Behavior engineering with “dual dynamics” models and design tools[*]

Ansgar Bredenfeld, Thomas Christaller, Wolf Göhring, Horst Günther, Herbert Jaeger, Hans-Ulrich Kobialka, Paul-Gerhard Plöger, Peter Schöll, Andrea Siegberg, Arend Streit, Christian Verbeek, Jörg Wilberg
GMD – German National Research Center for Information Technology

Abstract

Dual Dynamics (DD) is a mathematical model of a behavior control system for mobile autonomous robots. Behaviors are specified through differential equations, forming a global dynamical system made of behavior subsystems which interact in a number of ways. DD models can be directly compiled into executable code. The article (i) explains the model, (ii) sketches the Dual Dynamics Designer (DDD) environment that we use for the design, simulation, implementation and documentation, and (iii) illustrates our approach with the example of kicking a moving ball into a goal.

1 Introduction

In the RoboCup mid-size league, robots have to kick a ball into the right direction. For many reasons, this is a hard task, which calls for robotic methods from many fields:

1. The situation on the field changes rapidly and drastically. This suggests a reactive, behavior-based approach to robot control [Brooks, 1991].
2. Kicking a moving ball is a continuous and dynamic task. Methods from continuous-time robust control (like in [Aicardi *et al.*, 1995]) are required.
3. The meaning of “...into the right direction” also varies dynamically. A self-organising, dynamical-system realization of goals and motivations seems appropriate here [van Gelder, 1998].
4. Playing football involves many different kinds of actions, with complex relations and interactions between them. A hierarchical representation of actions and action selection control is a natural approach to handle this complexity [Tyrrell, 1993].
5. Developing complex robots is done in many iterated design-redesign cycles, often with substantial modifications both on the hardware, low-level software, and control program level. State-of-the-art co-design tools can become critically beneficial [de Micheli and Gupta, 1997].

This list is certainly incomplete, but it demonstrates that designing football-playing robots is a complex, interdisciplinary challenge. From a traditional engineering perspective, this cries out for a modularized, hybrid approach, where different specialized subsystems are designed by different specialists, with well-defined interfaces between them.

However, there are indications that the classical divide-and-conquer approach is not fully appropriate for football-playing robots. A fast, autonomous robot in a continuously dynamic environment must continuously construct a stream of action from a stream of sensor information. This is connected to, but transcends, the well-known action selection problem [Maes, 1990]: *construction* is harder than *selection*. The imperative of continuously “doing the right thing” can only be met by an agent that acts “holistically”, or to use a more modest term, in an integrated fashion. It is difficult to conceive how a classical modular system can rise to this task, at least when it consists of subsystems that communicate with each other over relatively narrow channels according to strict protocols, hiding from each other most of what is going on inside them. Unfortunately, the notion, “to act in an integrated fashion”, is as vague as the term “modular”. Practical examples of robotic systems that more or less successfully construct a stream of action will help us to advance our understanding.

Building such a robotic system can only be achieved by a team of engineers that also behaves in an integrated way. At the very least, this means that there is a close, *mutually informed* collaboration – information hiding of any sort stands in opposition to the goal of building a system that can act in an integrated fashion.

Thus, a fundamental challenge for mobile robotics is to reconcile, (i) the need for *some* sort of modular design, which results from the necessity of bringing together diverse techniques and human specialists, with (ii) integratedness both in the robot and in the developing process.

At the Behavior Engineering (BE) research group in the GMD Institute of Autonomous Intelligent Systems (AiS, *http://ais.gmd.de*) we explicitly address this chal-

[*]Accepted for the IJCAI-99 RoboCup Workshop

lenge. Our approach rests on two pillars. On the one hand, we develop a mathematical model of a behavior control system, which to a certain degree integrates the points 1 – 4 mentioned in the beginning: a behavior-based approach, robust control, a dynamical systems representation of actions and goals, and a hierarchical architecture. This is the Dual Dynamics (DD) model [Jaeger and Christaller, 1998]. On the other hand, we develop and utilize a design tool that fosters a close collaboration of engineers, by providing everyone with a unified access to the entire robot control system under construction. This is the Dual Dynamics Designer (DDD) tool [Bredenfeld, 1999].

