Fast, Neat and Under Control: Inverse Steering Behaviors for Physical Autonomous Agents

Heni Ben Amor, Oliver Obst, Jan Murray

12/2003

Fast, Neat and Under Control: Inverse Steering Behaviors for Physical Autonomous Agents

Heni Ben Amor, Oliver Obst, Jan Murray[*]
Universität Koblenz-Landau
Arbeitsgruppe Künstliche Intelligenz
{amor,fruit,murray}@uni-koblenz.de

April 18, 2003

Steering behaviors are a set of motion based reactive procedures used for navigating autonomous agents in their environment. Combinations of steering behaviors can be used to create complex behaviors. One problem inherent to existing approaches to arbitrating between single behaviors is that their combination may lead to suboptimal, undesired, or even catastrophic results in certain situations. In our paper we present a solution to these problems by introducing inverse steering behaviors for controlling physical agents. Inverse steering behaviors change the original concept of steering behaviors and facilitate improved arbitration between different options by using cost based heuristics.

We also show a concrete application of inverse steering behaviors, namely the implementation of a dribbling skill with sophisticated obstacle avoidance for a RoboCup soccer agent.

1 Introduction

Highly dynamic environments are a big challenge for mobile robotic systems. This is especially true when robots not only have to avoid obstacles but also have to continue pursuing a different task at the same time, or when adversaries are actively trying to stop them from reaching a certain place. On the one hand, robots have to be very reactive with respect to changes in the environment. On the other hand, being reactive alone is not enough to fulfill certain tasks

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

efficiently. To implement robotic agents that behave both reactive and goal directed these two issues are usually separated into a reactive and a deliberative layer. Steering behaviors [Reynolds, 1987] are a way to implement the reactive part in physical automous agents. Complex steering behaviors are created by combining several simple ones. In our paper we address drawbacks of the original steering behaviors we became aware of when implementing agents for RoboCup Soccer Simulation League [Noda *et al.*, 1998].

The rest of the paper is organized as follows. In Sections 2 and 3 a brief summary of steering behaviors and their limitations is presented. In Sec. 4 inverse steering behaviors are introduced. Section 5 shows, how they can be used for navigation tasks. A concrete application, the implementation of a dribbling skill for a RoboCup agent, is presented in Sec. 6. In Sec. 7 a short overview over related work is given, before the paper finally is concluded with a summary and an outlook to future work in Sec. 8.

2 About Steering Behaviors

Steering is the reactive, non-deliberative movement of physical agents [Nareyek *et al.*, 2003]. Reynolds [Reynolds, 1999] 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. The layer on top of steering, called action selection, represents the deliberative part of an autonomous character and deals with selecting the appropriate strategies and plans.

A basic ingredient to steering are *steering behaviors*, reactive procedures that take local information about the environment as input, producing a steering vector (or steering goal) as output. A set of basic steering behaviors presented in [Reynolds, 1999] includes *seek*, *flee*, *pursuit*, *evasion*, *containment*, and *obstacle avoidance* for static obstacles.

To produce complex behaviors,e.g. flocking or queuing at a doorway, several steering behaviors can be combined with each other at a time. Each of the basic steering behaviors is executed separately or in parallel, and the different steering goals have to be combined to one result that is passed to the locomotion layer. Reynolds [Reynolds, 1999] 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 [Reynolds, 1987] where it is called *prioritized acceleration allocation*. Here all applicable behaviors are used in order of their priority, until the maximum amount of acceleration is reached.

Example applications using steering behaviors are simulation of pedestrians [Feurtey, 2000] and vehicles [Bonakdarian *et al.*, 1998] in urban environments, emergency evacuation of human crowds [Helbing *et al.*, 2000], and also autonomous characters in computer games and movies.

3 Limitations of Steering Behaviors

With steering behaviors complex tasks can be achieved by using a combination of several simple but specialized components. These specialized behaviors are developed manually to carry out specific subtasks such as following a corridor, avoiding obstacles, or intercepting a moving object. The combination process of these components poses several difficulties, as each behavior makes its decision independently of the others. This can result in conflicting commands, and depending on the arbitration method single behaviors can possibly cancel the effect of each other out or result in suboptimal paths. The arbitration method of building a weighted average of steering vectors shares its greatest disadvantages with potential field methods when used for robot navigation [Koren and Borenstein, 1991]:

- Trap situations due to local minima (cyclic behavior).
- No passage between closely spaced obstacles.
- Oscillations in narrow passages or in the presence of obstacles.

Also priority based blending of behaviors as proposed by Reynolds [Reynolds, 1999] causes problems in many cases, so that constraints of some of the involved behaviors are violated.

In addition these behaviors lack any kind of *context awareness*, because of their limited complexity and the focus on local information. The result of applying steering behaviors is a steering vector which might violate other constraints of tasks to be performed. Usually, these constraints are established again in subsequent steps by other steering behaviors. In some cases however, it might be too late to fix the situation, for example an avoidance maneuver steering around one obstacle may lead the performing agent into another close obstacle, as the spatial context in which the behavior was triggered has not been taken into account previously. When the obstacle avoidance is triggered the next time, the collsion might already have occurred.

