The Dual Dynamics Design Scheme for Behavior-based Robots: A Tutorial

Herbert Jaeger
GMD, St. Augustin
herbert.jaeger@gmd.de
herbert@arti.vub.ac.be

December 19, 1995

Abstract: This paper is a tutorial on a particular method for designing behavior-based robots, the *dual dynamics* (DD) scheme. The DD scheme guides the specification of multi-level control architectures in a format of differential equations. DD design is characterized by two principal properties. First, behaviors are construed as dynamical systems, which consist of two subsystems (hence the name, "dual dynamics"). One subsystem generates the behavior's dynamics proper, the other is responsible for activating and de-activating the behavior. The second principal property is that higher levels in the control architecture do not directly "call" lower-level behaviors to execute. Rather, a higher-level behavior can "configure" the entire lower level, which thereafter can operate fully on its own, without ongoing supervision by the higher level.

Zusammenfassung: Dies ist ein Tutorial für das "Dual Dynamics" (DD) Entwurfsschema, eine spezielle Entwurfsmethode für behavior-basierte Roboter. Das DD-Schema gibt Anleitungen zur Spezifikation von mehrschichtigen Kontrollarchitekturen durch Differentialgleichungen. Das Schema ist durch zwei hauptsächliche Eigenschaften gekennzeichnet. Erstens werden Behaviors als dynamische Systeme modelliert, die jeweils aus zwei Subsystemen bestehen (daher der Name "Dual Dynamics"). Das erste dieser Subsysteme generiert die eigentliche Dynamik des Behaviors, während das zweite seine Aktivierung bzw. Deaktivierung regelt. Die zweite Haupteigenschaft besteht darin, daß Behaviors auf höheren Ebenen in der Kontrollarchitektur Behaviors auf tieferen Ebenen nicht direkt "aufrufen". Vielmehr "konfiguriert" ein höheres Behavior die ganze nächsttiefe Ebene von Behaviors. Diese tiefere Ebene operiert danach als Ganze selbständig, ohne "Überwachung" durch die höhere Ebene.

Contents

1 Introduction

This paper is a tutorial on a particular method for designing PDL-based robots, the *dual dynamics* (DD) scheme. It is assumed that the reader is familiar with PDL. The tutorial is primarily intended to serve in practicals where students learn how to design behavior-based robots.

The DD approach models robots as continuous dynamical systems, using differential equations (DE's). The mathematical aspects of the DD design scheme are tightly coupled to aspects of agent architecture. The scheme offers a particular way of structuring a multi-level system of behaviors, while at the same time offering the mathematical tools for effectively constructing this architecture. Fig. 1 shows the scope of DD.

The DD scheme in itself gets as concrete as DE's (point 3 in fig. 1), but no further. Therefore, it can be used for designing robots that are not PDL-based, provided that DE's can be implemented. Since PDL is particularly suited to implement DE's, I will extend the treatment in the present tutorial to describing how DE's translate into PDL processes.

The DD design scheme can briefly be characterized as follows.

- Behaviors are ordered in levels, with simple, *elementary* behaviors at the bottom (e.g. `move_forward` or `turn_left`), and *complex* behaviors at higher levels (first, second, ...). For instance, at some higher level one might find a complex behavior `work` which enables a complex, situation-driven, dynamic pattern of elementary behaviors.

- Two aspects of a behavior's performance are cleanly distinguished: first, *what* the behavior does when it is active, and second, *when* it becomes active. The first aspect is referred to as *target dynamics*, the second, as *activation dynamics*. Both aspects are construed as dynamical systems. Thus, each behavior essentially is a dynamical system comprising two subsystems, and hence the catchword, "dual dynamics".

- The dynamics for activating an elementary behavior usually changes qualitatively with the "mode" a robot is currently in. The mode of a robot depends on what complex behaviors are active. If, say, the complex behavior `work` is active, and all other complex behaviors are inactive, the robot will be in a pure working mode. Several complex behaviors can be active simultaneously , which will result in the robot's being in a mixed mode.