In this article, we give a quick introduction to the DD model (Section 2), describe the DDD tool (Section 3), and demonstrate its application with the example of kicking a moving ball (Section 4).

2 The Dual Dynamics model of behavior control

The Dual Dynamics scheme is a mathematical model of a behavior control system for autonomous mobile robots. It has grown from three roots: the behavior-based approach to robotics, the dynamical systems approach to cognition, and the mathematical theory of self-organizing dynamical systems. Discussions of these foundational topics can be found in [Jaeger and Christaller, 1998] [Jaeger, 1998] [Jaeger, 1997]. In the present article we concentrate on the mathematical and technical aspects of DD.

Behaviors are formalized as dynamical systems, using ordinary differential equations (ODEs). These dynamical systems interact through shared variables and certain control relations, yielding an complex control system, which in its entirety again is a dynamical system. The DD model specifies certain structural and dynamical constraints on admissible interactions and control relations between the various dynamical subsystems, which will be informally explained in this section. The formalism is mathematically specified in [Jaeger and Christaller, 1998].

The basic assumption on which DD rests is that a situated agent can work in different *modes*. Modes are coherent, relatively stable "frames of mind", which enable the agent to tune into different situations and tasks. Specifically, agents respond to sensory signal differently in different modes. In `defend` mode, a football robot would react to a ball quite differently than when it is in `attack` mode. The DD approach rests on the assumption that transitions between modes can be formally captured by bifurcations of dynamical systems. A direct implication of casting mode changes as bifurcations is that such changes are *qualitative*, discontinuous changes, not gradual ones. Our football robots do not gradually change from `defend` to `attack` mode, they either defend or attack. However, since these transitions are regulated by dynamical systems (in contrast to finite state machines), the decision point is dynamically and continuously tuned by the full wealth of incoming sensor information.

In the remainder of this section, we explain how this basic idea becomes the ordering principle for a dynamical systems engineering approach to behavior control.

The main building blocks of a DD robot architecture are *behaviors*. They are ordered in levels (fig. 1a). At the bottom level, one finds *elementary* behaviors: sensomotoric coordinations with direct access to external sensor data and actuators. Typical examples are `kick` or `fixateBall`. At higher levels, there are increasingly comprehensive behaviors. They also have access to sensoric information but cannot directly activate actuators. Their task is to regulate modes. As a first approximation, higher-level behaviors can be seen as instantiations of modes. An example of a first-level behavior in our football robots is `challenge1`, which corresponds to the first video qualification task of finding a ball and scoring a goal without opponents. Second-level higher behaviors would be even more comprehensive. For instance, `attack` would be a second-level behavior which coincides with the `attack` mode.

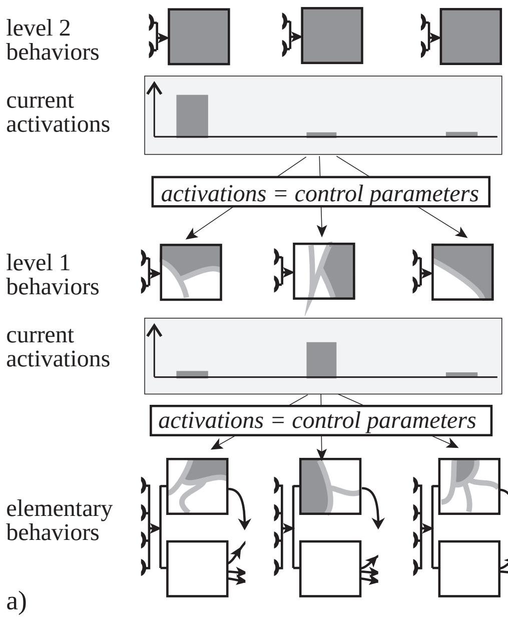
Elementary behaviors are different from higher-level behaviors in that they are made from two subsystems (fig. 1a), which serve quite different purposes. This has given the approach its name, "dual dynamics".

