Experiences Using the Dynamical System Paradigm for Programming RoboCup Robots

Hans-Ulrich Kobialka, Herbert Jaeger

Fraunhofer Institute for Autonomous Intelligent Systems, Sankt Augustin, Germany

E-mail: {kobialka,jaeger}@ais.fraunhofer.de

Abstract. *Autonomous mobile robots in dynamic environments have to switch from the „stop and wait until I'm sure" mentality to a continuous movement kind of style. As both the robot and objects in its surrounding move, the robot has to react on sensory input as fast as possible but without becoming instable. This is challenging also because speed increases the uncertainty of sensory input. Therefore techniques are needed to protect the behavior system from the effects of noisy input, to avoid self-induced instabilities, and to issue motor commands which are continuous over time. The mathematical framework of dynamical systems offers techniques to care for system stability and therefore is the natural choice. The Dual Dynamics architecture allows to model robot behavior as dynamical systems. We use the DD approach successfully since years to program our soccer robots. These experiences are reported in this paper.*

1 Introduction

In mobile robotics there is an inherent trade-off between speed and accuracy/completeness of information.

- Sensory equipment has to be minimized in order to reduce the mass of the robot (and its costs too). For instance, in the RoboCup mid-size league there were long debates about the use of laser range finders which on one hand can provide excellent information about the position of all robots, walls, and goals. On the other hand, such a laser range finder costs >2000 Euro and weights approx. 4.5 kg. Many teams decided to have faster robots with less accurate sensors.
- Sensors (e.g. cameras, distance sensors, odometry) become more noisy and inaccurate if the robot moves fast. Also the likelihood of reporting artifacts (i.e. objects which don't exist) increases.
- Speed requires fast decisions. The robot can't wait until it is sure about the state of the environment. Instead, it should react even if the sensory input is uncertain.
  High reactivity bears the risk that the robot may become unstable (i.e. showing oscillations or uncontrollable moves).

Robot soccer is a very dynamic and competitive game. It requires fast movement and sudden changes in direction. During a game, many different situations have to be mastered by the robot. Every year at the RoboCup [1] world championship the best teams of all over the world meet and compete in robot technology and behavior programming.

For implementing behaviors on a robot, behavior architectures are used. A behavior architecture provides definitions of modules which implement behaviors on a technical level, and performs 'action selection', i.e. coordination of behavior modules in order to determine the right motor signals according to the current situation.

Action selection for fast robots has to master the trade-off between reactiveness and stability.

An ongoing action should be pursued even in the presence of insufficient or conflicting sensory input. If the sensory input isn't appropriate for too long, another action has to be selected. During switching between actions, there is the risk of instability. There should be techniques which allow the designer to specify the stability of an action and how fast the switching between actions should be performed.

We participate in the RoboCup mid-size league since 1998. During this time, we experienced a lot of problems regarding behavior programming and tried out several solutions. The behavior architecture we use is the Dual Dynamics architecture. Its main advantage is that it supports to design robot behavior as a dynamical system.

Loosely speaking, a *dynamical system* describes how the state of some system evolves over time [2]. In robotics, state spaces are usually continuous and high-dimensional. Mapping such a state space to symbols bears the risk of information loss and discontinuities. Modelling state change as a dynamical system may lead to much richer and smoother trajectories in state space. Another aspect arises from the fact that the control program interacts with the physics of the robot and the environment. These physics are regarded (and modelled) as non-linear dynamical systems. In a dynamical approach, the control program, the robot's „body“ and the environment are viewed as coupled dynamical systems. A wide-spread way to specify dynamical systems are differential and difference equations.

In this paper, we will first give an overview on the Dual Dynamics architecture in section 2. Details on action selection and motor control are given in sections 3 and 4. Other approaches are discussed in section 5.

