemented in Prolog the interface wraps the most important functions to the rest of the system which is implemented in C++.

3.2 External Control Software

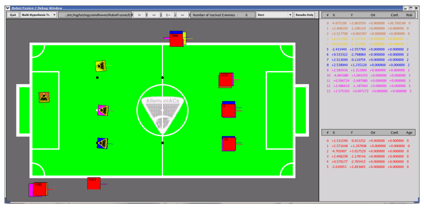
As mentioned in the previous section we use some modules running on an external computer for controlling the robot team and calculating parts of the global world model we use.

For controlling our team we implemented the RoboCup Control Center (*rccc*). This module runs on an external computer and is connected to the robots via WLAN. With this module we are able to control our software and get information about the game state. Fig. 3 shows a screen shot of the *rccc*. Fig. 3(a) shows the software control window. Here, every module on each robot can be started or stopped. Moreover, we get information about the status of particular module,



	Ball seen by Robot	X	Y	VelX	VelY	RobBallConf	RobPosConf	PanError	DistanceError	Weigth
1	Robot Data From File 0	-2.951750	+0.751626	-0.067863	+0.030933	+1.000000	+0.542023	+0.013994	+0.205339	+0.319586
2	Robot Data From File 1	-0.798612	-0.216888	+0.894578	+0.533030	+0.015690	+0.134568	+0.000239	+0.024371	+0.000000
3	Robot Data From File 2	-2.184460	+0.368951	+1.697860	+0.175064	+0.062672	+0.281269	+0.020365	+0.033006	+0.000000
4	Robot Data From File 3	-0.747231	-0.415664	-0.891827	-0.362242	+0.026205	+0.170974	+0.003059	+0.029236	+0.000000
5	kalman : FUSION RESULT	-2.951741	+0.751642	-0.014483	+0.380905	+1.000000	+0.000000	+0.000000	+0.000000	+0.079896
6	kalman : REAL BALL POSITION	-2.960000	+0.630000	+0.000000	+0.000000	+1.000000	+1.000000	+0.000000	+0.000000	+1.000000

(a) Ball fusion



#	X	Y	Ori	Conf.	Rob
0	-4.675100	+3.063550	+0.000000	+36.799198	0
1	+2.400150	-1.106110	+0.000000	+0.000000	0
2	+2.517700	+0.002397	+0.000000	+0.000000	0
3	+2.425390	-1.115330	+0.000000	+0.000000	1
4	+0.669309	-2.753830	+0.000000	+0.000000	1
5	-2.431440	+2.557760	+0.000000	+0.000000	2
6	+0.533322	-2.798060	+0.000000	+0.000000	2
7	+2.513690	-0.110754	+0.000000	+0.000000	2
8	+2.538940	+1.235120	+0.000000	+0.000000	2
9	+2.583430	+1.312890	+0.000000	+0.000000	3
10	-4.845080	+3.044350	+0.000000	+0.000000	3
11	+0.566724	-2.687080	+0.000000	+0.000000	3
12	+2.486610	-1.187440	+0.000000	+0.000000	3
13	+2.575350	+0.047173	+0.000000	+0.000000	3

#	X	Y	Ori	Conf.	Age
0	+2.531590	-0.013252	+0.000000	+0.000000	0
1	+2.571648	+1.297606	+0.000000	+0.000000	0
2	-4.703907	+3.017529	+0.000000	+0.000000	0
3	+2.446238	-1.178746	+0.000000	+0.000000	0
4	+0.570177	-2.765415	+0.000000	+0.000000	0
5	-2.630953	+2.833665	+0.000000	+0.000000	0

(b) Robot Tracking

Fig. 4. World Model Fusion

e.g. if it is running or if it crashed for some reasons. The GUI is configurable via an XML file for being able to easily integrate new modules to the software control. Fig. 3(b) shows the game state window. For indicating the robot the current game state we can choose between different play modes like *play on* or *kick off left* (right-hand side of Fig.3(b)). For getting information about the robot's internal state we visualized the robot's pose estimate and the opponent's pose estimates as well as the current action the robot is performing.

For enhancing our global world model, we provide modules for fusing the robot's perceptions of the ball and tracking opponent players. Fig. 4(a) shows the ball fusion module. The perceptions from each robot can be fused using several methods like Kalman Filter or Monte Carlo. The figure shows a scene from a log file from a test match against the Philips team. Note, that the camera picture was supplementary added to the log to verify the accuracy of the applied methods. Fig. 4(b) shows the opponent tracking module. we use an implementation of Multiple Hypothesis Tracking from [6,7] for keeping tracks of opponents recognized by the *object* recognition module over time.