The first of these subsystems is called the *target dynamics.* It calculates target trajectories for all actuators which are relevant for the particular behavior. For this calculation, the target dynamics has access to every relevant sensor information, and typically includes specific sensor preprocessing. The output of the target dynamics consists of as many variables as there are motoric degrees of freedom to be controlled.

A requirement for the target dynamics is that this system should not undergo bifurcations. This is what makes elementary behaviors elementary, and provides a very helpful criterium for deciding which behaviors are, in fact, elementary. For instance, the target trajectories of `kick` in our simple wheeled football robots are likely to remain qualitatively unchanged in different instances of the maneuver. Thus, `kick` would be a good candidate for an elementary behavior. By contrast, in an anthropomorphic football robot it is likely that there will be qualitatively different kicking maneuvers different circumstances. Each of them would thus yield a separate elementary behavior.

From an engineering perspective, the target dynamics is just a motor controller for a specific task. DD is not committed to a particular type of controller – any controller which promises success is welcome. The "no bifurcation" requirement, in this perspective, means that one has a uniform control law.

The other subsystem of an elementary behavior is its *activation dynamics.* It regulates a single variable, the behavior's *activation.* The equation ruling this variable should be written in a way that the variable displays a dynamic range between 0 and 1. Intuitively, a value of 1 means that the behavior is fully active, whereas 0 means that it is completely inhibited. High values of the



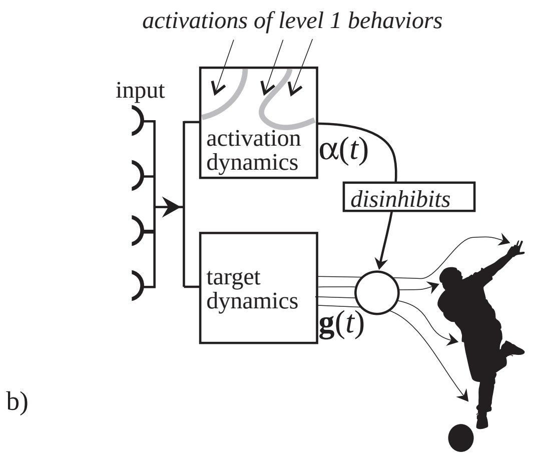


Figure 1: (a) Global structure of a DD behavior control system. At any time, every behavior has an activation. Activations of higher-level behaviors (depicted in shaded boxes) act as control parameters for the activation dynamics of lower levels. The dynamical system which maintains a behavior's activation can undergo bifurcations; this in indicated by depicting these systems as stylized "phase diagrams" (boxes with irregular partitions). A mode of the entire system is thus determined by the activations of all higher-level behaviors. (b) The target and activation subsystems of an elementary behavior.

activation mean that the target trajectories computed in the target dynamics are passed through to the actuators (cf. 1b).

The activation dynamics is allowed to undergo bifurcations. *The control parameters which induce these bifurcations are the activation variables of higher-level behaviors.* This is the core idea behind DD.

To illustrate this central point, consider the level-1 behaviors `charge` (quick advance with ball) and `freeBall` (liberate ball which has got stuck at wall or between robots). Consider an elementary behavior `bumpRetract`, a protective reflex which generally means: retract when the robot bumps into things. Standardly, the activation of `bumpRetract` jumps to 1 when the front bumper sensors are hit. However, this dynamical response changes qualitatively in different modes. Assume that the robot is charging and pushes the ball in front of itself. The bumper will be frequently hit by the ball. However, the activation of `bumpRetract` should not be triggered in this circumstance. Technically, the high activation of the level-1 behavior `charge` works on the activation dynamics of `bumpRetract` as a control parameter, pushing this dynamical system into a regime where it does not respond to bumper signals *if* the ball is seen directly in front. Now assume, by contrast, that the robot is trying to get the ball unstuck. Its level-1 behavior `freeBall` should have an activation of about 1. This value is again passed to the activation dynamics of `bumpRetract` as a control parameter, changing the response characteristics of this dynamical system. It should now indeed retract even when hitting the ball, since it makes little sense to try getting a ball unstuck by pushing it further into where it's been got stuck. In technical terms, the activation dynamics of `bumpRetract` undergoes a bifurcation when the activations of `charge` and `freeBall` change in a certain way.