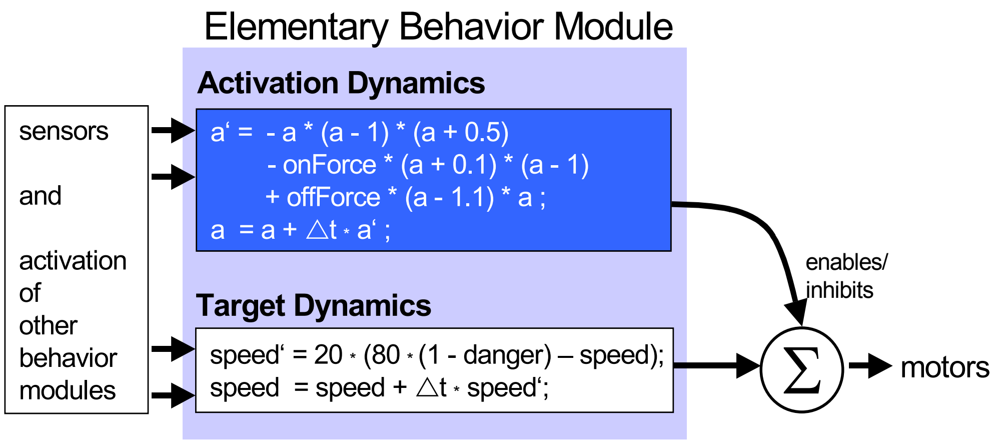
2 Overview on Dual Dynamics

The main idea of the Dual Dynamics (DD) architecture [11] is that behavior systems for autonomous robots should be designed as self-organizing dynamical systems. All transitions between control modes are regulated by dynamical systems (in contrast to finite state machines). This way, the decision point is dynamically and continuously tuned in a robust way.

The main building blocks of a DD architecture are behavior modules, called *behaviors* for short[1]. Typical examples of behaviors are *Kick* ("push the ball into the goal") or *BehindBall* ("drive behind the ball"). Behaviors are ordered in levels (figure 2). At the bottom level, there are elementary behaviors. Elementary behaviors are made from two parts which serve quite different purposes: target dynamics and activation dynamics. This has given the approach its name, "Dual Dynamics". The target dynamics define how the behavior interacts with the environment (i.e. how to react on sensor input) while the activation dynamics compute the activation of the behavior. The activation of a behavior ranges between 0 (not active) and 1 (fully active).

The final motor values are computed by taking the output of the target dynamics of all elementary behaviors and computing the weighted sum by using the activation values as weights (see 4.3). In this way, the activation of a behavior modulates the effects of its target dynamics to the actuators (i.e. the motors).

1.A behavior (module) has to be distinguished from the behavior of a robot which can be viewed by a human observer. An observed behavior may be influenced by several behavior modules.



Elementary Behavior Module
Activation Dynamics
a‘ = - a * (a - 1) * (a + 0.5)
- onForce * (a + 0.1) * (a - 1)
+ offForce * (a - 1.1) * a ;
a = a + △t ⋆ a‘ ;
Target Dynamics
speed‘ = 20 ⋆ (80 ⋆ (1 - danger) – speed);
speed = speed + △t ⋆ speed‘;
sensors
and
activation
of
other
behavior
modules
enables/
inhibits
Σ
motors

Figure 1. The interface and internal structure of an elementary behavior.

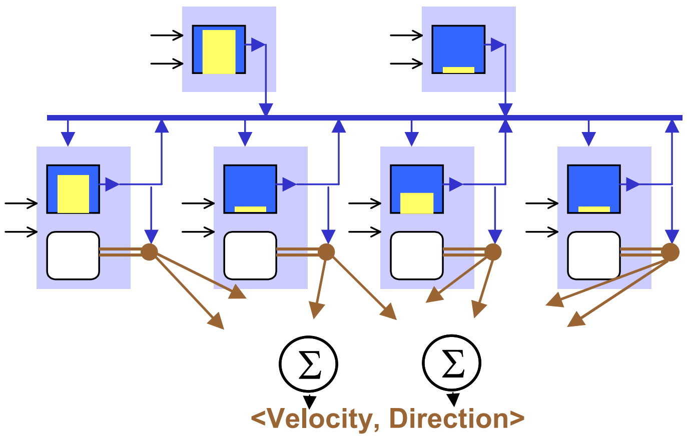


Figure 2. The behavior hierarchy in Dual Dynamics.