4 Decision Making and High-level Control

Our research topic is the decision making of intelligent agents. For us, this means that the robot should be able to make plans about future courses of actions instead of deciding what to do based only on the current situation. For this purpose we use an extension of the logic-based robot control language GOLOG. As planning is in general costly and in robotic soccer making a sub-optimal decision is better than making no decision about which action to take, we believe that some form of reactive control is still needed in order to keep reactivity. Our aim is therefore to combine planning with reactive action selection to get the best performance for the high-level control of the robot.

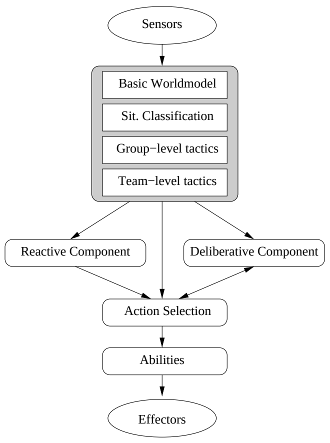


Fig. 5. DR-Architecture

In the following we will sketch our architecture for the high-level control before giving an introduction to the robot control language GOLOG. We then give a detailed example of the kind of planning we use for high-level control by a situation from the world championships 2003 in Padova.

4.1 The DR-Architecture

While deliberation has many advantages for decision making of an agent, it has the disadvantage of being slow compared to generating actions in a reactive fashion. In [8] we proposed a hybrid architecture which allows the combination of deliberation with reactivity. Here, we give a brief overview of our architecture. For more information we refer to [8].

Figure 5 shows the DR-Architecture. From the sensory input we build our world model (gray box in Figure 5). It contains data like the positions of teammates, opponents, or the ball (*Basic World Model*), classification data about the current situation, e.g. good possible pass partners (*Situation Classification*) and group resp. team-level information, e.g. the basic formation of the team (*Group- and Team-level tactics*).

The decision module in the DR-Architecture is divided into three modules. To be able to settle an action immediately the *Reactive Component* computes the next action to be executed based on the current game situation. The *Deliberative Component* calculates a plan projecting into the future choosing among the possible action sequences. Section 4.2 covers this in greater detail. Having an immediate action and a plan concurring for executing one needs to decide which of both to use. A simple strategy for the *Action Selection* is to use the plan whenever there exists one or execute the action provided by the Reactive Component otherwise. More complex approaches have been tested in [9] and are under further research.

For the Reactive Component we made some good experiences with the C4.5 decision tree learning method [10] in our Simulation league team [11]. One goal is to apply C4.5 also for our middle size team.

Abilities are the basic actions the agent can perform in the world. These are defined in the *skill* module (cf. Section 3.1).

4.2 The agent specification language GOLOG

For specifying our high-level control we use a variant of the logic-based high-level agent programming language GOLOG [12]. GOLOG is a language based on the situation calculus [13]. Over the past years many extensions like dealing with concurrency, exogenous and sensing action, a continuous changing world and probabilistic projections (simulation) [14,15,16] made GOLOG an expressive robot programming language. We integrated those features in our ICPGOLOG interpreter [17]. For the decision making, we further integrated a planning module into GOLOG which chooses the best action to perform by solving a Markov Decision Process (MDP) (we refer to [18] for reading on MDP and to [19] on integrating MDPs into GOLOG).

In the following we give an overview of some of the features of ICPGOLOG not going into details. For an extended example of how a multi-agent plan for a double pass can be modeled and executed in the SIMULATION league in ICPGOLOG we refer to [17]. The ICPGOLOG language features are:

- Sequence: a_1, a_2
- Nondeterministic Choice: $a_1; a_2$
- Solve an MDP: $solve(p, h)$, where p is a GOLOG program and h is the horizon up to which the MDP is solved
- Test: $?(c)$
- Event-Interrupt: $waitFor(c)$
- If-then-else: $if(c, a_1, a_2)$
- While-loops: $while(c, a_1)$
- Condition-bounded execution: $withCtrl(c, a_1)$
- Concurrent actions: $pconc(a_1, a_2)$
- Probabilistic actions: $prob(val_{prob}, a_1, a_2)$
- Probabilistic (offline) projection: $pproj(c, a_1)$
- Procedures: $proc(name(parameters), body)$