These bifurcations are mathematically designed in the simplest possible way. For each relevant higher-level behavior, the activation equation is equipped with a particular additive term, which is multiplied with the concerned higher-level activation. For instance, the equation for the activation $\alpha_{\mathrm{bumpRetract}}$ would be controlled by the activations α_{charge} and $\alpha_{\mathrm{freeBall}}$ in the following way:

$$\dot{\alpha}_{\mathrm{bumpRetract}} = \alpha_{\mathrm{charge}} T_1 + \alpha_{\mathrm{freeBall}} T_2 + \ldots + \mathrm{decay}, \quad (1)$$

where T_1, T_2 are hand-designed dynamical laws which yield an appropriate activation characteristics in the `charge` and `freeBall` modes. The decay term and other details are explained in [Jaeger and Christaller, 1998].

To reiterate, only the activation dynamics subsystem undergoes bifurcations in a properly designed DD scheme. The fact that bifurcations (which are inherently difficult to master from a designer's perspective) are confined to these single-variable subsystems is critical for the transparency of DD behavior control systems.

Higher-level activation variables yield control parameters for lower-level activation dynamics. Now, in the theory of dynamical systems it is assumed that control parameters change on a (much) slower timescale than the systems they control. This implies that behaviors on different levels in a DD architecture must have different timescales, with higher-level behaviors being long-term and lower-level behaviors become active/inactive on a short-term scale. This provides the designer with a formal criterium for level organization: order higher-level behaviors according to time scales.

We emphasize that an elementary behavior is not "called to execute" from higher levels. The level of elementary behaviors is fully operative on its own and would continue to work even if the higher levels were cut off. The effect of higher levels is not to "select actions", but to change the overall, integrated dynamics of the entire elementary level, by inducing bifurcations in the activation dynamics on that level.

3 The dual dynamics design tool

Programming a football playing robot is a group activity, where different researchers are occupied with designing different branches and levels of the overall robot control system. In order to achieve an "integrated" behavior, the design process must be maximally transparent for all group members. Essentially, everybody must be able to understand and use, what everybody else designs. Therefore, we have developed a unified software developing environment, the "Dual Dynamics Designer" (DDD). Specifically, DDD provides automated editing, documentation, simulation and code generation facilities.

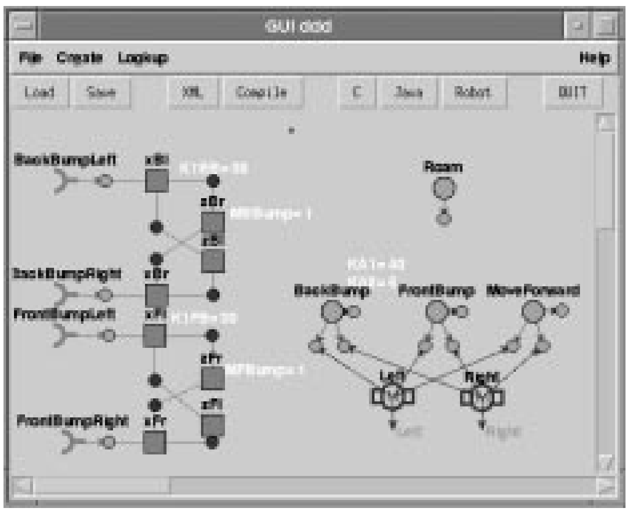


Figure 2: The primary DDD user interface. The example shows a basic `roam` behavior with bumper-based obstacle avoidance. Sensor filters and intermediate representations are on the left, higher-level behaviors (only `roam` in this case) are on the right upper part, and elementary behaviors on the right lower part of the screen.

The primary graphical user interface for designing a DD model is shown in Fig. 2. It includes icons of sensors, sensor filters and intermediate sensor representations, elementary and higher-level behaviors. Important global variables and constants (time constants, especially) appear highlighted besides the concerned icons. By clicking on the icons, context-sensitive editor windows pop up in which equations and/or ODEs can be specified in an intuitive syntax.

