Team Description RoboLog

Heni Ben Amor[1], Joschka Boedecker[1], Rodrigo da Silva Guerra[1], and Yuichi Matsumoto[2]

[1]Dept. of Adaptive Machine Systems,
[2]Dept. of Information Science
Osaka University robolog@googlegroups.com

Abstract. In 2006 the team RoboLog will be witnessing the steady pace of RoboCup towards the 2050 goal for the eighth time. During these last eight years RoboLog have accompained the growth of the number of participating teams, as well as the growth in the complexity of code of both server and teams over the years. This paper focus on those two growth aspects, showing how this veteran team faces them in a healthy way. Traditionally composed exclusively by people from Koblenz university in Germany, this team benefited from an internationalization process welcoming members from different countries. From the technical side this paper recycles several development-aid tools that were formulated but never documented during the last three years. Finally, as the major scientific contribution, the authors propose a new *meta-learning architecture* for the modular employment and combination of different learning models, allowing the effortless assemble of complex mixed learning systems.

1 Introduction

The RoboLog Simulation team is a veteran participant of RoboCup games since 1999. Originally as the name suggests, agents were programmed by using Prolog. As the years passed by RoboLog grew along with the simulation league and the RoboCup itself. During these eight years of games several subjects were brought to the team, eventually leaving Prolog behind.

For 2006 deep changes are to be made in the formation of the team. Traditionally, RoboLog was formed by students and researchers from Koblenz University, Germany. This year the team becomes an example of the international integration that might result due to new rules which limit the number of teams per institution. Members of RoboLog currently affiliated to Osaka University in Japan merged their team with a local participants to form what is now a team mixed with germans, japanese and brazilian participants. We mention this fact in this paper as an example of the kind of integration and collaboration that we believe RoboCup is about to face more and more often in the future.

Despite the changes in the formation, this year RoboLog chose also to rescue several tools developed over the years but not properly mentioned in papers and symposiums – some of these tools were even forgotten, left unmantained. Several very useful and worth mentioning development tools are described for the first time in this paper (despite their long existence in some cases). The team currently work in recycling this

code and some results are already being shared as contributions for this year's development competitions. To complement with this a Prolog agent is being ported to the 3D Simulation team, thus rescuing the term which originally coined the name of the team.

Finally, as a major cientific contribution to the RoboCup community, we propose a new and percursor *meta-learning architecture*. This architecture serves as a tool for the co-engineering of different learning models, allowing hybrid systems to be easily built and tested in different configurations.

As we get closer to 2050, RoboCup tends to grow in complexity toward the accomplishment to the goal of having a team of fully autonomous humanoid robots having games against human players. As a consequence, it seems teams and even leagues are likely to merge during the years to come. Throughout this paper RoboLog addresses two of the most important issues related to this, which are: (a) the consistent development of development-aiding tools to fight increasing complexity, and (b) the example to integration and internationalization of members being faced in a positive way.

In section 2 three software tools for helping the development and tuning of the team are described. In section 3 the meta-learning architecture is introduced and described in detail.

2 Development-Aid Tools

During the years several original ideas were implemented and throughfully tested in actual games. The RoboLog system grew as a whole and as a consequence of that the development process became more complicated. With lots of code and tons of parameters to tune, development would be almost intractable if not for the development aiding solutions our team have implemented over the years. In the past these tools were not always mentioned in papers and a few not even documented at all. The team RoboLog 2006 committed itself into the recycling of such useful pieces of code. In the following subsections three of these development-aiding are described in more detail *one of which is to become available as a contribution for the simulation 3D development competition.

2.1 Behavior Editor based on UML Statecharts

Beginning in 2003, Jan Murray started the development of a tool for describing RoboLog agents in terms of a sequence of states and transitions based on the UML Statecharts format (more details to be found in [5]). UML Statecharts are hierarchical state transition diagrams, which means a state main contains several other states or even entire state-machines within. This hierarchical structure helps the designer to specify high-level plans by hand,

The behavior editor was implemented such as to account for three layers:

1. *Mode level*: More global patterns of behavior;
2. *Scripts level*: Repertoire of skeleton plans for each mode, including the explicit cooperation and coordination of the agents.
3. *Skills level*: Simple and complex actions which the agents use in their scripts

We plan to use this interface for tuning the behavior of our players during the RoboCup games of 2006.