For modeling our high-level agent we have to specify the primitive actions the robot can perform together with their preconditions and effects. These are actions like $goto(x, y, \theta)$, *intercept_ball*, or $dribble\_to(x, y)$. Besides primitive action there are exogenous and sensing actions. With exogenous actions external events, e.g. a game restart, can be modeled. With the *pconc*-statement programs can be executed concurrently, the *pproj*-statement starts a simulation of a ICPGOLOG program. The robot can simulate several different alternatives which actions to perform and then choose the most promising. Another important feature is the possibility to model uncertainty by the *prob*-statement. It is possible to model for example the success of a *goto* action or even build a model for the behavior of the opponent goal-keeper. The *solve* statement initiates the solving of the MDP formulated by the program p up to a fixed horizon h. It returns a policy, i.e. an optimal course of action given p the robot can perform regarding the current game situation.

4.3 Decision Making at RoboCup 2003

The offense's behavior of the AllemaniACs robots was totally determined by the MDP-like planning of our GOLOG dialect. The robot made plans of length four including teammate's and opponent's behavior.

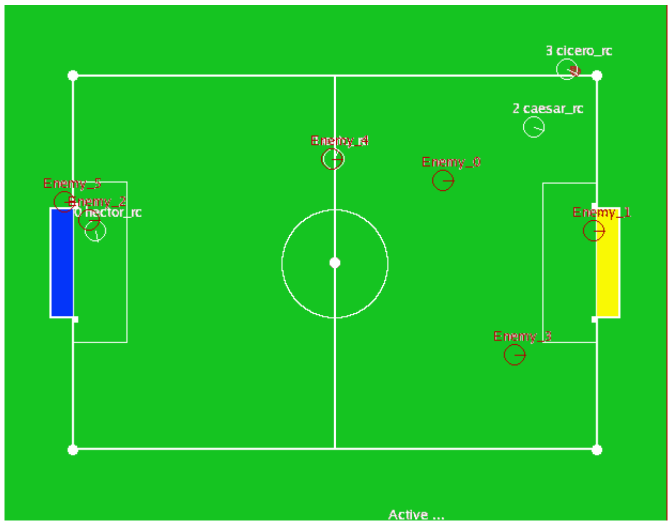
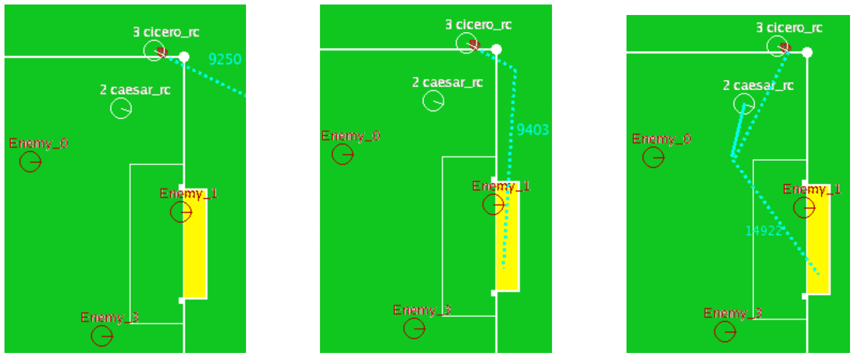


Fig. 6. A situation that occurred during a match against *ISePorto*: One of our robots (Cicero) has the ball and the supporter (Caesar) is at its place.

In the following we present an example of such decision making in depth. The used data was taken from log files recorded during a match against Ise-Porto. Figure 6 shows the view of the world in the example situation as logged during the game. This means, the positions of the ball (in the upper right), our robots (in white), and the opponents (red) are estimated based on the fused information from all of our robots.[1] In this situation most of the dribbling actions are considered impossible: The robot has only limited control over the ball. In particular, it is not possible to drive in circles with low radius without loosing the ball. Thus, we set the precondition for dribbling as depending on the angle

[1] We remark that these positions in general are quite erratic due to time lags in the remote connections and due to sensor inaccuracies. This is another source of difficulties for decision making.

to the target point. In this situation the attacker (Cicero) considered, among others, the following alternatives, which are illustrated in Figure 7[2]:



(a) (b) (c)

Fig. 7. (a) a straight kick (b) a move-kick towards the empty opponents goal corner (c) team-play

- *a straight kick*
  (Figure 7 (a)) Kicking in this situation would shoot the ball out of bounds. The associated reward is based on the intersection with the goal line and is computed as 9250.[3]
- *a move-kick to the goal*
  (Figure 7 (b)) The best corner to shoot at is obviously the right one. However, the model describes a move-kick by driving one meter to the front and then accelerating the ball towards the target.[4] Again the ball would go out of bounds, achieving a similar reward as the kick (9403).
- *team-play*
  (Figure 7 (c)) The action eventually taken was to turn with the ball, pushing it in front of the supporter.