- The mode a given level of behaviors is in typically changes on a slower time scale than that of the behaviors' activity. A work mode will typically remain present for an extended period of time, during which many elementary behaviors (like `move_forward` or `turn_left`) will have several activation episodes. The only influence that higher levels exert over lower ones is the regulation of modes. While a mode remains set, a lower level does not receive any "commands" or "calls" form higher levels. For instance, as long as `work` is active, elementary behaviors like `move_forward` or `turn_left` are activated due to being stimulated by sensor input and, possibly, by monitoring each other's activity. Given a mode, the elementary level performs autonomously and situation-driven, without any top-down control on its behaviors. This is the inheritage of the behavior-oriented tradition in robotics.

- Modes provide the robot with *expectations* about what might happen next, which in turn enable it to react appropriately and swiftly. For instance, in a pure work mode,

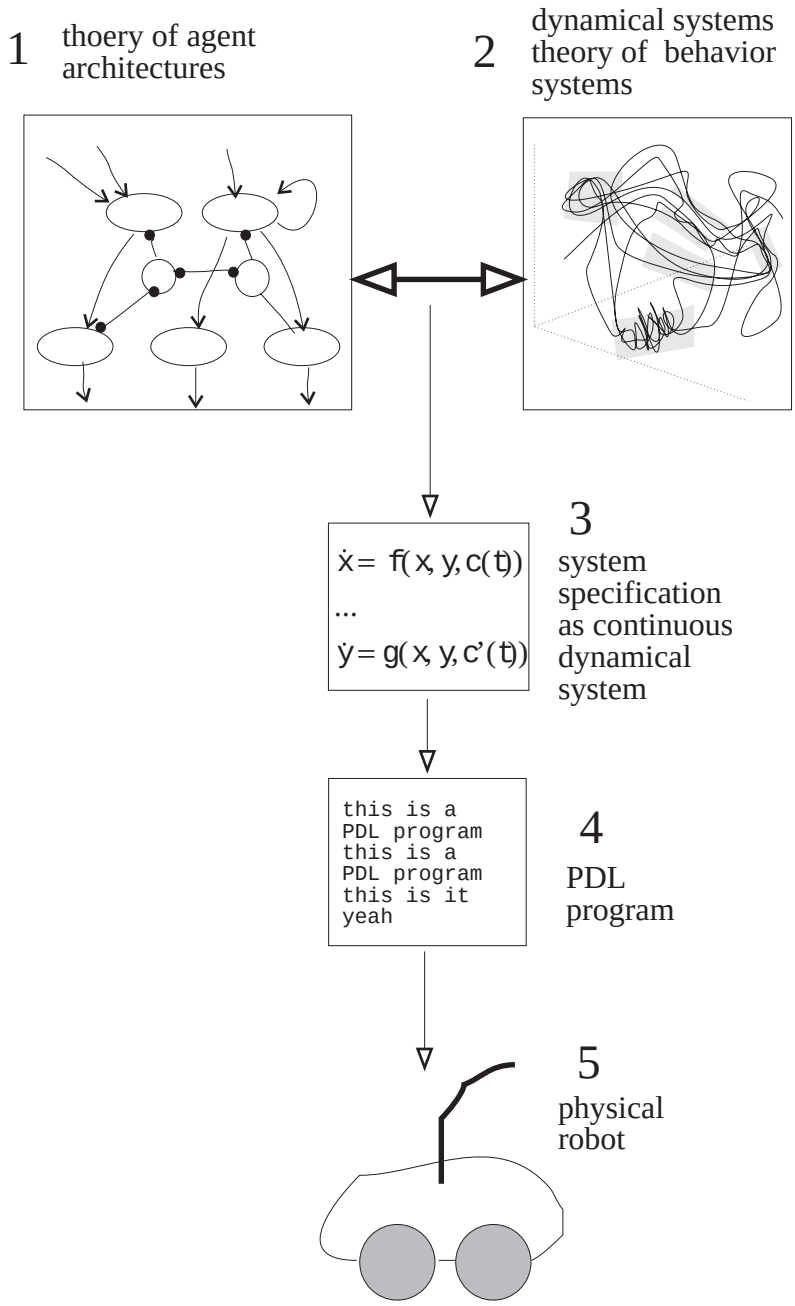

Figure 1: Contributions to robot design. The DD design scheme proposes a particular agent architecture (1) which is cast into a corresponding, "co-evolved" mathematical framework (2). The result is a behavior system specification by differential equations (3). The DD method covers (1) to (3). Translating the differential equations into PDL (4) and implementing the program on a robot (5) is not actually part of the DD methodology.

a Coke can collecting robot will perform taxis toward red spots. In a pure recharge mode, it will ignore this kind of stimulus completely and perform taxis to the white light mounted on the charging station instead. When the mode is a mixture of work and recharge, both types of taxis will compete, with the relative strength of stimuli and of `work`'s vs. `recharge`'s activation determining the outcome. The management of mode variation is the core of the DD model.

