RoboLog Koblenz 2003 – Team Description

Heni Ben Amor, Jan Murray, Oliver Obst, and Christoph Ringelstein *

Universität Koblenz-Landau, AI Research Group
Universitätsstr. 1, D–56070 Koblenz, GERMANY
{amor,murray,fruit,cringel}@uni-koblenz.de

Abstract. As in the previous years, the RoboCup Soccer Simulation League team *RoboLog* consists of a C++ low level and a Prolog higher level part. For implementation of reactive skills of a soccer agent, we are making use of and extending the idea of steering behaviors. In our team, we use these extended steering behaviors, called *inverse steering behaviors*, for dribbling and obstacle avoidance.

1 Introduction

With RoboCup 2003, RoboLog Koblenz is participating in the simulated soccer world championship for the fifth time. As in the years before, high level decisions in our team are specified with statecharts and implemented in Prolog. For lower levels, we are using a C++ layer between Prolog and the Soccer Server network interface. The C++ part handles the parsing of messages, maintains a consistent world model, and also implements the lower level behaviors of our players.

We have started to reimplement the RoboLog-Kernel from scratch before RoboCup 2002, where we already have been using it for our goalie. To support goal-directed decision making in the Prolog level, our kernel provides basic actions that can make short-term decisions for obstacle avoidance and skill optimization autonomously. Currently we are investigating on making use of steering behaviors [5] for making short-term decisions while pursuing several goals at once (like keeping control of the ball, avoiding opponents and reaching a specific place at the same time).

For the higher-level part of our team we are currently investigating ways to specify [2] and implement soccer agents. While in the last years goals of an agent were specified implicitly in our architecture, we are moving to a explicit representation of goals this year. We hope that the benefit of this approach is an easier maintenance of our Prolog code base because separate concerns can also be represented separately. The generation of goals is driven by combination of different kinds of spatial information, which is another research issue in our project [6].

In this paper, we focus on the steering part of our team. In the subsequent sections we give a brief overview of the original concept of steering behaviors and present our extension of this concept, called *inverse steering behaviors*, which we use for dribbling and obstacle avoidance.

* This research is supported by the grants *Fu 263/6-1* and *Fu 263/8-1* from the German research foundation *DFG*.

2 Inverse Steering Behaviors

Steering is the reactive, non-deliberative movement of physical agents [3]. Reynolds [5] defines a hierarchical model where steering is responsible for path determination for autonomous characters in animations and games. In this hierarchy, the layer below steering, called locomotion, is responsible for the actual implementation of steering goals. In Simulation League, the locomotion layer is realized by using the Soccer Server low level commands.

The layer on top of steering, called action selection, represents the deliberative part of an autonomous agent and deals with selecting the appropriate strategies and plans. In our framework, the top layer is represented by a Prolog implementation.

The steering layer usually consists of a collection of *steering behaviors*. When applied, steering behaviors produce a steering vector which serves as input for the locomotion layer. An advantage of steering behaviors is that several ones can be combined to form a more complex behavior. A problem, however, is how this combination actually can be achieved without resulting in undesired behaviors. Reynolds [5] proposes different ways of "blending" steering behaviors, e.g. using their weighted average or choosing only one behavior at a time according to given priorities. A hybrid arbitration scheme using and extending these ideas can be found in [4], where it is called *prioritized acceleration allocation*.

When experimenting with these arbitration methods for implementing a dribbling skill, we ran into several problems:

1. Trap situations due to local minima (cyclic behavior).
2. No passage between closely spaced obstacles.
3. Oscillations in the presence of obstacles.

To overcome these drawbacks, we extended the concept of steering behaviors [1]. In the original approach, steering behaviors only take facts about the environment as input and produce a single steering vector as output. Our approach additionally takes a number of different steering vectors as input denoting possible solutions for the steering task. Based on a given criterion, each direction produces a certain cost. In turn, from the calculated cost we produce a rating for each steering vector as output. In principle we inverted the process of each single steering behavior, which is why we call our approach *inverse steering behaviors*. To execute a single behavior, the steering vector producing the smallest cost is selected.

To achieve complex tasks, several inverse steering behaviors can be combined similar to the original approach. In our case the problem is not to combine different steering vectors or actions, but to merge the ratings for each given direction. All relevant behaviors for a task have to be executed, producing a rating for each given steering vector. Merging the ratings is achieved with a heuristics combining ratings of all involved behaviors for each direction separately. The result of applying the heuristics for each given steering vector is a list of ratings like the one produced by a single inverse steering behavior. Like in the case of single steering behaviors the "best" action can be chosen by simply selecting the steering direction minimizing the cost.

The heuristics employed in the process of combining several behaviors is dependent on the number of behaviors involved and specific to the complex task that has to be achieved.

With an appropriate heuristics, merging the ratings for each single behavior can produce better results than the original approach simply adding up steering vectors or selecting one. In cases where two behaviors have conflicting desires, inverse steering behaviors will select a steering vector obeying each of the behaviors to a degree, and thus make a reasonable compromise for all involved behaviors.

3 Navigation Using Inverse Steering Behaviors

When using inverse steering behaviors the task of navigation in a cluttered environment is modeled as a cost minimization problem. For both, the separate inverse steering behaviors and the overall navigation task we create heuristic cost functions, based on which the quality of a given solution can be measured. For a better understanding of the latter statements, we will subsequently discuss a navigation example, where collision free paths are achieved using the obstacle avoidance and seek (goal approaching) behavior.

In Fig. 1 we see a particular situation, where a robot has to decide in which direction to steer. The decision of the robot has to satisfy both his will of avoiding collisions and taking the fastest way to the goal. Given the discrete set $\{\theta_1, \theta_2, \theta_3, \theta_4\}$ of steering directions to be evaluated, we first apply the inverse steering behaviors *obstacle avoidance* and *seek* separately. These inverse steering behaviors assign each of the given directions a cost with respect to a criterion related to the inherent task.