One possible solution to this problem would be *high-level reasoning* about the effects of each action generated in the reactive layer. In the layered model of Reynolds deliberation is done in the action selection layer, so this solution amounts to lifting the tasks of the steering layer to the action selection layer. In most cases reasoning about every single steering action is computationally intractable. Additionally, in a very dynamic environment, planning each step from start to a goal state is a waste of resources, because it is most likely that some or all parts of a previously computed path will be invalidated in near future. Reactive behaviors are necessary for efficiency, but a deliberative part is also important: due to their local nature reactive behaviors are incapable of achieving global goals.

Because of this, a solution to our problem should ensure the decision autonomy of the reactive layer. That means no precise steering instructions but only subgoals are communicated from the action selection layer to the steering layer. We present a method called *inverse steering behaviors* that changes the way steering behaviors are used and makes use of heuristics in the reactive steering layer to keep the tradeoff between local and global decisions minimal.

4 Inverse Steering Behaviors

To overcome the drawbacks mentioned in the last section, we extended the concept of steering behaviors. 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 separatly. 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. We are going to illustrate building a sample heuristics in a subsequent section.

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.

5 Navigation Using Inverse Steering Behaviors

When using inverse steering behaviors the task of navigation in a cluttered environment is modelled 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 Figure 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

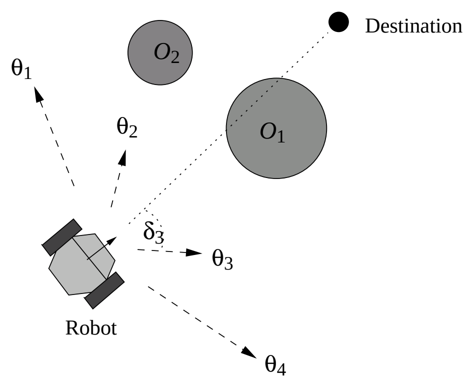

Figure 1: Evaluation of directions to avoid obstacles.

criterion related to the inherent task.

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 the subsequent sections we will refer to α as the "optimistic optimal" direction. In contrast, the obstacle avoidance behavior assigns costs based on the number x_{θ_i} of obstacles in the currently processed direction (cf. Fig. 2). 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. However, the number x_{θ_i} is hard to be derived analytically and would lead to complex expressions defined over space. We tackle this problem by assigning each direction a rectangular area using a function rectRegion(θ_i). These rectangular areas represent a possible path to be taken by the agent. The width and the length of each rectRegion are based on the agent's width and velocity. The faster he is, the sooner must he 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.

Once the cost assignment procedure of each separate behavior is finished, we have to find a way how to combine the different results in order to derive the overall navigation costs. This is accomplished using another heuristic cost function. A possible heuristics for the robot in Figure 1 using a weighted sum of the previously computed costs would for instance be:

$$h(C^{seek}, C^{oa}) = \frac{1}{80} C^{seek} + C^{oa} \tag{1}$$

where C^{seek} and C^{oa} denote the costs derived from the seek and obstacle avoidance behaviors.

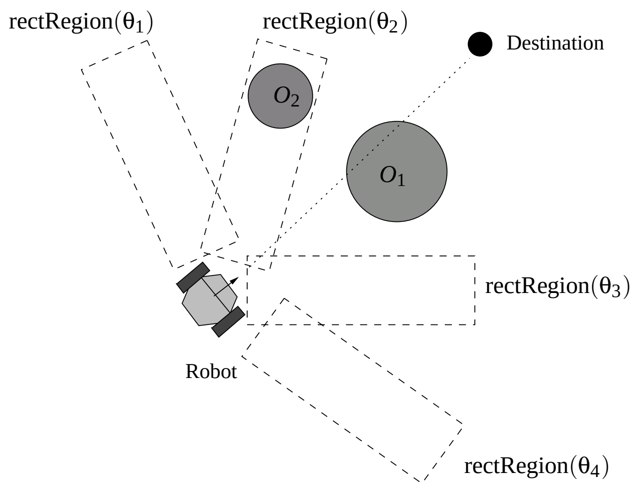

Figure 2: Assigning of rectangular regions for every direction.

The result of function *h* is the total navigation cost of the currently processed direction. Table 1 shows the costs determined for the robot in Fig. 1. As lower costs indicate better solutions, the optimal steering direction is the one which minimizes *h* in the current time step. In our example this direction is θ_3.

	θ_1	θ_2	θ_3	θ_4
C^{seek}	80	40	40	80
C^{oa}	0	1	0	0
$h(C^{seek}, C^{oa})$	1	1.5	0.5	1

Table 1: Navigation costs for the robot in Fig. 1

The general algorithm for selecting a steering direction is given as Algorithm 1. It takes three input parameters. The borders of the sector to be searched for steering directions are denoted by ϕ_{min} and ϕ_{max}. The third parameter, *n* gives the number of discrete directions between ϕ_{min} and ϕ_{max} to be evaluated with inverse steering behaviors. The direction with minimal costs is returned by the algorithm.

The main difficulty of this approach is to find a “good” heuristics producing an appropriate mapping from directions to costs. The weight of each behavior involved in the navigation process is set based on its priority and range of results. For a small number of behaviors these weights can be derived using prior human knowledge or through empirical testing. But if the number of subtasks in navigation grows higher, other techniques for extracting the weights, such as reinforcement learning or genetic algorithms have to be used.