The DD design scheme aims at enabling a modular, localized fashion of designing robot control programs. The target dynamics of a behavior is mode-independent and is only designed once per behavior. Walking towards a food source in replenish-energy mode is assumed to be the *same* kind of walking as in walking towards a mate in reproduce mode. Therefore, what it means to be walking, has to be specified only *once*. The activation dynamics, on the other hand, is mode-specific. Walking towards a food source is activated by internal and external conditions that have not much in common with the conditions activating walking towards a mate. However, the design of the activation dynamics still is modular in that different, situation-specific variants of the activation dynamics can be constructed separately and incrementally. All in all, the dual design scheme supports incremental development and re-usability of behavior subsystems.

The DD design scheme combines, and develops further, ideas from architectures for behavior control that have been proposed earlier in ethology [1] and, more recently, in autonomous agents research [4] [10] [9] [2] [11]. While I defer a more detailed discussion of methodological questions to another occasion, I will briefly point out what I think are some main differences between DD and previous work:

- Modeling the activation dynamics as a dynamical system allows one to shape the activation characteristics of a behavior eplicity and with sophistication. This concerns both the onset and fadeout dynamics per se (e.g., steepness and delay of onset, fatigue, self-reinforcement, oscillatory activation), and the context sensitivity of a behavior's activation. By contrast, many classical approaches model the activation of a behavior in a more coarse-grained fashion as a *selection* boiling down to a yes-no-decision. In behavior-oriented approaches [9] or in neural network controllers, complex activation dynamics can emerge at runtime, but it is hard to explicitly design them since they cannot be easily *described* in the first place. Dynamical systems theory, the language of the DD scheme, provides rich and well-understood descriptions for temporal phenomena. This is in agreement with recent trends in cognitive science and psychology, where the importance of understanding the temporal dynamics of mental phenomena has been emphasized [6].

- The guiding notion for most classical robotic systems is action *control*. Typically, it is decided at higher levels when an action should be carried out, and then this action is "called" in a top-down fashion. By contrast, the DD scheme should be understood in terms of *configuration* rather than of control. Each mode configures the level of elementary behaviors. Once configured, elementary behaviors work together as an integrated dynamic system which is independent from top-down influences.

The tutorial begins with a brief presentation of sigmoid functions, a commodity that will be much used throughout (section 2). Then, the presentation of the DD scheme starts with a detailed description of elementary behaviors (section 3), followed by a more concise

treatment of complex behaviors (section 4). Then, the full picture of a multi-level architecture is given in section 5. Additional remarks on actuators (section 6) and translating differential equations into PDL processes (section 7) round off the tutorial.

2 Sigmoids

Sigmoid functions will occur in many examples throughout this article. This section provides a brief introduction to them. Students already familiar with the concept can skip this section.

The term, "sigmoid function", refers to any "S-shaped" functions. There are many mathematical formulae that yield S-shaped functions. We will present here the one that is most commonly used. In its basic form, it is defined by

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

It looks like in fig. 2a. It is point symmetric in $(0, 1/2)$ and has a steepest slope, at the turning point $(0, 1/2)$, of $1/4$.

It is often useful to shift the turning point to some value a on the x-axis, and to change the maximal slope (called the "steepness" of the sigmoid) to some value β, in order to get the sigmoid $\sigma_{a,\beta}$ (see fig. 2b). It is defined by the formula

$$\sigma_{a,\beta}(x) = \frac{1}{1 + e^{-4\beta(x-a)}} \tag{2}$$

Sigmoids are particularly useful to equip continuous systems with a conditional switching-like characteristics, or, which amounts to the same thing, with a kind of "if-then-else" mechanism. It often occurs that we wish a dynamical system to behave like *"if the value of the control parameter C is below threshold a, then behave according to type A, else behave according to type B."*. This can be achieved with a differential equation of the kind

$$\dot{x} = (1 - \sigma_{a,\beta}(C)) \cdot A(x) + \sigma_{a,\beta}(C) \cdot B(x) \tag{3}$$

The parameter β determines how decisively the switching between cases is. For small values of β, the alternatives A and B blend smoothly into one another, while for big β, one can generate almost "digital" switching.