Higher level behaviors have no target dynamics and therefore can’t influence the motors directly. Their activation values are used as control parameters by other behaviors. Higher level behaviors typically implement ’moods’ of the robot, like ball seeking or homing.

3 Motor Control

The target dynamics of an elementary behavior can be regarded as a ’special-purpose controller’, i.e. a controller which regulates the robot’s motion in order to achieve the purpose of this particular behavior. We typically use difference equations (discretisation to the differential equations typically considered in control theory) to specify target dynamics [7]. As there is no single controller to cope with every situation occurring during a complex task, switching between controllers has to modelled (’action selection’, see section 4). Together this leads to a set of coupled differential equations which implement a dynamical system.

4 Action Selection

In the presence of several alternatives, the robot has to select an action suitable for the current situation. In an behavior architecture, action selection takes place at several stages:

- when selecting which behaviors should be active (in DD, this is computed by the activation dynamics, see 4.1 and 4.2),
- when selecting either which of the active behaviors determines the motor values (winner-takes-all), or compromising between the active behaviors (see 4.3, 4.4).
- while a behavior computes its motor controls, it can consult other concurrently active behaviors in order to avoid conflicts in the motor output (see end of 4.4).

4.1 Selection of Active Behaviors Using Activation Dynamics

In DD, action selection is mainly done by the activation dynamics. These should ensure that no conflicting behaviors are concurrently active.

Start and stop conditions have to be specified.

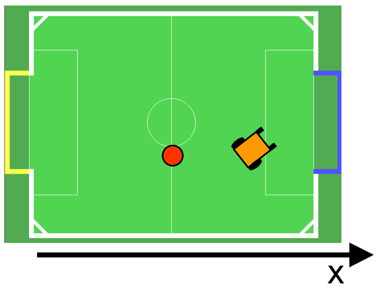
For each behavior, two kind of conditions have to be specified in its activation dynamics: a) the condition(s) under which the behavior should become active, and b) the condition(s) under which the behavior should be deactivated again. Such a condition is written as a boolean expression and characterizes a situation which the robot senses on the field, and the execution state of its behavior system.

Such conditions can be quite simple. For instance, the behavior *BehindBall* becomes active if the ball is between the robot and its own goal:

$$onFlag = Robot\_PosX > Ball\_PosX \qquad (1)$$

BehindBall is turned off when the robot is more than 100 cm behind the ball:

$$offFlag = (Ball\_PosX - Robot\_PosX) > 100 \qquad (2)$$



Behaviors may become active in quite different situations. For instance, the *DriveBack* behavior becomes active

- if the robot actively or passively collides with an obstacle at its front,
- if the robot is turning towards its own goal, the ball is near, and there is the danger that the robot might push the ball towards the own goal, or
- if the robot is beside the ball and may get into a good shooting position if it drives backward a short distance.

Forces determine the switching speed.

In a next step, forces are computed from these flags:

$$onForce = T_1\ onFlag \qquad (3)$$
$$offForce = T_2\ offFlag \qquad (4)$$

T_1 and T_2 are constants which determine the magnitude of the force. For stability reasons (not explained here) related to the differential equation below, T_1 and T_2 have to be <= 50 because our control loop runs at 50 Hz.

The *onForce* and *offForce* are used to change the activation value of the behavior. This is done using an ordinary differential equation:

$$\begin{aligned} a' &= -a(a-1)(a+0{,}5) \\ &\quad - onForce(a+0{,}1)(a-1) \\ &\quad + offForce(a-1{,}1)a \end{aligned} \tag{5}$$

Because values are updated once during a control loop, differential equations are implemented in a discrete way:

$$a = a + a' \; deltaTime \tag{6}$$

with *a* denoting the activation value and *a'* its temporal derivative. *deltaTime* is the current cycle time, about 0.02 seconds in our case (50 Hz). The first line of the equation (5) is a cubic function which stabilizes *a* either in 0 or 1 (the zero point at 0.5 is unstable), see figure 3.