In general it can be said, that in our approach favorable navigation appears as side effect of minimizing the costs related to each involved behavior. The robot example given in this section can for instance be compared to a car driver's tendency to minimize his expenses, which is done by both, economizing his fuel and avoiding any accidents.

Algorithm 1 `steeringDirection`($\phi_{\min}, \phi_{\max}, n$) – Calculation of the steering direction with minimal costs between $\phi_{\min}$ and $\phi_{\max}$ by evaluating n discrete directions.

```
1:  {Part 1: Discretization of steering directions}
2:  $\Delta \leftarrow (\phi_{\max} - \phi_{\min})/n$
3:  for $i = 0$ to $n$ do
4:      $\theta_i \leftarrow \phi_{\min} + \Delta \times i$
5:  end for
6:
7:  {Part 2: Apply Inverse Steering Behaviors}
8:  Let $S_1, \ldots, S_m$ be the relevant Inverse Steering Behaviors
9:  for $k = 1$ to $m$ do
10:     $(c_0^k, \ldots, c_n^k) \leftarrow S_k(\theta_0, \ldots, \theta_n)$
11: end for
12:
13: {Part 3: Apply heuristics}
14: min_costs $\leftarrow \infty$    {Set minimal costs to $\infty$}
15: $l \leftarrow 0$    {Set index of selected direction to 0}
16: for $j = 0$ to $n$ do
17:     cur $\leftarrow h(c_j^1, \ldots, c_j^m)$    {Calculate costs for current direction}
18:     if min_costs $>$ cur then
19:         min_costs $\leftarrow$ cur    {Update minimal costs}
20:         $l \leftarrow j$    {Update (index of) selected direction}
21:     end if
22: end for
23: return $\theta_l$    {$\theta_l$ is the direction producing minimal costs}
```

5.1 Moving Obstacles

Decisions based only on momentary information can heavily degrade the agent's ability to avoid obstacles. This is especially true, when the agent's environment is dynamically changing or when opponents try to keep the agent from achieving his goal. The efficiency of navigation in such environments strongly relies on the agent's ability to predict future situations and include them in the current decision process. This demands a higher amount of abstraction, as it involves both predicting possible collisions with nearby moving objects and altering its own direction based on their current speed and orientation. In [Reynolds, 1999] a behavior called *unaligned collision* *avoidance* has been proposed, which tries to apply a corrective steering by predicting the closest approach between the agent and opponents or moving objects. This method, however, assumes that both, agent and moving obstacle are eager to avoid a collision.

In order to keep our approach consistent for both, mobile and stationary obstacles we investigated geometric representations of an object's motion. Rather than adding supplementary handling for moving objects, we perform the same method as in Section 5, but with revised shapes based on the obstacles velocity and heading. These shapes represent the *influence region* of an obstacle. We assume that the mobile obstacle can reach any position within this influence region faster than the agent, so all of them must be avoided. This implies, that the agent's task is to find a way which has no intersections with any influence region. Therefore instead of checking for intersections of the bounding sphere of an object with the rectangular representation of the currently pursued direction, we check for intersections between the influence region of an object with our direction. For the trivial case that the object is stationary, the influence region equals the obstacle's bounding sphere, which results in the same process as described above. For the non-trivial case, that the object is moving, we distinguish between two types of geometric representations chosen with respect to the obstacle's means of locomotion.

For domains with non-holonomic mobile objects, influence regions are represented by a semi-circle in the back of the obstacle and a semi-ellipse in its front. The diameters of both shapes are proportional to the velocity of the observed obstacle, as it elongates with speed (a similar approach has been taken in [Reynolds, 2000] for defining neighborhood relationships). This representation is due to the fact, that non-holonomic objects or robots have less degrees of freedom than the total number of degrees available. This means, that such an object would have first to turn and then to accelerate in two separate time steps, if it would have to change its heading. The more the direction to a goal position deviates from the current heading, the harder are they to reach. Because of this limitation, positions which lie in the opposite direction of the object take much more time to reach. For holonomic objects however every position within a particular radius r takes the roughly same amount of time to reach, as such objects have a higher amount of controlable degrees. Therefore the influence region of the latter are represented by circles with radius r, which can be computed using the object's speed scaled by some constant factor.

The resulting influence regions are then used to replace the original obstacle during the collision detection step. This means that, rather than determining if there is a collision (intersection) between the bounding spheres of all obstacles and currently processed way, we determine the collisions between the influence regions and the latter way. This is shown in Algorithm 2.

6 Application in Robotic Soccer

In this section we describe the implementation of a dribbling skill for the RoboCup Simulation League with the help of inverse steering behaviors. After a brief introduction to the Simulation League we present our approach to dribbling with integrated obstacle avoidance. The section is finally closed with some experimental results.

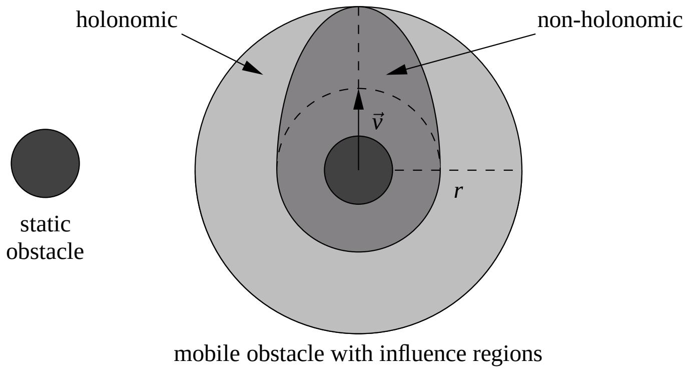

Figure 3: Influence region of a mobile obstacle compared to a static obstacle with $r = 2 \times |\vec{v}|$.