By suitable combination of sigmoids, one can mimick almost any kind of conditional expressions in differential equations: multiple conditions, boolean expressions, etc.

3 The makeup of an elementary behavior

The basic building block within the DD design scheme is a behavior. Behaviors are arranged in several levels. On the lowest level, *elementary* behaviors are simple sensomotoric regulations, like `move_forward` or `turn_left`. On higher levels, behaviors correspond to increasingly integrated *modes* which refer to the interactions of lower-level behaviors with each other and with the environment. For instance, the elementary behaviors `move_forward` and `turn_left` might occur, among others, as relevant in a mode determined by a `phototaxis` behavior on the next higher level. In order to simplify our way of talking, we will say that `move_forward` and `turn_left` *participate* in `phototaxis`. `Phototaxis`, in

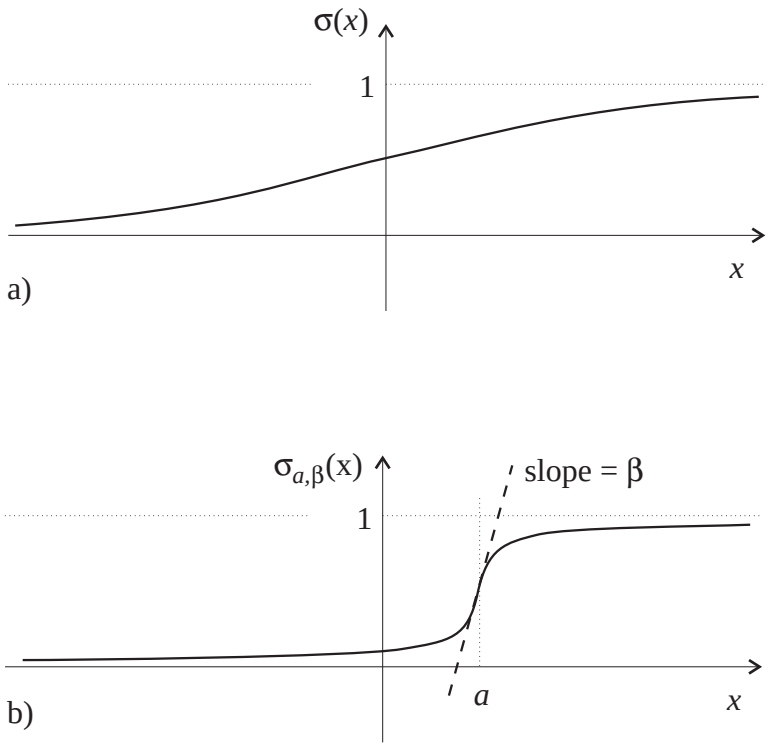

Figure 2: Two sigmoids.

turn, might participate in even higher-level behaviors like `work` or `replenish_energy`. The mathematical makeup of behaviors is similar on all levels. However, the effects of a behavior's activity are different on the elementary vs. higher levels. In this section I explain how elementary behaviors are specified as dynamical systems.

Mathematically, an elementary behavior is made up from two dynamical systems, plus a product term in which the effects of these two systems are combined. The two dynamical systems generate the *target dynamics* and the *activation dynamics*, respectively. The former specifies *how* the actuators should work, while the latter determines *when* and *how strongly* the behavior is to be activated. I will now introduce the target dynamics, activation dynamics, and the product term, in this order.

3.1 Target dynamics

In order to illustrate the central distinction between target and activation dynamics, I will use the example of an elementary behavior `walk_forward`, which one might assume to exist in an insect-like robot. When this behavior is active, it should issue to the actuators (of which there are many in this case) signals which induce a walking pattern to be physically executed. The signals will vary in time, on several accounts. First, walking is in itself a rhythmic activity, which implies that signals to most actuators will have a periodic component. Second, there might be small-scale obstacles detected by tactile sensors on the extremities, which should induce local accomodations of the basic walking pattern in the legs concerned. While a good deal of a perturbation might locally be coped

with in the leg affected, the other legs must to some extent accomodate to that local variation, which means that the overall pattern of `walk_forward` is modulated. Third, external conditions might change on a larger scale that globally afflicts the walking pattern. For instance, walking on different slopes might induce an accomodation of walking velocity and stance amplitude. All of these dynamic contributions to the actuator signals (basic rhythm, externally conditioned variations) concern *how* the actuators should do their job while walking is active. Therefore, all of this is handled in the target dynamics. It becomes clear from this example that the target dynamics is generated in an *open* system, which is modulated by input from sensors and other behaviors.