After designing the network of behaviors and preprocessing filters, a syntax check, global and local variable detection and checking for cyclic dependencies between equations is performed in a compilation step. Cyclic dependencies (which are unavoidable in coupled dynamical systems) are highlighted in the graphical representation on the screen. It is left to the designer to schedule a processing order for cyclically connected variables, which s/he can do by simply rearranging the icons from left to right.

By hitting the C, Java, and Robot buttons, executable standard C code, Java code, and robot C++ code is generated. The Java code can be fed into a simulation engine, which currently simulates the interaction of a single robot and a ball in an empty arena. The simulator provides a number of diagnostic traces of activations and target variables, as well as a graphical rendering of the robot's doings in the arena. We find the simulation of inestimable value in detecting "dynamo-logical" misconceptions in the designed activation and target dynamics.

The DDD tool is based on a proprietary object-oriented behavior representation, which is taken as common source for all target code generators (C, C++, Java, HTML). Therefore, the C++ code generated for our robots' onboard PCs, exactly mirrors the Java code used in the simulation. The generated documentation is common for all targets and hides language dependent implementation and syntactical details from the behavior designer. The documentation of the sensor preprocessing and DD control program allows a convenient inspection of all parts of the robot control system, ordered by various aspects.

The DDD tool itself is constructed with the Rapid Prototyping Environment APICES [Bredenfeld, 1998]. Readers interested in software engineering aspects can find more details on the software architecture and development process of the DDD tool in [Bredenfeld, 1999].

An exemplary control program (sketched in the next section), the simulator, and the documentation are available on our web server at *http://ais.gmd.de/BE/ddd/*, and can be run on Web browsers that support Java (tested on Netscape).

4 Kick a moving ball: a case study

In this section we sketch a DD behavior control system for achieving the first RoboCup-99 video qualification task. This task for a single robot consists in finding a stationary ball and scoring a goal without opponents. We made this task a bit more difficult by using a ball that rolls about while the robot tries to find and kick it.

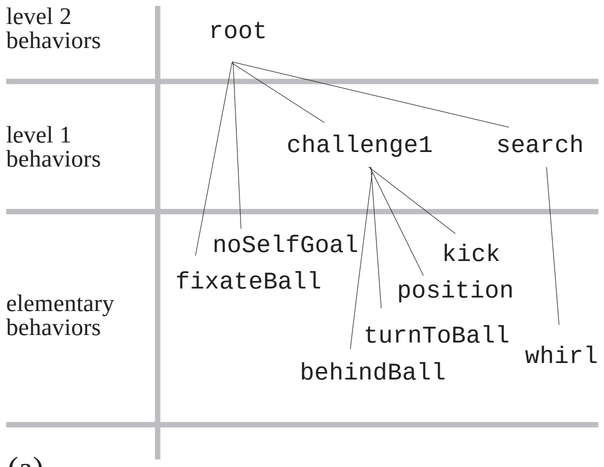
We employ a team of custom-built 2 degree of freedom, 1-PC-3-microcontroller equipped robots that rely on the well-known Newton Lab's Cognachrome system for ball and goal detection, infrared-based distant obstacle avoidance and otherwise standard bumper ring sensors and odometry. The robots do not have concavities to guide the ball. Instead, they hit the ball with their straight front portion and rely on billiard-like ball reflection. A more detailed description of our robots is given in [Kuth et al., 1998].

The difficult part of this task is kicking the moving ball into the right direction after it has been spotted. This implies hitting the ball with some appropriate (underconstrained) combination of angle, velocity and position, grounded on rather noisy estimates of ball state. This problem lies well beyond the powers of classical approaches to motor control.