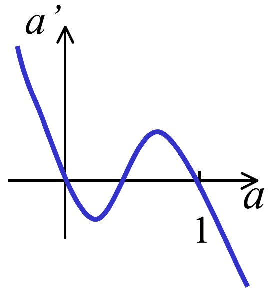


Figure 3. Stabilization of *a* if no forces are active.

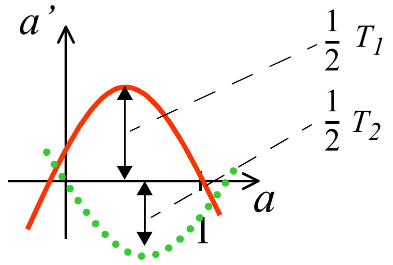


Figure 4. The change of activation *a* caused by *onForce* (solid line) and by *offForce* (dotted line).

If the *onForce* or the *offForce* are active, they change *a* according to the second and third line in the equation (5), as illustrated in figure 4.

In order to bring *a* from 0 to 1, the *onForce* has to be strong enough (the strength of the force is determined by T_1 above) and the *onForce* has to be in operation for several time steps, i.e. there is some hysteresis.

- The stronger the force, the less time steps are needed to switch *a* from 0 to 1. Fast switching is used in case of emergency, e.g., if a collision is detected.
- The weaker the force, the more time steps are needed. This requires the start condition to be stable for several time steps. If not (in case the start condition soon becomes invalid, or it was triggered by outliers in the sensory input) the activation *a* remains at 0.

The same holds for the *offForce* bringing *a* from 1 to 0.

Figure 5 shows an example: the activation of the behavior *DriveBack* ($a_{\text{DriveBack}}$) rapidly grows towards 1 if a collision occurs, but fades out slowly if there are no collision events any more.

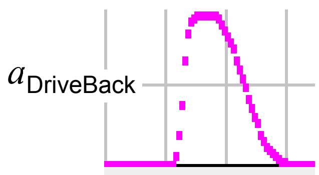


Figure 5. A strong *onforce* (great *T1*) and a weak *offForce* (small *T2*).

4.2 Higher Level Behaviors and Team Cooperation

In our current soccer behavior systems, we use higher level behaviors similar as state machine which smoothly(!) switches between ball-oriented behaviors (*ShootBall*) and homing behaviors (*GoToHomePosition*). According to this, the activation of elementary behaviors are enabled or disabled as described in 4.1.

The activation of *ShootBall* goes down if either the ball gets out of sight, or if another team mate reports a better opportunity to approach the ball successfully [6].

Higher level behaviors have to operate on a slower time scale compared to elementary behaviors in order not to cause instabilities in the behavior system. Currently, switching between higher level behaviors takes about 70 cycles of the control loop, while switching between elementary behaviors lasts about 3 to 15 cycles (depending on T_1 and T_2 as mentioned above).

But higher level behaviors can be more than just smoothly switching state machines. We plan to use higher level behaviors to implement „moods“ of the robot like aggressiveness, mobility, risk tolerance, curiosity (when seeking the ball), creativity, and willingness to learn.

4.3 Superposition of Behaviors

If several behaviors are concurrently active (i.e. their activation values are > 0), their output to the motors is superposed. In our current behavior system, each behavior produces two different outputs: a) the desired speed v_i of the robot and b) the desired direction d_i where the robot should turn. The direction d_i is a vector, so a vector summation has to be performed when computing the weighted average:

$$v_{desired} = \frac{\sum a_i \cdot v_i}{\sum a_i} \quad , \quad \vec{d}_{desired} = \frac{\sum a_i \cdot \vec{d}_i}{\sum a_i} \qquad (7)$$

where a_i are the activation values of behaviors and $v_{desired}$ and $d_{desired}$ are forwarded to motor control.

