GermanTeam 2002

Uwe Düffert[1], Matthias Jüngel[1], Tim Laue[2],
Martin Lötzsch[1], Max Risler[3], Thomas Röfer[2]

[1] Institut für Informatik, LFG Künstliche Intelligenz,
Humboldt-Universität zu Berlin, Rudower Chaussee 25, 12489 Berlin, Germany
[2] Bremer Institut für Sichere Systeme, TZI, FB 3 Mathematik/Informatik,
Universität Bremen, Postfach 330440, 28334 Bremen, Germany
[3] Fachgebiet Simulation und Systemoptimierung, FB 20 Informatik,
Technische Universität Darmstadt, Alexanderstr. 10, 64283 Darmstadt, Germany

1 Introduction

The GermanTeam participates as a national team in the Sony Legged Robot League. It currently consists of students and researchers from five universities: the Humboldt-Universität zu Berlin, the Universität Bremen, the Technische Universität Darmstadt, the Universität Dortmund, and the Freie Universität Berlin. The members of the GermanTeam are allowed to participate as separate teams in national contests such as the RoboCup German Open, but will jointly line up at the international RoboCup championship as a single team. Obviously, the results of the team would not be very good if the members will develop separately until the German Open in the middle of April, and then try to integrate their code to a single team in only two months. Therefore, an architecture for robot control programs was developed that allows to implement different solutions for the tasks involved in playing robot soccer. The solutions are exchangeable, compatible to each other, and they can even be distributed over a varying number of concurrent processes. The approach is described in this year's RoboCup book [6]. Other contributions of the GermanTeam to the book are [4] and [5]. The team description here will focus on topics not published elsewhere so far, namely vision and behavior.

2 Vision

In the Sony Legged Robot League, a color camera integrated into the 3-DOF head of the robots is the most important sensor. All relevant objects on the soccer field are color-coded, e.g. the flags (for self-localization), the goals, the ball, the opponents, and the teammates. Therefore, image processing is the most basic and time-consuming skill for robots competing in this league. The GermanTeam implemented two different approaches to recognize the different objects on the field:

RLE-Floodfilling. The first approach is similar to [2]. The image provided by the sensor is converted into a color class run length encoded image, the runs of which are afterwards grouped to octangular blobs (cf. Fig. 1a). The vertices of the blobs are rep-

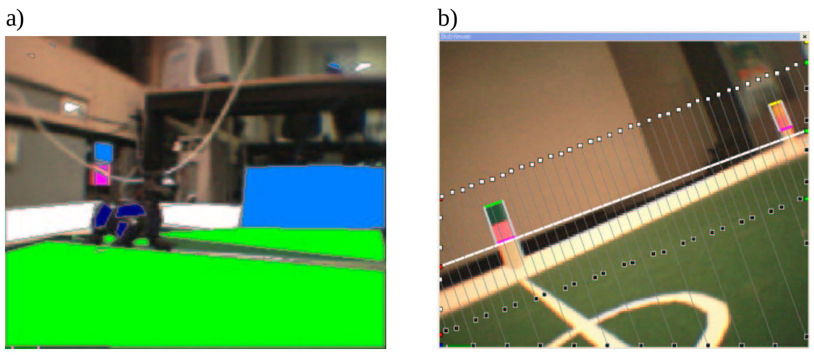


Fig. 1. Image processing. a) Blob-based. b) Grid-based.

resented in angular coordinates, integrating the orientation of the camera. Then, these blobs are used to detect the relevant objects using the techniques developed last year [1, 3].

Grid Image Processing. To increase the speed of the vision system, each image is scanned at low resolution to find objects of interest using a roughly perspective-oriented, horizon-aligned grid. Scan-points are dense around the horizon to find far and thus small features, and coarse when close to the camera (cf. Fig. 1b). Whenever object candidates are found (by identifying typical patterns of neighboring colors), they are examined locally at high resolution using so-called "specialists". The result are horizon-aligned bounding boxes for landmarks, players and goals, a circle for the ball and arrays of points for the field lines. The new image processing performs no global filtering or segmentation and is not only more accurate, but also fast enough to process the full frame rate of the robot, i.e. 25 images per second, while leaving enough computing power to other tasks.

3 Behavior

The behavior architecture is based on the last year's approach of the GermanTeam.

Hierarchies of Options. The general idea is that there is a hierarchy of options that are organized in an option tree with skills as the leaves (cf. Fig. 2a.). Each option can execute a specific behavior independently from its context in the option tree, using the options and skills below it. The execution of the option tree always starts from the root option. Then, each option activated determines which suboption will be executed until an option refers to a skill. In contrast to the last year's approach, the options are not organized in fixed layers but can be combined freely.

Each option contains an internal state machine. Each state of that state machine stands for a skill or a suboption (cf. Fig. 2b). Each state has a decision tree that selects between transitions to states (cf. Fig. 3a, b). If an option is executed, the state machine is carried out to determine which state is activated next (that can also be the same).