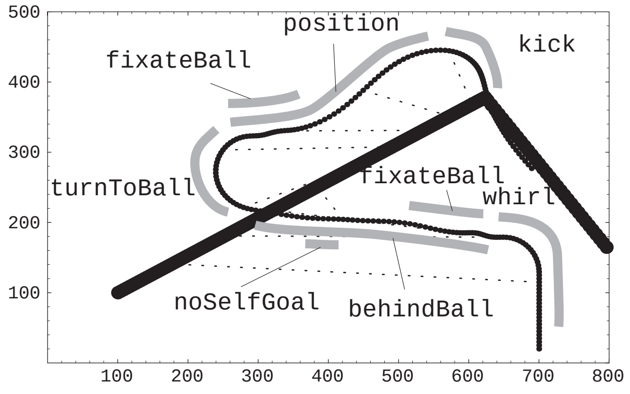
We approached the task by breaking it up into various elementary behaviors, each of which comes with its own sensor-motor control strategy. The overall goal is solved by an appropriate chaining, superposition, and inhibition of the participating activation dynamics.

Fig. 3 lists the relevant behaviors (a) and depicts a typical search–intercept–kick episode (b). The latter diagram was obtained from an simulation implemented in Mathematica in the early stages of the DDD development. Initially the robot does not see the ball, having a vision field of only ± 33 degrees. When ignorant of the ball position, the robot falls into `search` mode. Besides some obstacle avoidance behaviors, this mode basically comprehends only the elementary behavior `whirl`. The motor commands issued by `whirl` consist of a simple alternation of straight move-forwards and circles. This pattern is active until the ball is seen after a half left turn. Seeing the ball, the robot falls into `challenge1` mode. The elementary behaviors `behindBall`, `turnToBall`, `position`, and `kick` can now potentially become activated. It depends on the situation and history which behavior is triggered. In the example in Fig. 3(b), `behindBall` is activated first. Its control law says: "move toward own goal at max velocity until robot is well behind ball". The next behavior is `turnToBall` ("turn into direction where ball is expected"), followed by `position` ("move to a position from which ball can be kicked into goal") and finally, `kick` ("bump into ball with velocity that makes it billiard-bounce toward goal"). During this sequence, there are also two activation periods of `fixateBall`. This is a mode-independent (technically: `root`-mode) elementary behavior, which tries to keep the ball inside the vision cone for about 1 sec – the time needed for sampling enough video frames for a useful estimate of the ball vector. Finally, there is a brief activation of `noSelfGoal` ("if in danger of kicking ball into own goal, avoid the ball"), which however in this case has no motor effects since the robot quickly calculates that it avoids the ball anyway.

The sensor-motor control laws of these behaviors range from trivial to tricky. For instance, the motor target trajectories generated by `whirl` are actually entirely pre-



(a)



(b)

Figure 3: (a) The behaviors involved in the first qualification video task (several obstacle avoidance behaviors are omitted). (b) A successful (simulated) search – intercept – kick episode. The diagram shows head view of arena, opponent's goal on the right side. Ball (thick black line) starts at lower left corner with velocity 95 cm/sec. Robot (narrow black line) starts at lower right corner. Dotted lines connect equitemporal points on robot and ball trajectory every second. Shaded lines indicate activation periods of elementary behaviors.

coded and independent of sensor input. The target dynamics of `position`, by contrast, includes mechanisms of ball prediction and a position evaluation.

The powers (and difficulties) of the DD approach, however, lie in the activation dynamics rather than in the target dynamics. Several mechanisms, all of which are locally coded into the activation dynamics laws of the behaviors, control the interaction and trigger pattern of these activations. The most important mechanisms are:

Sensor conditions. Activate or inhibit a behavior when certain sensor input conditions are satisfied. Example: `kick` gets active "opportunistically" when ball is seen roughly in line with goal.

Chaining. Activate a behavior when certain other behaviors become deactivated. Example: `kick` gets active when the activation of `position` goes down.

Inhibition. Inhibit a behavior by the activation of others. Example: most behaviors are inhibited by obstacle avoidance behaviors.

Furthermore, these activations can be gradual (e.g., `fixateBall`'s activation grows with the uncertainty of ball estimates) or almost binary (standard case); they can have a fast dynamics (typical example: protective reflexes) or a slow one (useful for behaviors whose motor trajectors blend into one another, for example the transition from `position` to `kick` is relatively slow).