Superposition of behaviors occurs most frequently during switching from one behavior to another one (see figure 6). So while switching from one target dynamic to another, the motor commands change smoothly.

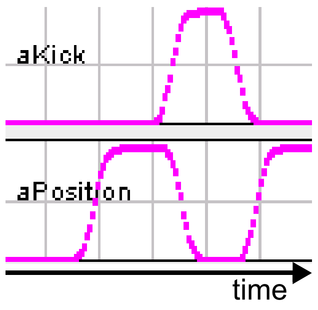


Figure 6. Switching between behaviors. aKick and aPosition denote the activation values of the behaviors *Kick* and *Position*, respectively.

Behaviors can also be superposed for longer periods of time, as long as the activation dynamics ensure that only behaviors are concurrently active which have no conflicting intentions. For instance, we had a behavior called *NoSelfGoal* which took care that the *BehindBall* behavior didn’t shoot self goals. There wasn’t any conflict because the performance of *BehindBall* behavior was in principle not handicapped by the deviations caused by *NoSelfGoal*.

4.4 Compromising with Obstacle Avoidance

Integrating obstacle avoidance via superposition caused more problems. We first tried to implement obstacle avoidance as an ordinary elementary behavior. The main problem was that after superposition, the final motor commands could still cause collisions.

We then performed obstacle avoidance in a postprocessing step after the superposition of the 'ordinary' elementary behaviors has been done. This basic idea is not new. Many other approaches perform obstacle avoidance in a postprocessing step too (e.g. [15]). The resulting architecture is illustrated in figure 7. Note that postprocessing ensures that obstacle avoidance is mandatory but this does not exclude 'ordinary' behaviors to compromise with obstacle avoidance in a way suitable for their tasks. This is illustrated at the end of this section.

Obstacle avoidance is implemented by two elementary behaviors: one for reducing the robot velocity if needed (*SlowDown*), and one for adjusting the direction of the robot in the presence of obstacles (*TurnAway*). Both behaviors consider only obstacles which lie in a parabolic scope. Depending on the current speed of the robot, this scope is enlarged or reduced. The important difference between the two behaviors is that the scope of *SlowDown* is always in the front of the robot and centered around the robot's axis, while the scope of *TurnAway* is along the direction where the robot should move.

The activation values of both behaviors depend on the distance to the nearest obstacle within the scope.

$$a_{ofCurrentSensorReadings} = \max\left(e^{(-K_1\, Dist_i)}\ K_2 \cos(Angle_i) + K_3 \right) \qquad (8)$$

$$a' = T\left(a_{ofCurrentSensorReadings} - a \right) \qquad (9)$$

with K_1, K_2, K_3, and T being constants. $Angle_i$ is the angle between the i'th obstacle and direction of the behavior's scope and $Dist_i$ is the distance of the i'th obstacle.

The target dynamics of both behaviors follow the potential field approach [12]; each obstacle issues a repulsive force and from the vector sum of all repulsive forces the maximum admissible speed (*SlowDown*) and the final turning direction (*TurnAway*) are computed. Note that *SlowDown* and *TurnAway* operate on different scopes and therefore use different potential fields.

The output of the obstacle avoidance behaviors and the motor commands produced by