In order to preclude a likely misunderstanding, I should emphasize that the target dynamics is unrelated to *goals*. A goal, in the standard sense of the term, is a final state of affairs that is typically satisfied *after* an action, or a sequence of actions, has been executed. By contrast, the target dynamics prescribes how the behavior should work at each moment *while* it is active. No explicit handling of classical goals exists in the DD scheme.

The target dynamics of an elementary behavior gives a target trajectory $g_i(t)$, where $i = 1, \ldots, n$, for each of the n degrees of freedom (df) of actuators involved. Thus, the target dynamics yields an n-dimensional vector $\mathbf{g}(t)$ which has the $g_i(t)$ for its components. While in a Lego vehicle, $\mathbf{g}(t)$ will be typically two-dimensional (two motors with one df each), $\mathbf{g}(t)$ can have a considerable number of dimensions for less simplicistic devices (say, a six-legged walking machine or a redundant robot arm), not to mention the number of df's that are involved in biological systems.

Fortunately, this does not imply that we have to come up with a dynamical system of n dimensions in order to generate $\mathbf{g}(t)$. It is a general observation that (bio-)mechanical systems with many df's actually can be reduced in their dynamics to a low-dimensional system. The reason is that the many mechanically possible df's are coupled and do not vary independently from each other. One popular physico-mathematical approach to understanding this reduction in dimensionality is the slaving principle [3]. This topic deserves a more detailed discussion than is possible here. Let me only mention that motor action patterns involving many dozens of muscular and joint df's have been described in terms of low-dimensional systems in human psychophysics [8] [7].

DD target dynamics makes use of this potential for simplification by generating $\mathbf{g}(t)$ in two steps. In the first step, a low-dimensional dynamical system produces a *target representation trajectory* $\mathbf{x}(t)$, from which the high-dimensional target trajectory that regulates the effectors is computed in a mechanical fashion. The target dynamics of a behavior thus comes in two (sets of) equations:

$$\dot{\mathbf{x}} = G(\mathbf{x}, \text{sensor-input}(t), \ldots) \tag{4}$$

$$\mathbf{g}(t) = \Gamma(\mathbf{x}) \tag{5}$$

The dynamical system (4) can be considered as a *representation* of the behavior's target dynamics, albeit not a symbolic one. It carries the "essentials" necessary for qualifying how the behavior should work. For a walking behavior in a six-legged walking machine, for instance, it might be sufficient to keep record in (4) of ground slope, position of the walker with respect to small obstacles, and acceleration/deceleration (in systems with noticeable inertia, the target dynamics during acceleration conditions might differ qualitatively from

constant-velocity dynamics).

In mechanically simplicistic systems like 2-df Lego robots, it is not worth going through the modeling effort of splitting (5) from (4). A single dynamic system

$$\dot{\mathbf{g}} = G(\mathbf{g}, \text{sensor-input}(t), \ldots), \tag{6}$$

or even a simple functional expression like

$$\mathbf{g}(t) = G(\text{sensor-input}(t), \ldots) \tag{7}$$

will suffice in such cases. Here, the "representation" of the target trajectory coincides with the target trajectory itself. An example for such a simple target dynamics is the following version of the elementary behavior `turn_left`. Assume that the voltages V_{right} and V_{left} currently issued to the right and left motor, respectively, are known to the control program. One intuitive version of what it means to "turn left" is the following prescription: "increase the current voltage of the right motor, and decrease the current voltage of the left motor, both by a default amount D". This would yield the target dynamics

$$\begin{aligned} g_{left}(t) &= V_{left}(t) - D \\ g_{right}(t) &= V_{right}(t) + D \end{aligned} \tag{8}$$

Some mysterious dots appear within the rhs. of (4). They represent optional time-varying input into the trajectory representation dynamics, which comes from other sources than sensors. The question of which sources of information can be used as input into an elementary behavior is answered in section 3.3.