Interested readers can inspect all equations of the behavior system presented here in the automatically generated DDD documentation at *http://ais.gmd.de/BE/ddd/chall1.html*.

5 Conclusion

Identifying and coding appropriate dynamical activation schemes is decisive for the performance of a DD control system. Specifically, a simple switch-on / switch-off chaining of behaviors (like in classical action selection literature) is insufficient for a motor control task as complex and dynamic as the one in RoboCup. The phenomenology of dynamic onset, offset, and superposition of behaviors is rich and only dimly understood. We also believe that the elusive "integratedness" of situated motor control, which we mentioned in the introduction, is somehow connected to the problem of shaping appropriate activation patterns. Currently the DD framework does not spell out how the terms T in activation equations (cf. eqn. (1)) have to be written. However, certain standard terms in the activation equations begin to evolve in the BE group's everyday work. One of our major current research topics is to develop a systematic repertoire of such activation schemes, and integrate them into the DDD tool. Other robotics goups with whom we collaborate [Steinhage and Schöner, 1998] have started to work along the same lines.

Acknowledgments We would like to thank Karl-Ludwig Paap and his group for their generous assistance in designing and building the initial version of our robot hardware and onboard CAN communication system.

References

[Aicardi *et al.*, 1995] M. Aicardi, G. Casalino, A. Bicchi, and A. Balestrino. Closed-loop steering of unicyle-like vehicles via Lyapunov techniques. *IEEE Robotics and Automation Magazine*, March:27–35, 1995.

[Bredenfeld, 1998] A. Bredenfeld. APICES - rapid application development with graph pattern. In *Proceedings of the 9th IEEE International Workshop on Rapid System Prototyping (RSP 98), Leuven, Belgium*, pages 25–30, 1998.

[Bredenfeld, 1999] A. Bredenfeld. Co-design tool construction using APICES. In *Proc. of the 7th Int. Workshop on Hardware/Software Co-Design (CODES'99)*, 1999. To appear.

[Brooks, 1991] R. A. Brooks. Intelligence without reason. A.I. Memo 1293, ftp'able at http://www.ai.mit.edu/, MIT AI Lab, 1991.

[de Micheli and Gupta, 1997] G. de Micheli and R.K. Gupta. Hardware/software co-design. *Proc. of the IEEE*, 85(3):349–365, 1997.

[Jaeger and Christaller, 1998] H. Jaeger and Th. 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.

[Jaeger, 1997] H. Jaeger. From continuous dynamics to symbols. In *Proceedings of the 1rst Joint Conference on Complex Systems in Psychology, "Dynamics, Synergetics, Autonomous Agents", Gstaad, Switzerland*, 1997.

[Jaeger, 1998] H. Jaeger. Multifunctionality: a fundamental property of behavior mechanisms based on dynamical systems. In R. Pfeifer, B. Blumberg, J.-A. Meyer, and S.W. Wilson, editors, *From animals to animats 5: Proc. SAB-98*, pages 286–290. MIT press, 1998.

[Kuth et al., 1998] A. Kuth et al. Team description of the GMD RoboCup-Team. In M. Asada, editor, *Proceedings of the 2nd RoboCup Workshop*, pages 439–450, 1998.

[Maes, 1990] P. Maes. Situated agents can have goals. *Robotics and Autonomous Systems*, 6:49–70, 1990.

[Steinhage and Schöner, 1998] G. Steinhage and G. Schöner. Dynamical systems for the behavioral organization of an anthropomorphic robot. In R. Pfeifer, B. Blumberg, J.-A. Meyer, and S.W. Wilson, editors, *From animals to animats 5: Proc. SAB-98*, pages 147–152. MIT press, 1998.

[Tyrrell, 1993] T. Tyrrell. The use of hierarchies for action selection. *Adaptive Behavior*, 1(4):387–420, 1993.

[van Gelder, 1998] T. van Gelder. The dynamical hypothesis in cognitive science. *Behavior and Brain Sciences*, 21:615–665, 1998.