Let us see what lead to this decision. For readability we only use integer values. The initial situation has a reward of 4557 and turning with the ball will certainly (probability = 1.0) lead to a situation with reward 4169 (see Figure 8 (a)). Next, the supporter (Caesar) is assumed to intercept the ball. This is only successful

[2] All presented data is taken from log-files that were recorded during the game.
[3] We only use integer numbers here, to improve readability.
[4] Of course, this model again is very heuristic and is based solely on observations during testing.

with probability 0.2. The reward for the new situation is unchanged, since the ball is not moved – in this model (see Figure 8 (b)). However, the intercept has negative costs, as the ball is not far for the supporter and it is in front of the opponents goal, where an intercept always seems a good idea. Alternatively, interception by the attacker is considered. But since that player is farer away from the ball, the costs for that are higher (see Figure 9). Finally, the supporter now having the ball is assumed to score a goal by performing the move-kick skill (see Figure 8 (c)). The resulting situation, with the ball in the opponents goal, gets a reward of 10000. The move-kick also has negative costs of 70. This is to encourage the robot to generally shoot more often. Overall an expected reward of 14922 is computed for this policy. Figure 9 shows the resulting decision tree

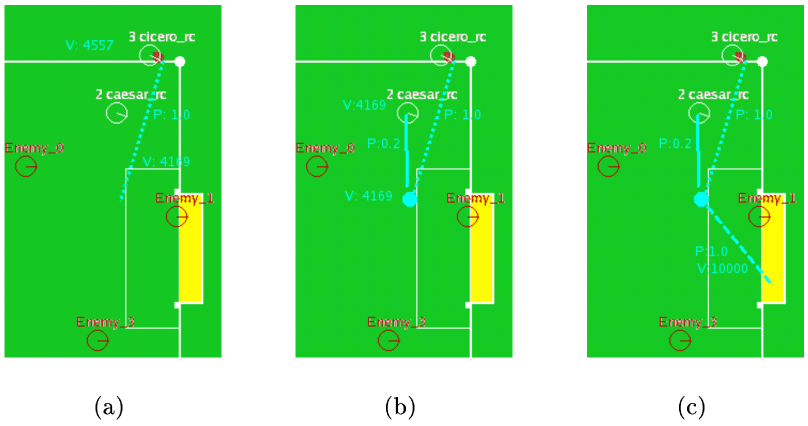


Fig. 8. Team-play

for the three described alternatives. The optimal policy is marked by a thick line. The real decision tree was much larger (about factor 4). We pruned it due to space reasons.Overall decision making in this situation took 0.61 seconds.

In general, the time the attacker spent on projection depended highly on the number of alternatives that were possible. The most significant difference it made whether the ball was kickable or not (all times in seconds):

	examples	min	avg	max
without ball	698	0.0	0.094	0.450
with ball	117	0.170	0.536	2.110

From experience we can say that nearly all the time is spent for determining successor situations, that is, doing projection, and reward calculation. Nevertheless, the presented amounts of time are still small enough for the Mid-Size League as this league is not yet as dynamic as the simulation.

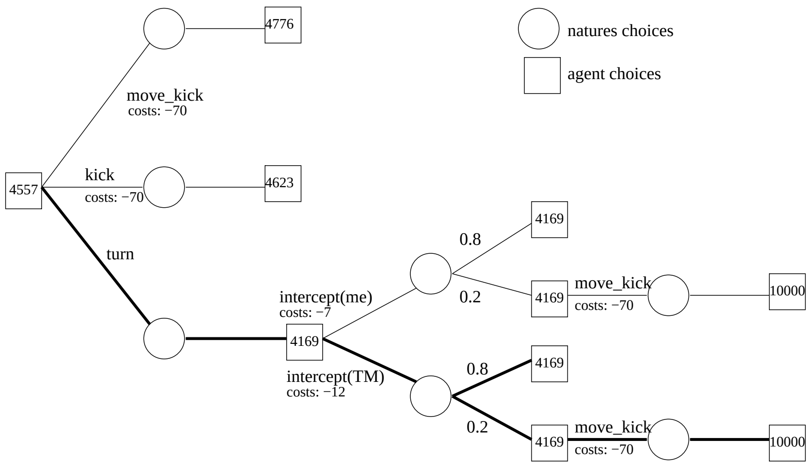


Fig. 9. The decision tree traversed by the optimization mechanism in this situation.