Algorithm 2 `ObstacleAvoidance(`$(\gamma_0, \ldots, \gamma_n)$`)`
Example Inverse Steering Behavior: Obstacle avoidance.

```
1:  {calculate set of influence regions of relevant objects}
2:  R ← getInfluenceRegions($\gamma_0, \gamma_n$)
3:  for $i = 0$ to $n$ do
4:      $\text{count}_i \leftarrow 0$
5:      for all $R_j \in R$ do
6:          if $R_j \cap$ rectRegion($\gamma_i$)$\neq \emptyset$ then
7:              $\text{count}_i \leftarrow \text{count}_i + 1$
8:          end if
9:      end for
10: end for
11: return $(\text{count}_0, \ldots, \text{count}_n)$
```

6.1 RoboCup Simulation League

The Simulation League [Noda *et al.*, 1998] is part of the RoboCup Initiative [RoboCup Federation,], which aims at fostering research in robotics, multiagent-systems and AI. The domain of RoboCup is robotic soccer where teams of robots have to work together in order to win soccer matches against other robot teams. Researchers have the opportunity to present their results and evaluate their approaches under real world conditions during the annual international RoboCup World Championships as well as a variety of local events.

In the Simulation League two teams of 11 autonomous agents compete in a simulated soccer match. The two dimensional, discrete-time simulation is carried out in a client/server style by the *RoboCup Soccer Simulator* (or *Soccer Server* for short) [Chen *et al.*, 2003]. The Soccer Server maintains a model of the world containing the positions of all objects on the field, as well as additional information about them, e.g. the velocities of moving objects or the remaining stamina of all players. In each simulation step clients may send *one* command for moving or manipulating the ball, e.g. *dash*, *kick* or *turn*, and several minor commands to the server. The effects of these commands are taken into account by the simulator when the world model is updated for the next step. Sensory input is sent to the agents at regular intervals, but asynchronously to the simulation steps.

The commands provided by the server allow for very primitive interaction with the environment only, so in order to successfully achieve higher level behaviors and abstract goals more complex abilities have to be synthesized from them. Thus in the lower levels of a soccer playing agent a set of *skills* like passing, shooting or intercepting the ball is implemented. The overall performance of an agent and even the whole team is heavily influenced by these skills.

6.2 Dribbling

One important skill, called *dribbling*, is defined as the ability to move towards a target point, while keeping possession of the ball. This confronts an agent with a hard problem. In soccer the player controlling the ball is generally attacked by one or more opponents. Hence he has to find a strategy how to outplay them, while still approaching his target point. The latter includes several subtasks such as kicking and intercepting the ball, and avoiding adversaries and sidelines. Obviously, this results in a complex navigation task, as it involves the use of different behaviors in a real-time, dynamic, and rapidly changing environment with limited resources (time and stamina). Further difficulties are added to the problem, as a dribbling agent is in general slower as his opponents. This is due to the fact that many of his action opportunities are spent kicking rather than dashing, as every time step only one such action is possible. Also, in order to turn into a given direction without losing control of the ball, a sequence of 2 to 3 primitive actions is needed.

Building upon the above analysis of the dribbling skill, we identified the following tasks to be taken out. Both the tasks and their equivalent steering behaviors are given in Table 6.2.

As can be seen in this table, the dribbling process can be expressed as a combination of

Task	Steering Behavior
Approach target position	Seek
Intercept the ball	Pursuit
Avoid enemies	Obstacle avoidance
Staying on the play field	Containment
Controlling the ball	-

Table 2: Behaviors involved in the dribbling skill.

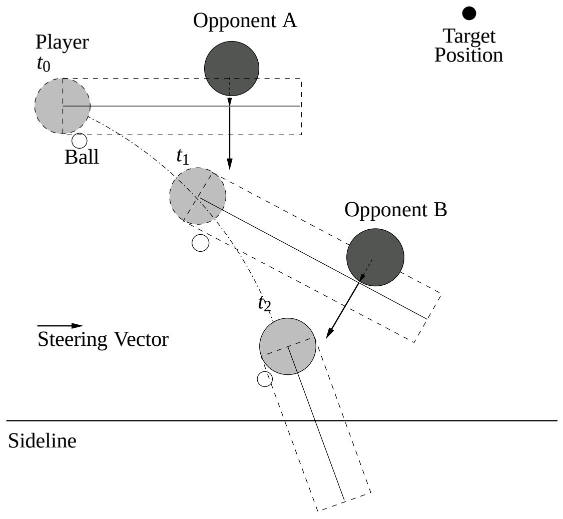

Figure 4: Player is trapped near the sideline.

different steering behaviors. However, a straightforward implementation of the idea revealed several shortcomings of this approach. The observed properties of the resulting dribbling behavior confirmed most of the theoretical considerations made in Section 3. Due to the lack of context awareness, many cases appeared where the resulting direction after performing obstacle avoidance would directly point at a second opponent. As a consequence the agent would make oscillating turns between different opponents without dashing forward. A more severe problem occurred if a player tried successively to outplay two opponents close to the sideline. In such situations the repeated avoidance maneuvers would often lead him into facing the sideline, where he would then be trapped and thus an easy prey for the approaching opponents (cf. Fig. 4). The aforementioned problems are inherent to various other navigation and obstacle avoidance techniques and were already identified as limitations of *potential field methods* (PFM) in [Borenstein and Koren, 1989]. Especially the drawback of being trapped near the sideline bares great similarity to *local-minima* problems in PFM.