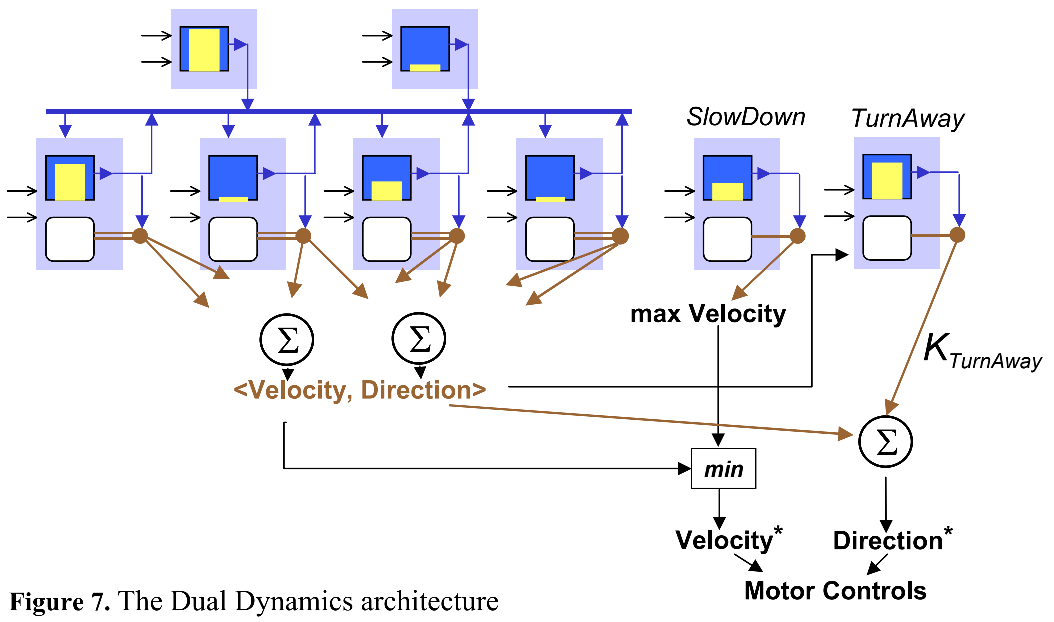


Figure 7. The Dual Dynamics architecture extended by two behaviors for obstacle avoidance.

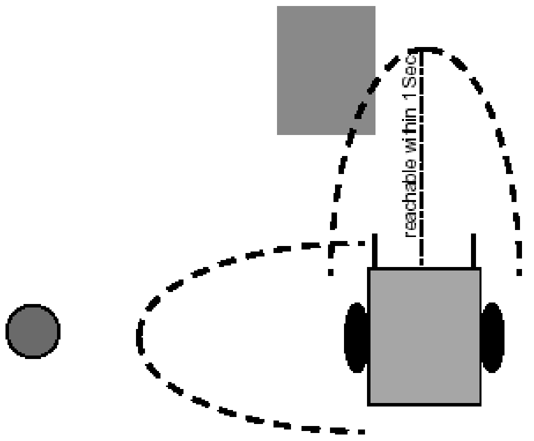


Figure 8. The scope of the *SlowDown* behavior is always in the front of the robot, while the scope of *TurnAway* points towards the target direction of the robot (e.g. towards the ball).

superposition are integrated as follows: the desired velocity is simply clipped with the maximum admissible speed computed by *SlowDown*. From the desired direction $d_{desired}$ and the direction $d_{TurnAway}$ suggested by *TurnAway*, again the weighted sum is computed.

$$\vec{d}_{final} = \frac{\vec{d}_{desired} + K_{TurnAway} a_{TurnAway} \vec{d}_{TurnAway}}{1 + K_{TurnAway} a_{TurnAway}} \tag{10}$$

$a_{TurnAway}$ is the activation value of *TurnAway*. $K_{TurnAway}$ is a constant which gives *TurnAway* the power to suppress $d_{desired}$ if obstacles are very close (i.e. if $a_{TurnAway}$ gets close to 1).

As mentioned before, concurrently active behaviors should be aware of each other. In case of obstacle avoidance, this means that behaviors should consider obstacles. For instance, if the behavior *GoHome* finds the direction towards the home position blocked, it may choose a direction towards a gap thus minimizing the interference with obstacle avoidance behaviors.

Another example is the behavior *BehindBall*: if it notices that the robot slows down because of obstacle avoidance, and that the robot is already behind the ball (even not at the desired distance), *BehindBall* deactivates itself thus allowing other ball following behaviors to proceed.