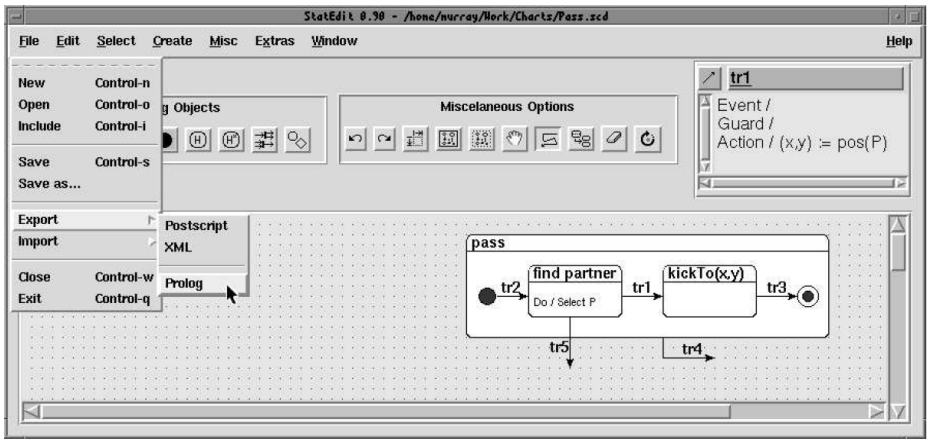

Fig. 1. StatEdit main window and export menu.

2.2 Offline Trainer

We developed a special monitor client that can be used to send commands to the 3D server which affect the properties of the ball or the agents on the field. It is possible to affect the position and velocity of these objects, but also internal player parameters like the battery state. The trainer can be used in addition to a regular monitor. It does not visualize the current simulation, but it receives all the information describing the scene at every simulation step. Two modes are available for the trainer, namely interactive and automated mode.

In interactive mode, commands can be sent to the server from the command line using Lisp-like S-Expressions. This is handy especially for debugging purposes or quick, controlled tests. In automated mode, the program is provided with a file of *scenarios*, i.e., initial configurations for player and/or ball positions and velocities. Furthermore, it uses *evaluation functions*, to determine the termination condition for a scenario. The trainer will reset the current scenario once the termination condition is met and record the results for further evaluation. Every scenario is repeated a specified number of times before the next scenario is used. This is done for all the scenarios in the given file. Using the automated mode, the trainer can be used to easily setup experiments and collect data for evaluation purposes.

2.3 SVG based formations

The formation of the players on the field is an aspect which has high influence into the performent of the team. Tweaking the standard position of the players might my quite cumbersome, even more if one has to rely on the explicit handcoding their coordinates. To address this problem RoboLog implemented for the first time in 2004 an interface for allowing the setup of the formation in a straightforward way.

RoboLog implemented a standard vector graphics (SVG) parser with which it became possible to input graphically the different formations. By employing this parser

any editor capable of dealing with SVG format can used for the actual drawing of the formation – graphical features are converted into their corresponding coordinates for the positioning of the players on the field.

3 Meta learning architecture

In recent years the RoboCup has witnessed various successful applications of learning techniques, such as reinforcement learning [9], genetic programming [4] or neural networks [8]. None of these techniques seems to be superior to the others, as each has advantages and drawbacks. Still, in most cases only one of the mentioned techniques is used to solve the problem at hand. Recent advances in hybrid metaheuristics research [3], however, suggest, that combinations of different learners can lead to even more powerful and robust techniques. As a result, there is an increasing body of research aiming at combining different learning approaches. These combination strategies can be roughly divided into two groups.