After tests with selection and blending techniques [Reynolds, 1999] and parameter adjustments, we incorporated our *inverse steering behavior* approach into the dribbling skill. First we distinguished between sequentially switching and parallelly firing of behaviors. While the pursuit and ball control behavior are switched sequentially (if the ball must be intercepted or is endangered to get out of control), the seek, avoidance and containment behaviors are used in parallel every time step determining for collision free and valid paths. This means that we have to find a heuristic cost function, which arbitrates between the latter three behaviors. The resulting function would then be used to evaluate and rate possible steering directions in each time step. Thus a solution can be found which satisfies all criteria of interest (seek, avoidance, containment) up to some extent.

6.3 Behaviors, Criteria and Heuristics

Crucial points in the inverse steering behavior model explained in this paper are, the cost assignment process of currently active behaviors and the choice of the arbitrating heuristic. In the first process, each active behavior receives a set of possible steering directions, to which it assigns costs according to some predefined criteria. The arbitrating heuristic then takes the resulting costs, in order to determine the ideal steering direction for the momentary situation. As a first step for applying this approach, we outlined for each behavior the criterion, based on which it assigns costs.

Obstacle Avoidance In order to avoid being hit by any occurring obstacles the agent should prefer directions with fewer influence regions of obstacles. An example algorithm for the cost assignment process of this behavior is given in Algorithm 2.

Seek In order to successfully approach his goal, the agent has to minimize the deviation to the direction of the target point. Therefore the difference between his current orientation and the “optimistic optimal” one is taken as a criterion for this behavior.

Containment The containment behavior tries to keep the agent from leaving the play field. Therefore we use his predicted position (linearly extrapolated) in the currently processed direction after 5 time steps $\vec{pos}_{t+5}$. If $\vec{pos}_{t+5}$ is inside the play field, then no costs will be assigned. However, if the predicted position lies outside the play field, then the distance between $\vec{pos}_{t+5}$ and the nearest sideline is returned as a cost.

Using the above criteria, we derive for each direction its costs C_θ^{oa}, C_θ^{seek} and C_θ^{cont}.

As next step we have to design the heuristic function transforming the separate costs into an overall cost value. Basically we have to assign a weight to each behavior, denoting its contribution to the total costs of a direction. For our dribbling skill, we derived these weights empirically after various tests, resulting in the following heuristics:

$$h(C_\theta) = C_\theta^{oa} + \frac{1}{4}C_\theta^{cont} + \frac{1}{180}C_\theta^{seek} \quad . \tag{2}$$

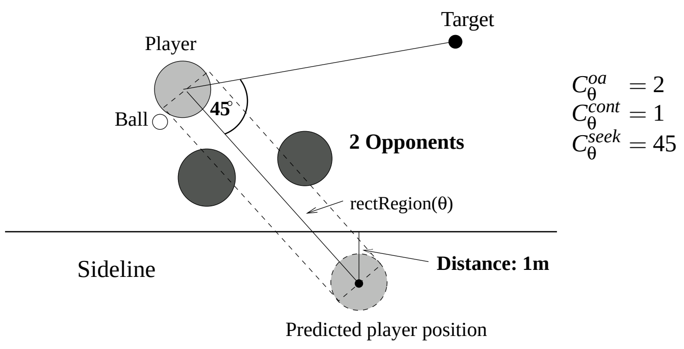

Figure 5: Calculating costs for direction θ

As can be seen in (2) the primary contribution to the costs of a direction θ is done by the obstacle avoidance behavior. This is due to the fact that one of the most important principles in dribbling is the avoidance of any possible collision with nearby opponents. The least influence on the cost function $h(C_\theta)$ has the seek behavior, which penalizes the deviation between the ”optimistic optimal” direction (see Section 5) and the agents current heading. In all situations where the agent is not heading directly into an obstacle or into a sideline however, the direction costs will be based only on the outcome of C_θ^{seek}. In such situations, both variables C_θ^{oa} and C_θ^{cont} take the value 0, indicating that no avoidance or containment maneuvers have to be done. Hence the agent will choose his steering direction solely according to the seek behavior and thus follows his high level goal.

6.4 Experimental Results

With our dribbling player we conducted several experiments with static and dynamic obstacles. For this end we used both, artificially created scenarios and real soccer games, mostly in the Simulated Soccer Internet League [Simulated Soccer Internet League,], a spin-off of the RoboCup Simulation League.

In the artificial scenarios a player repeatedly had to dribble the ball from a given start position to a postion on the field which allowed for a clear scoring opportunity. All obstacles on the field had to be avoided. Figure 6 shows the trajectory of the agent as he has dribbled across almost the whole field to reach a save scoring position.

Our tests showed, that our approach to dribbling based on inverse steering behaviors allows for flexible behaviors in the agent and fast reactions to changes in the environment while staying focused on the given high level goal.