5 Results

We evaluated the performance of one robot during the semi final of the RoboCup world championship 2002 in Fukuoka. We measured the change in activation of all behaviors (e.g. the a_i). An activation is regarded to be changed if its value changes from below 0.1 (no impact on actuators) to greater than 0.5 (significant impact), or vice versa.

We then looked for slope changes in the time series of the behavior activations. A slope change occurs, if the slope becomes negative during an ascending phase, of if the slope becomes positive during a descending phase of an activation variable.If we would have used a finite state machine approach for behavior switching, state transitions form one behavior to another would have occurred at these slope changes. During 12 minutes, 336 changes in activation took place. In the same data, 1116 slope changes were discovered. This gives an impression about the stability that can be gained compared to a finite state machine approach for behavior switching.

6 Discussion

Many behavior-based architectures originate from the ideas of the Subsumption architecture introduced by Rodney Brooks in 1986 [9]. The main mechanism for action selection is a priority hierarchy where behaviors can be overruled by others having a higher priority (winner-takes-all). In the Subsumption architecture and in many other architectures (e.g. [5]), behaviors are simply enabled or disabled, thus causing discontinuities.

The Saphira architecture [14] comes closer to our approach (and somehow inspired it) as it proposes fuzzy logic as a mechanism for both computing motor output and smooth switching/ blending between behaviors („context-dependent blending“). The latter is achieved by 'context rules' like

$$\text{IF } nearDoor \wedge \neg(obstacleClose) \text{ THEN } Cross \tag{11}$$

where the relevance of the behavior *Cross* depends on the value of the fuzzy logic condition [This roughly corresponds to the activation dynamics in DD]. Tuning this context rule means tuning the parameters for computing *nearDoor* and *obstacleClose* rather than the context rule itself.

With this style of context rules, activation may be somewhere between 0 and 1, depending on the sensory input. In DD, it is also possible to write the activation dynamics in the Saphira style; we have done this for the obstacle avoidance behaviors. For instance the activation of *SlowDown* depends on how near an obstacle is and how far is in front of the robot (see 4.4).

Another style of writing activation dynamics is described in 4.1. It concentrates on writing start and stop conditions. Outliers and artifacts are compensated by the hysteresis property of the differential equation implementing the behavior switching. The hysteresis is tuned by choosing the force constants T_1 and T_2. Following the idea of switching, the activation of a behavior is soon pulled either towards 1 or 0.

In DD, activation dynamics need several time steps to be pulled from 0 to 1, or vice versa. In fact this works similar to a low pass filter. Fuzzy rules do not provide such filters and therefore can be immediately affected by outliers if no additional care (outside the fuzzy framework) is taken.

An promising idea which may replace the way superposition is now performed in DD, are techniques for solving multi objective decision problems [13]. There, each active behavior computes an objective function which computes weights for all possible actions. The final motor controls are chosen so that they maximize the sum of all objective functions.

The work of Steinhage et. al. [4] has the same credo as the Dual Dynamics approach as they model behavior systems as dynamical systems. Although the maths differ, the goal is the same: to model the behaviors of a robot in a mathematically sound and well understood framework.

7 Conclusions

Speed is a success factor for many applications. What was regarded as 'fast' yesterday will be 'moderate' tomorrow. But increasing speed may cause a robot (functioning well on slower speeds) to become instable.

Therefore techniques are needed to protect the behavior system from the effects of noisy input, to avoid self-induced instabilities, and to issue motor commands which are continuous over time. The mathematical framework of dynamical systems offers many techniques to care for system stability and therefore is the natural choice.

The Dual Dynamics architecture allows to model robot behavior as dynamical systems. It offers a set of basic techniques which are well understood and used as is. This way, the behavior designer doesn’t need to have any special mathematical background. We use the DD approach successfully for years to program our soccer robots. DD is also used to program the ’FU Fighters’, a top ranked team in the RoboCup small-size league [3]. Furthermore the integration of planners is under way [10].