First, there are approaches in which a given problem is divided in two parts, each of which is solved by a different technique. Examples for this group include neuro-fuzzy [6] and neuro-evolutionary [1] approaches. Second, there are also approaches where the problem at hand is solved by a hierarchical combination of learners. For example, in the mixture-of-experts approach [7], small neural networks are learned at lower levels which are then combined by a neural network at a higher level. The intuition behind this approach is that each small network can specialize on some part of the problem e.g. become an expert in that part. The goal of the high level network is then to combine the expert knowledge, in order to achieve higher generalization. Other approaches which employ a similar idea include Bagging [2] and Boosting [10].

While the above mentioned combination strategies resulted in many state-of-the-art learning techniques, it is also known from optimization theory [11] that no optimal combination of techniques will be found. Each combination which leads to an increase in performance for a particular class of problems, will pay the price by worsening it's performance on other problem classes. Although no perfect combination of learning methods will be found, doing different combinations is still a good idea. Following this reasoning we propose a meta-learning architecture, which automatically creates a hierarchy of different learners to solve a given problem. Boosting, Bagging and existing hybrid technqiues can be regarded as instances of this architecture.

In Figure 2 we see the lowest level of the proposed architecture. In this level the building blocks of the system are specified. Building blocks are for instance, data filters and learning techniques. Data filters are used to preprocess the training data, so that learning can be made easier. Learning techniques are, of course, the main component of our architecture. The basic learning techniques of level 0 have to be provided by the user. While implementing a new learner the user has to conform to a standardized input and output interface. This transformes the all learning algorithms to black-blox techniques from the standpoint of the system. Furthermore, it facilitates the reuse and exchange of implemented methods between researchers. After implementing a basic learning technique, the user notifies the meta-learning architecture about the new addi-

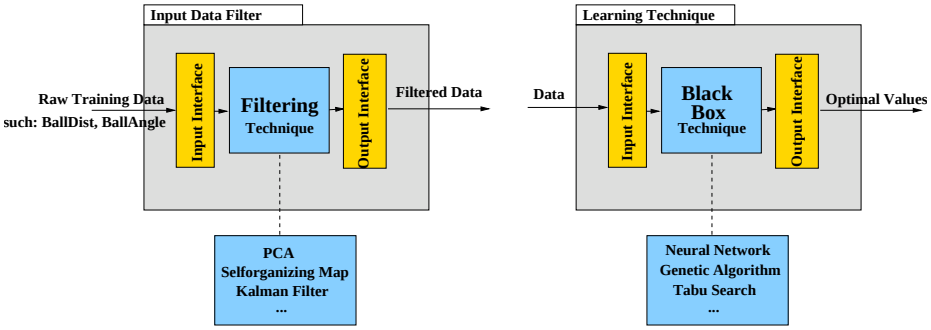

Fig. 2. Level 0 of the meta-learning architecture: a set of data filters and learning techniques are implemented by the user. By using a standardized input- and outputinterface we make sure the algorithms are interchangeable.

tion by changing the script file of the system. By looking up the script file, the system identifies the building blocks which can be combined and hybridized.

These building blocks of level 0 are combined at a higher level, in order to learn simple tasks such as going to a point or kicking the ball. Figure 3 shows an example of how this can be done. A smoothing filter and a learning technique are combined to produce a going to point behavior. In this example a neural network using backpropagation is used. For the task of finding the right combination of data filters and learners, the system uses another level 0 building block, such as a genetic algorithm.

Fig. 3. Level 1: A data filter and a learning technique from level 0 are combined in order to learn a go to point behavior. The input data is first processed by the filter, the passed to the learner which learns a visuo-motor mapping from the input to steering actions.

To solve more complex problems, level 0 and level 1 building blocks can be used. For instance, the previously learned *going to point* and *kicking the ball* skills from level 1 can be combined with a learning algorithm from level 0 in order to learn a dribbling behavior, see Figure 4. Finding the right combination of level 0 and level 1 components can, for instance, be done by tabu search. As a result, we have a hierarchy which enables us to learn increasingly complex skills through a reuse of components from lower levels.