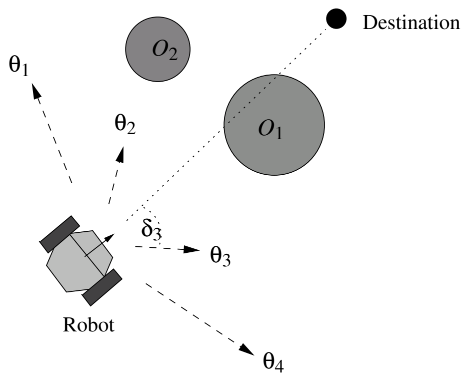

Fig. 1. Evaluation of directions to avoid obstacles.

For instance the *seek* behavior assigns costs according to the difference between the direction α denoting the straight path to the goal position and the currently processed direction θ_i, which can be expressed using the formula $\delta_i = ||\alpha - \theta_i||$. Consequently direction θ_2 and θ_3 receive fewer costs, as their deviation to α is smaller. In contrast, the *obstacle avoidance* behavior assigns costs based on the number x_{θ_i} of obstacles in the currently processed direction. As a result θ_2 receives the cost value 1 due to an occurring obstacle, whereas θ_1, θ_3 and θ_4 are obstacle free and thus receive the cost value 0. The number x_{θ_i} is derived by assigning each direction a rectangular area. These rectangular areas represent a possible path to be taken by the agent. The width and the length of each rectangular area are based on the agent's width and velocity. The faster he is, the sooner he must anticipate a potential collision. This enables us to extract information about the existence of obstacles very easily using a geometrical construction of the problem. The bounding sphere of each object in the near range of the agent is intersected with the rectangular region of the currently processed direction in order to determine if it lies inside this region. All objects that fulfill the last condition are considered to be possible collision partners. Figure 2 shows a trajectory of a dribbling agent using inverse steering behaviors to determine its path.

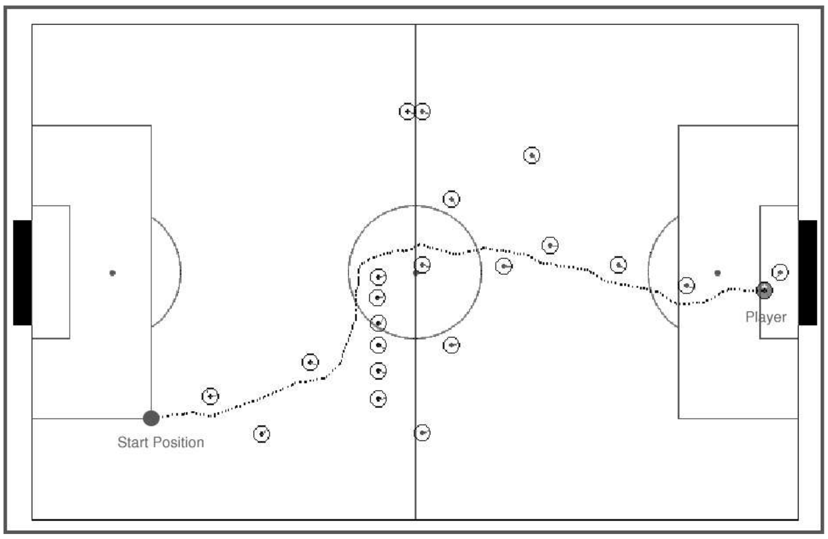

Fig. 2. Dribbling around static obstacles.

4 Conclusions and Future Work

In our paper we presented how to implement skills of a soccer agent using inverse steering behaviors. The introduction of inverse steering behaviors in [1] was motivated by drawbacks of the original steering behaviors as proposed by Reynolds. Our inverse steering behaviors are still reactive, fast to calculate, and easy to combine with each other to build more complex behaviors. Because our agents evaluate several options before steering, the actions to be performed are better motivated. Additionally, it is easier to solve the problem of arbitration inherent to steering behaviors using costs as a measure of quality.

For future work we are planning to investigate on using machine learning techniques to simplify building heuristics and to combine our hand crafted behaviors with machine learned ones.

References

1. Heni Ben Amor, Oliver Obst, and Jan Murray. Fast, neat and under control: Inverse steering behaviors for physical autonomous agents. Fachberichte Informatik 12–2003, Universität Koblenz-Landau, Universität Koblenz-Landau, Institut für Informatik, Rheinau 1, D-56075 Koblenz, 2003.
2. Jan Murray. Specifying agent behaviors with UML statecharts and StatEdit. In *Proceedings of the RoboCup Symposium 2003*, Padova, Italy, 2003. (to appear).
3. Alexander Nareyek, Nick Porcino, and Mark Kolenski. AI interface standards: The road ahead. A Roundtable Discussion of the 2003 Game Developers Conference, March 2003. `http://www.ai-center.com/events/gdc-2003-roundtable/`.
4. Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.
5. Craig W. Reynolds. Steering behaviors for autonomous characters. In *Proceedings of the Game Developers Conference (GDC-1999)*, pages 763–782, San Francisco, California, 1999.
6. Frieder Stolzenburg, Oliver Obst, and Jan Murray. Qualitative velocity and ball interception. In Matthias Jarke, Jana Köhler, and Gerhard Lakemeyer, editors, *KI-2002: Advances in Artificial Intelligence – Proceedings of the 25th Annual German Conference on Artificial Intelligence*, number 2479 in Lecture Notes in Artificial Intelligence, pages 283–298, Berlin, Heidelberg, New York, 2002. Springer.