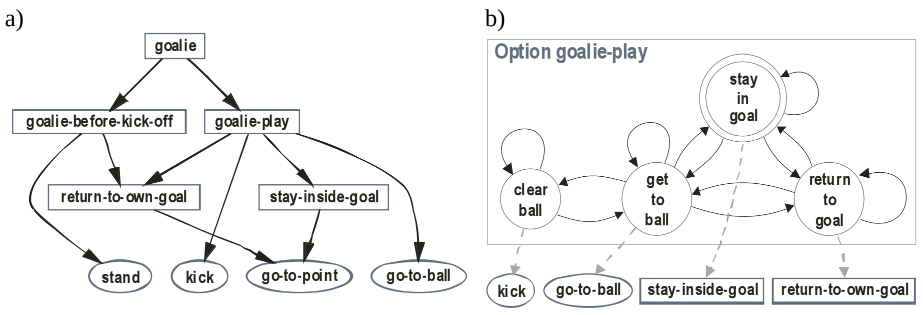


Fig. 2. a) The option tree of a simplified goalie. Boxes stand for options, ellipses for skills. b) The internal state machine of the option "goalie-play". The double circle is the initial state.

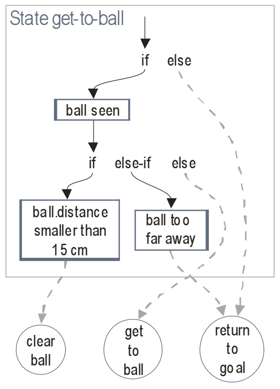
Then, for the active state, the referred suboption or skill is executed and so on. Due to the separation of long term and context dependent tasks into different options on different levels of the option tree, very complex behaviors can be implemented with that architecture.

Formalization of Behavior – XABSL. In GermanTeam 2001, it turned out that implementing such architecture in C++ is not very comfortable and error prone. The size of the source files of the complete implementation exceeded 500 KB and it was complicate to add new options. Therefore, the *extensible agent behavior specification language* (XABSL) was developed, an XML dialect specified with XML Schema. The reasons for the use of XML technologies were the big variety and the quality of existing tools, the possibility of easy transforming from and to other languages as well as the general flexibility of data represented in XML languages.

Behaviors using the architecture described in the previous section can be completely described in XABSL. There is a class library called XabslEngine that can execute behaviors written in XABSL using an intermediate code that is generated from that language. To be independent from specific software environments and platforms, the formalization of the interaction with the software environment is also included in XABSL by defining symbols. Interaction means access to input functions and variables (as, e.g., of the world state), to output functions (e.g. to set the LED of the robot) and to the skills, which are written in C++. At the start of the engine, a platform and application dependent XabslSymbolProvider gives the engine access to variables and functions for each symbol. For example, one can introduce the arbitrary numeric symbol "ball.distance" in the XML document. The engine then asks the symbol provider to return the address of a function or a variable for the symbol "ball.distance" without any semantic knowledge about it, and it will later work with that address.

The execution of the engine is not as fast as a behavior description written in C++, but it is fast enough, because the intermediate code is only read once at the startup of the robot. For debugging the behavior, the XABSL Behavior Tester was added to the RobotControl [6] application, which allows tracing the symbols of the formalized behavior as well as the options and states activated. In addition, single options or skills can be selected to be tested separately. A detailed documentation can be generated from the XML document (Fig. 2 and 3 were taken from that documentation).

a)



b)

```
if (ball.time-since-last-seen < 2000)                //ball seen
{
  if  (ball.distance < 150)  //ball.distance smaller than 15 cm
  {
    transition-to-state (clear-ball);
  }
  else if  (ball.distance > 900)            //ball too far away
  {
    transition-to-state (return-to-goal);
  }
  else
  {
    transition-to-own-state (get-to-ball);
  }
}
else
{
 transition-to-state (return-to-goal);
}
```

Fig. 3. a) The decision tree for the state "get-to-ball". The leaves of the tree are transitions to other states. b) Pseudo code of the decision tree for the state "get-to-ball".

4 Conclusion

The GermanTeam has developed architectures on two different levels. On the one hand, a national team has the special need for a multi-team architecture, supporting multiple solutions for the tasks required to play robot soccer. On the other hand, XABSL was established as a powerful tool for fast and comfortable behavior developing. In addition, methods for fast image processing were implemented, one of which only analyses the relevant areas of the camera image.

Future work will concentrate on the extension of the behavior architecture and the language by deliberation and communication as well as on a "more natural" self-localization that is based on field lines, goals, and teammates.

References

1. Burkhard, H.-D., Düffert, U., Jüngel, M., Lötzsch, M., Koschmieder, N., Laue, T., Röfer, T., Spiess, K., Sztybryc, A., Brunn, R., Risler, M., v. Stryk, O.: GermanTeam 2001. Technical report. Only available online: http://www.tzi.de/kogrob/papers/GermanTeam2001report.pdf (2001).
2. Bruce, J., Balch, T., Veloso, M.: Fast and inexpensive color image segmentation for interactive robots, In: Proceedings of IROS-2000. Japan (2000).
3. Brunn, R., Düffert, U., Jüngel, M., Laue, T., Lötzsch, M., Petters, S., Risler, M., Röfer, T., Spiess, K., Sztybryc, A.: GermanTeam 2001. In *RoboCup 2001*. Lecture Notes in Artificial Intelligence. Springer (2002), to appear.
4. Dahm, I., Ziegler, J.: Adaptive Methods to Improve Self-Localization in Robot Soccer. In: RoboCup 2002. Lecture Notes in Artificial Intelligence. Springer (2003), to appear.
5. Hardt, M., von Stryk, O.: The Role of Dynamics in the Design, Control and Stability of Bipedal and Quadrupedal Robots. In: RoboCup 2002. Lecture Notes in Artificial Intelligence. Springer (2003), to appear.
6. Röfer T. (2003). An Architecture for a National RoboCup Team. In: RoboCup 2002. Lecture Notes in Artificial Intelligence. Springer (2003), to appear.