Using this architecture we will hopefully be able to expand current application of learning techniques from learning low-level behaviors such as going to the ball or kicking to more high-level problems such as cooperation and coordination of agents.

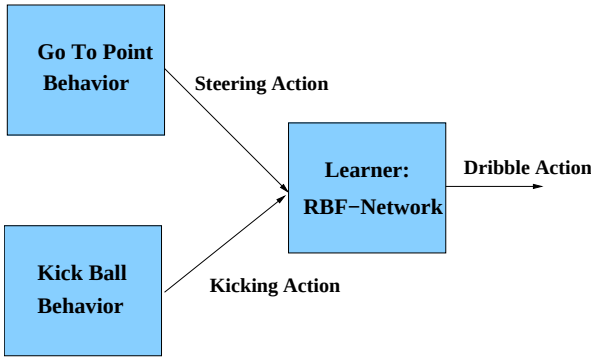

Fig. 4. Level 2: Two level 1 components (behaviors) and a level 0 component (a learner) are plugged together in order to learn a dribbling skill. An RBF-Network learns to arbitrate between the two low-level behaviors in order to achieve a complex dribbling.

4 Discussion

As mentioned in the beginning, RoboCup is growing ever more complex with the years. Accordingly, the agents that are built will also become much more complex – too complex to be handled by just a few members in a team. Cooperation between different teams will become increasingly important to handle all the aspects involved in building a competitive team. At the same time, an array of sophisticated development tools to aid the process of making a strong team are important, now and even more so in the future. Our team has already recognized these developments and sets an example of international collaboration to share knowledge and man-power to arrive at a more capable team. In this paper, we mainly described the development tools that we use to achieve this, and our concept of the meta-learning architecture that should facilitate the combination of different learning systems suitable for specialized tasks to a more powerful system that can be used to tackle the challenges the agents have to face on the field.

5 Acknowledgments

We would like to thank all former members of RoboLog who contributed so much to the development of this platform that we have today.

References

1. A. Agogino, K. Stanley, and Risto Miikkulainen. Online interactive neuro-evolution. *Neural Processing Letters*, 11(1):29–38, 2000.
2. Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139, 1999.

3. Maria J. Blesa, Christian Blum, Andrea Roli, and Michael Sampels, editors. *Hybrid Metaheuristics, Second International Workshop, HM 2005, Barcelona, Spain, August 29-30, 2005, Proceedings*, volume 3636 of *Lecture Notes in Computer Science*. Springer, 2005.
4. Sean Luke, Charles Hohn, Jonathan Farris, Gary Jackson, and James Hendler. Co-evolving soccer softbot team coordination with genetic programming. In Hiroaki Kitano, editor, *RoboCup-99: Robot Soccer World Cup I*, pages 398–411, Berlin, 1998. Springer Verlag.
5. Jan Murray. Specifying agents with uml statecharts and statedit. In *Proceedings of the RoboCup Symposium 2003*, July 2003.
6. Detlef Nauck. A fuzzy perceptron as a generic model for neuro–fuzzy approaches. In *Proc. Fuzzy–Systeme'94*, Munich, 1994.
7. S.J. Nowlan R.A. Jacobs, M.I. Jordan and G.E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79, 1991.
8. Achim Rettinger. Learning from Recorded Games: A Scoring Policy for Simulated Soccer Agents. Fachberichte Informatik 8–2004, Universität Koblenz-Landau, Universität Koblenz-Landau, Institut für Informatik, Universitätsstr. 1, D-56070 Koblenz, 2004.
9. Martin A. Riedmiller, Artur Merke, David Meier, Andreas Hoffmann, Alex Sinner, Ortwin Thate, and R. Ehrmann. Karlsruhe brainstormers - a reinforcement learning approach to robotic soccer. In *RoboCup*, pages 367–372, 2000.
10. Robert E. Schapire. A brief introduction to boosting. In *IJCAI*, pages 1401–1406, 1999.
11. David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.