Many of the situations that cannot be handled by agents using classical steering behaviors for navigation at all are managed by our dribbler without even losing the ball, e.g. the situation shown in Fig. 4.

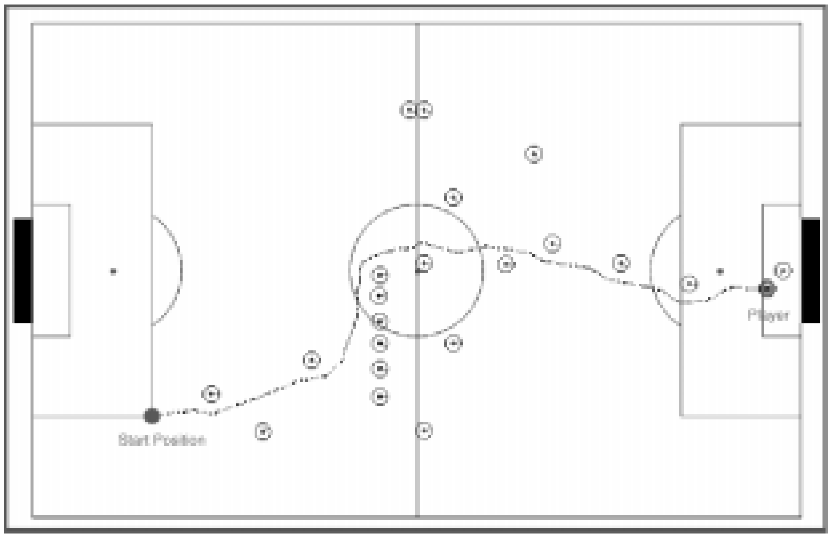

Figure 6: Dribbling around static obstacles.

Up to now we were only able to conduct qualitative tests, but for the near future exhaustive testing of the approach with statistical methods and comparisons to other approaches are planned.

7 Related Work

Steering behaviors found their way to robotic soccer in the early days of RoboCup. In 1998 both the simulated and the real (small-size) robot team from Carnegie Mellon University used an approach called Strategic Positioning using Attraction and Repulsion (SPAR) [Veloso *et al.*, 1999]. SPAR can be seen as a variant of the flocking behavior [Reynolds, 1987]; in contrast to earlier positioning approaches in robotic soccer SPAR takes into account not only the position of the ball but also positions of other players on the field. Two elements responsible for flocking behavior, *repulsion* and *attraction*[1], were realized by a method that can be seen as an extension of potential field methods. Forces repulse and attract players from and to the other objects on the field, i.e. the teammates, opponents, ball and goals. Another ingredient for flocking described in [Reynolds, 1987], *alignment*, was not used in SPAR.

Situation Based Strategic Positioning (SBSP) [Reis *et al.*, 2001] is a second approach in this line from the FC Portugal simulator team. With respect to the positioning, SBSP is more simplistic than the approach used in SPAR, taking mainly the attraction of the ball into account. In terms of steering behaviors the team performs *leader following*, a kind of flocking behavior,

[1]In [Reynolds, 1987] they are called *cohesion* and *separation*.

using the ball as leader of the flock. To add further flexibility, SBSP is wrapped by a mechanism called DPRE (Dynamic Positioning and Role Exchange), which extends earlier work by [Stone and Veloso, 1998].

In the CMUnited-98 small-size robots team, a simple steering behavior built into the motion control leads to obstacle-free paths in a dynamic environment [Bowling and Veloso, 1999]. If an obstacle occurs too close in the target direction of the robot, the steering vector is adjusted so that it runs tangent to a preset circle around the object in question.

8 Conclusions and Future Work

In our paper we presented a new approach to steering. With inverse steering behaviors decisions are made by evaluating costs of a discrete set of possible solutions. Preserving advantages of the original approach, 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.

We successfully implemented our approach in the simulated soccer world for an agent dribbling the ball while avoiding different obstacles.

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

[Bonakdarian *et al.*, 1998] Esmail Bonakdarian, James Cremer, Joseph Kearney, and Pete Willemsen. Generation of ambient traffic for real-time driving simulation. In *Image Society Conference*, pages 123–133, Scottsdale, AZ, USA, August 1998.

[Borenstein and Koren, 1989] Johann Borenstein and Yoram Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1179–1187, 1989.

[Bowling and Veloso, 1999] Michael Bowling and Manuela Veloso. Motion control in dynamic multi-robot environments. In *Proceedings of the1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA '99)*, volume November, pages 168–173, 1999.

[Chen *et al.*, 2003] Mao Chen, Klaus Dorer, Ehsan Foroughi, Fredrik Heintz, ZhanXiang Huang, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Jan Murray, Itsuki Noda, Oliver Obst, Pat Riley, Timo Steffens, Yi Wang, and Xiang Yin. *RoboCup Soccer Server*, 2003. Manual for Soccer Server Version 7.07 and later (obtainable from `sserver.sf.net`).

[Feurtey, 2000] Franck Feurtey. Simulating the collision avoidance behavior of pedestrians. Master's thesis, University of Tokyo, School of Engineering, February 2000.

[Helbing *et al.*, 2000] Dirk Helbing, Illés Farkas, and Tamás Vicsek. Simulating dynamical features of escape panic. *Nature*, 407:487–490, 2000.