A key insight of the soccer experience with the interpreter is that the action models and the reward function are very crucial. The user is recommended to take great care designing them. Figure 8 (a), for instance, reveals a "mistake" in the used reward function: Here it appears much better to have the ball on the penalty area line in front of the opponents goal than at the touch line. Unluckily, this opinion had not been modeled into the reward function, such that the touch line position is evaluated higher for being farer into the opponents half. The agent, which is almost exclusively guided by the reward function, is very sensitive to such mistakes. They can easily lead to unintended behavior. All the more, sound approaches to creating reward functions, for example by learning, could be beneficial and investigated as future work.

References

1. Wunderlich, J.: Technical description of the AllemaniACs soccer robots. Technical report, LTI / KBSG, Aachen University, Germany (2002) in German.
2. Fox, D.: Kld-sampling: Adaptive particle filters. In: Advances in Neural Information Processing Systems 14, MIT Press (2001)
3. Milstein, A., Sànchez, J.N., Williamson, E.: Robust Global Localization Using Clustered Particle Filtering. In: Proc. of the National Conference on Artificial Intelligence (AAAI). (2002) 581–586
4. Gönner, C., Rous, M., Kraiss, K.F.: Robust color based picture segmentation for mobile robots. In: Autonome Mobile Systeme, Karlsruhe, Germany, Springer (2003) 64–74 (in German).
5. Burgard, W., Cremers, A., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., Thrun, S.: The interactive museum tour-guide robot. In: Proceedings of the AAAI 15th National Conference on Artificial Intelligence. (1998)

6. Reid, D.B.: An algorithm for tracking multiple targets. IEEE Transactions on Automatic Control **AC-24 No.6** (1979)
7. Cox, I.J., Hingorani, S.L.: An efficient Implementation of Reid´s Multiple Hypothesis Tracking Algorithm and its Evaluation for the Purpose of Visual Tracking. IEEE Transactions on PAMI **18/2** (1986) 138–150
8. Dylla, F., Ferrein, A., Lakemeyer, G.: Acting and Deliberating using Golog in Robotic Soccer – A Hybrid Approach. In: Proc. 3rd International Cognitive Robotics Workshop (CogRob 2002), AAAI Press (2002)
9. Riedel, B.: Die Entwicklung von ähnlichkeitsmaßen für den Vergleich von Spielsituationen im RoboCup. Master Theses, in progress, in german, Lehr- und Forschungsgebiet Informatik V, RWTH Aachen, Aachen, Germany (2003)
10. Quinlan, J.: C4.5 Programs for Machine Learning. Morgan Kauffmann (1993)
11. Konur, S.: Applying decision tree learning to the reactive component of robotic soccer agents. Master's thesis, Knowledge-based Systems Group, Computer Science V, RWTH Aachen, Aachen, Germany (2004)
12. Levesque, H.J., Reiter, R., Lesperance, Y., Lin, F., Scherl, R.B.: GOLOG: A logic programming language for dynamic domains. Journal of Logic Programming **31** (1997) 59–83
13. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press (2001)
14. Giacomo, G.D., Lésperance, Y., Levesque, H.J.: ConGolog, A concurrent programming language based on situation calculus. Artificial Intelligence **121** (2000) 109–169
15. Grosskreutz, H., Lakemeyer, G.: cc-Golog: Towards more realistic logic-based robot controllers. In: Proceedings of the 7th Conference on Artificial Intelligence (AAAI-00) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-00), Menlo Park, CA, AAAI Press (2000) 476–482
16. Grosskreutz, H.: Probabilistic projection and belief update in the pGolog framework. In: Second International Cognitive Robotics Workshop. (2000)
17. Dylla, F., Ferrein, A., Lakemeyer, G.: Specifying multirobot coordination in ICP-Golog – from simulation towards real robots. In: Proc. of the Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World modeling, planning, learning, and communicating (IJCAI 03). (2003)
18. Puterman, M.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley (1994)
19. Boutilier, C., Reiter, R., Soutchanski, M., Thrun, S.: Decision-theoretic, high-level agent programming in the situation calculus. In: AAAI'2000. (2000)