In order to bridge the gap from mathematical concepts to engineering practice, we developed the Behavior Engineering Design Environment [7, 8]. It provides a dedicated editor for Dual Dynamics behaviors and offers simulation, code generation, and behavior monitoring.

References

[1] http://www.robocup.org

[2] R.D. Beer, ’Dynamical approaches to cognitive science’, *Trends in Cognitive Sciences* 4(3):91-99, 2000.

[3] S. Behnke, R. Rojas, G. Wagner, ’A Hierarchy of Reactive Behaviors handles Complexity’, *Workshop ’Balancing Reactivity and Social Deliberation in Multi-Agent Systems’* at the 14th European Conference on Artificial Intelligence (ECAI), 2000.

[4] T. Bergener, C. Bruckhoff, P. Dahm, H. Janßen, F. Joublin, R. Menzner, A. Steinhage, W. von Seelen, ’Complex Behavior by means of Dynamical Systems for an Anthropomorphic Robot’, *Neural Networks* 12:7-8, 1087-1099, 1999.

[5] P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, M. Slack, ’Experiences with an architecture for intelligent, reactive agents’ *Journal of Experimental and Theoretical Artificial Intelligence* 9(2/3):237-256, 1997.

[6] A. Bredenfeld, H.-U. Kobialka, ’Team Cooperation Using Dual Dynamics’ *Balancing reactivity and social deliberation in multi-agent systems,* Lecture notes in computer science 2103, pp. 111 - 124, Springer, 2001.

[7] A. Bredenfeld, G. Indiveri, ’Robot Behavior Engineering using DD-Designer’, *Proc IEEE International Conference on Robotics and Automation (ICRA 2001),* pp. 205-210, 2001.

[8] A. Bredenfeld, T. Christaller, W. Göhring, H. Günther; H. Jaeger, H.-U. Kobialka, P. Plöger, P. Schöll, A. Siegberg, A. Streit, C. Verbeek, J. Wilberg, ’Behavior engineering with "Dual Dynamics" models and design tools’, *RoboCup-99: Robot Soccer World Cup III*, Lecture notes in computer science 1856, Springer, 1999.

[9] R. A. Brooks, ’A robust layered Control System for a Mobile Robot’, *IEEE Journal of Robotics and Automation* 2(1):14-23, March 1986.

[10] J. Hertzberg, F. Schönherr, ’Concurrency in the DD&P Robot Control Architecture’, *Proc. Int. NAISO Cong. Information Science Innovations (ISI'2001)*, March 17-21, 2001, pp. 1079-1085, 2001.

[11] H. Jaeger, T. Christaller. ’Dual Dynamics: Designing behavior systems for autonomous robots’, *Artificial Life and Robotics*, 2:108-112, 1998. (http://www.gmd.de/People/Herbert.Jaeger/Publications.html).

[12] O. Khatib, ’Real-time obstacle avoidance for manipulators and mobile robots’, *The International Journal of Robotics Research* 5(1):90-98, 1986.

[13] P. Pirjanian, M. Mataric, ’Multiple Objective vs. Fuzzy Behavior Coordination’, in *Fuzzy Logic Techniques for Autonomous Vehicle Navigation*, D. Drainkov and A. Saffiotti, eds., Studies on Fuzziness and Soft Computing, pp. 235-253, Springer, 2000.

[14] A. Saffiotti, E.H. Ruspini, and K. Konolige. ’Using Fuzzy Logic for Mobile Robot Control’. Chapter 5 in H-J. Zimmermann, ed, *Practical Applications of Fuzzy Technologies*, Handbooks of Fuzzy Sets series, vol. 6, pp. 185-205, Kluwer Academic, MA, 1999.

[15] I. Ulrich, J. Borenstein, ’VFH*: Local Obstacle Avoidance with Look-Ahead Verification’, *IEEE Int. Conf. Robotics Automation*, pp. 2505-2511, April 2000.

Most papers can be retrieved from http://citeseer.nj.nec.com/cs