[Koren and Borenstein, 1991] Yoram Koren and Johann Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings of the IEEE Int. Conf. on Robotics and Automation*, pages 1398–1404, 1991.

[Nareyek *et al.*, 2003] 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/`.

[Noda *et al.*, 1998] Itsuki Noda, Hitoschi Matsubara, Kazuo Hiraki, and Ian Frank. Soccer Server: a tool for research on multi-agent systems. *Applied Artificial Intelligence*, 1998.

[Reis *et al.*, 2001] Luis P. Reis, Nuno Lau, and Eugénio C. Oliveira. Situation based strategic positioning for coordinating a team of homogeneous agents. In Markus Hannebauer, Jan Wendler, and Enrico Pagello, editors, *Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, number 2103 in LNCS, pages 175–197. Springer, 2001.

[Reynolds, 1987] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

[Reynolds, 1999] Craig W. Reynolds. Steering behaviors for autonomous characters, 1999.

[Reynolds, 2000] Craig W. Reynolds. Interaction with groups of autonomous characters, 2000.

[RoboCup Federation,] Official homepage of the RoboCup Federation. `http://www.robocup.org/`.

[Simulated Soccer Internet League,] Simulated Soccer Internet League. `http://sserver.sf.net/league/index.html`.

[Stone and Veloso, 1998] Peter Stone and Manuela Veloso. Task decomposition and dynamic role assignment for real-time strategic teamwork. In *Agent Theories, Architectures, and Languages*, pages 293–308, 1998.

[Veloso *et al.*, 1999] Manuela Veloso, Peter Stone, and Michael Bowling. Anticipation: A key for collaboration in a team of agents. In *Third International Conference on Autonomous Agents (Agents'99)*, 1999.

Available Research Reports (since 1998):

2003

12/2003 *Heni Ben Amor, Oliver Obst, Jan Murray.* Fast, Neat and Under Control: Inverse Steering Behaviors for Physical Autonomous Agents.

11/2003 *Gerd Beuster, Thomas Kleemann, Bernd Thomas.* MIA - A Multi-Agent Location Based Information Systems for Mobile Users in 3G Networks.

10/2003 *Gerd Beuster, Ulrich Furbach, Margret Groß-Hardt, Bernd Thomas.* Automatic Classification for the Identification of Relationships in a Metadata Repository.

9/2003 *Nicholas Kushmerick, Bernd Thomas.* Adaptive information extraction: Core technologies for information agents.

8/2003 *Bernd Thomas.* Bottom-Up Learning of Logic Programs for Information Extraction from Hypertext Documents.

7/2003 *Ulrich Furbach.* AI - A Multiple Book Review.

6/2003 *Peter Baumgartner, Ulrich Furbach, Margret Groß-Hardt.* Living Books.

5/2003 *Oliver Obst.* Using Model-Based Diagnosis to Build Hypotheses about Spatial Environments.

4/2003 *Daniel Lohmann, Jürgen Ebert.* A Generalization of the Hyperspace Approach Using Meta-Models.

3/2003 *Marco Kögler, Oliver Obst.* Simulation League: The Next Generation.

2/2003 *Peter Baumgartner, Margret Groß-Hardt, Alex Sinner.* Living Book – Deduction, Slicing and Interaction.

1/2003 *Peter Baumgartner, Cesare Tinelli.* The Model Evolution Calculus.

2002

12/2002 *Kurt Lautenbach.* Logical Reasoning and Petri Nets.

11/2002 *Margret Groß-Hardt.* Processing of Concept Based Queries for XML Data.

10/2002 *Hanno Binder, Jérôme Diebold, Tobias Feldmann, Andreas Kern, David Polock, Dennis Reif, Stephan Schmidt, Frank Schmitt, Dieter Zöbel.* Fahrassistenzsystem zur Unterstützung beim Rückwärtsfahren mit einachsigen Gespannen.

9/2002 *Jürgen Ebert, Bernt Kullbach, Franz Lehner.* 4. Workshop Software Reengineering (Bad Honnef, 29./30. April 2002).

8/2002 *Richard C. Holt, Andreas Winter, Jingwei Wu.* Towards a Common Query Language for Reverse Engineering.

7/2002 *Jürgen Ebert, Bernt Kullbach, Volker Riediger, Andreas Winter.* GUPRO – Generic Understanding of Programs, An Overview.

6/2002 *Margret Groß-Hardt.* Concept based querying of semistructured data.

5/2002 *Anna Simon, Marianne Valerius.* User Requirements – Lessons Learned from a Computer Science Course.

4/2002 *Frieder Stolzenburg, Oliver Obst, Jan Murray.* Qualitative Velocity and Ball Interception.

3/2002 *Peter Baumgartner.* A First-Order Logic Davis-Putnam-Logemann-Loveland Procedure.

2/2002 *Peter Baumgartner, Ulrich Furbach.* Automated Deduction Techniques for the Management of Personalized Documents.

1/2002 *Jürgen Ebert, Bernt Kullbach, Franz Lehner.* 3. Workshop Software Reengineering (Bad Honnef, 10./11. Mai 2001).

2001

13/2001 *Annette Pook.* Schlussbericht “FUN - Funkunterrichtsnetzwerk”.

12/2001 *Toshiaki Arai, Frieder Stolzenburg.* Multiagent Systems Specification by UML Statecharts Aiming at Intelligent Manufacturing.

11/2001 *Kurt Lautenbach.* Reproducibility of the Empty Marking.

10/2001 *Jan Murray.* Specifying Agents with UML in Robotic Soccer.

9/2001 *Andreas Winter.* Exchanging Graphs with GXL.

8/2001 *Marianne Valerius, Anna Simon.* Slicing Book Technology — eine neue Technik für eine neue Lehre?.

7/2001 *Bernt Kullbach, Volker Riediger.* Folding: An Approach to Enable Program Understanding of Preprocessed Languages.

6/2001 *Frieder Stolzenburg.* From the Specification of Multiagent Systems by Statecharts to their Formal Analysis by Model Checking.

5/2001 *Oliver Obst.* Specifying Rational Agents with Statecharts and Utility Functions.

4/2001 *Torsten Gipp, Jürgen Ebert.* Conceptual Modelling and Web Site Generation using Graph Technology.

3/2001 *Carlos I. Chesñevar, Jürgen Dix, Frieder Stolzenburg, Guillermo R. Simari.* Relating Defeasible and Normal Logic Programming through Transformation Properties.

2/2001 *Carola Lange, Harry M. Sneed, Andreas Winter.* Applying GUPRO to GEOS – A Case Study.

1/2001 *Pascal von Hutten, Stephan Philippi.* Modelling a concurrent ray-tracing algorithm using object-oriented Petri-Nets.

2000

8/2000 *Jürgen Ebert, Bernt Kullbach, Franz Lehner (Hrsg.).* 2. Workshop Software Reengineering (Bad Honnef, 11./12. Mai 2000).

7/2000 *Stephan Philippi.* AWPN 2000 - 7. Workshop Algorithmen und Werkzeuge für Petrinetze, Koblenz, 02.-03. Oktober 2000 .

6/2000 *Jan Murray, Oliver Obst, Frieder Stolzenburg.* Towards a Logical Approach for Soccer Agents Engineering.

5/2000 *Peter Baumgartner, Hantao Zhang (Eds.).* FTP 2000 – Third International Workshop on First-Order Theorem Proving, St Andrews, Scotland, July 2000.

4/2000 *Frieder Stolzenburg, Alejandro J. García, Carlos I. Chesñevar, Guillermo R. Simari.* Introducing Generalized Specificity in Logic Programming.

3/2000 *Ingar Uhe, Manfred Rosendahl.* Specification of Symbols and Implementation of Their Constraints in JKogge.

2/2000 *Peter Baumgartner, Fabio Massacci.* The Taming of the (X)OR.

1/2000 *Richard C. Holt, Andreas Winter, Andy Schürr.* GXL: Towards a Standard Exchange Format.

1999

10/99 *Jürgen Ebert, Luuk Groenewegen, Roger Süttenbach.* A Formalization of SOCCA.

9/99 *Hassan Diab, Ulrich Furbach, Hassan Tabbara.* On the Use of Fuzzy Techniques in Cache Memory Managament.

8/99 *Jens Woch, Friedbert Widmann.* Implementation of a Schema-TAG-Parser.

7/99 *Jürgen Ebert, and Bernt Kullbach, Franz Lehner (Hrsg.).* Workshop Software-Reengineering (Bad Honnef, 27./28. Mai 1999).

6/99 *Peter Baumgartner, Michael Kühn.* Abductive Coreference by Model Construction.

5/99 *Jürgen Ebert, Bernt Kullbach, Andreas Winter.* GraX – An Interchange Format for Reengineering Tools.

4/99 *Frieder Stolzenburg, Oliver Obst, Jan Murray, Björn Bremer.* Spatial Agents Implemented in a Logical Expressible Language.

3/99 *Kurt Lautenbach, Carlo Simon.* Erweiterte Zeitstempelnetze zur Modellierung hybrider Systeme.

2/99 *Frieder Stolzenburg.* Loop-Detection in Hyper-Tableaux by Powerful Model Generation.

1/99 *Peter Baumgartner, J.D. Horton, Bruce Spencer.* Merge Path Improvements for Minimal Model Hyper Tableaux.

1998

24/98 *Jürgen Ebert, Roger Süttenbach, Ingar Uhe.* Meta-CASE Worldwide.

23/98 *Peter Baumgartner, Norbert Eisinger, Ulrich Furbach.* A Confluent Connection Calculus.

22/98 *Bernt Kullbach, Andreas Winter.* Querying as an Enabling Technology in Software Reengineering.

21/98 *Jürgen Dix, V.S. Subrahmanian, George Pick.* Meta-Agent Programs.

20/98 *Jürgen Dix, Ulrich Furbach, Ilkka Niemelä .* Nonmonotonic Reasoning: Towards Efficient Calculi and Implementations.

19/98 *Jürgen Dix, Steffen Hölldobler.* Inference Mechanisms in Knowledge-Based Systems: Theory and Applications (Proceedings of WS at KI ’98).

18/98 *Jose Arrazola, Jürgen Dix, Mauricio Osorio, Claudia Zepeda.* Well-behaved semantics for Logic Programming.

17/98 *Stefan Brass, Jürgen Dix, Teodor C. Przymusinski.* Super Logic Programs.

16/98 *Jürgen Dix.* The Logic Programming Paradigm.