3.2 Activation dynamics

Now let us turn to the activation dynamics. It is responsible, in our example, for *when* walking is to occur. It regulates a single parameter, α, which determines whether the behavior is inhibited ($\alpha \sim 0$) or active ($\alpha \sim 1$), or something in between. Thus, while the target dynamics potentially is high-dimensional (each relevant actuator requiring a dimension), the activation dynamics is expressed in a one-dimensional trajectory.

However, the impression would be misleading that this is a simple system. It can be very complex. This is due to the fact that behaviors are activated differently in different modes. Let us consider an example. It leans on experiments carried out at the VUB AI Lab. Assume that a Lego vehicle is to be designed, which has to work in an obstacle-cluttered arena, and which has to autonomously recharge its batteries at a charging station. In the VUB setup, "working" means that the robot has to repeatedly push against cylindric "push-boxes", which are distributed in the arena, and which can be detected by the vehicle due to modulated red light that they emit. The charging station can be detected by an unmodulated white light mounted on it.

Now, if the vehicle has nearly emptied its batteries, it will be desperately "hungry" and should be in a rather pure recharge mode. In this case, the elementary behavior `turn_left` should be activated when the white light of the charging station is perceived at the left.

By contrast, if the batteries are full and the robot is in work mode, `turn_left` should *not* be triggered by a white light at the left (at least not easily). Rather, `turn_left` should

become active when there is a modulated light source detected at the left side, indicating a push-box waiting for work to be spent on.

Thus, in different modes we usually find different conditions which should activate an elementary behavior. In order to accomodate for this mode-specific differentiation, the dynamical system that regulates a behavior's activation dynamics can become quite complex.

Technically, this accomodation to different modes is done in the following fashion. Assume that we have $i = 1, \ldots, m$ different "pure" modes to account for. Then, for an elementary behavior we have to construe (up to) m different kinds of activation conditions. To this end, to each "pure" mode we devote an additive term T_i in the activation dynamics of the elementary behavior. The various such terms are weighted with factors μ_i. These factors vary in time (due to top-down mechanisms that will be explained in subsequent sections). Thus, the activation dynamics of an elementary behavior is realized by an equation of the following kind:

$$\dot{\alpha} = \mu_1 T_1(\alpha, \ldots) + \ldots + \mu_m T_m(\alpha, \ldots) \tag{9}$$

When the agent is in a "pure" mode, exactly one of the μ_i should be roughly equal to 1, the others roughly equal to 0, leaving only one T_i to determine the activation dynamics for the time being. "Mixed" modes occur when several μ_i are appreciably greater than 0. This scheme supports an incremental design of the activation dynamics, since further terms T can be added when the behavioral repertoire is expanded to more modes.

For a simple example of (9), consider the elementary behavior `turn_left` in our work and recharge mode example. Let us first deal with the activation dynamics for the work mode. In intuitive terms, we would like to shape the activation of `turn_left` along the following guidelines:

(A) `turn_left` should be activated only if the signal ML_{left} of the left modulated light sensor is above some threshold min, i.e. if $ML_{left} \geq min$. This precludes triggering by background noise.

(B) `turn_left` should be activated only if ML_{left} is greater than the signal ML_{right} of the right modulated light sensor, i.e. only if $ML_{left} - ML_{right} =: diff > 0$.

(C) The activation of `turn_left` should not be all-or-nothing. Rather, it should decrease as the alignment to the modulated light becomes better, i.e. as *diff* goes to zero. The reason for this is to prevent overshoot and oscillations. More specifically, we wish the activation strength to depend on *diff* according to a function f as sketched in figure 3 (a function of this kind can easily be derived, e.g. from a sigmoid).

The requirements (A) - (C) can be met e.g. by the following specification of an activation dynamics:

$$\dot{\alpha} = k\sigma_{min,\beta_1}(ML_{left}) \cdot \sigma_{0,\beta_2}(diff) \cdot (f(diff) - \alpha) \tag{10}$$

$$=: T_{work}(\alpha, ML_{left}, ML_{right}) \tag{11}$$

k is a time constant that adjusts the overall time scale of the dynamics, $\sigma_{a,\beta}$ denotes a sigmoid with turning point in a and steepness β, ML_{left} and *diff* are as introduced before,

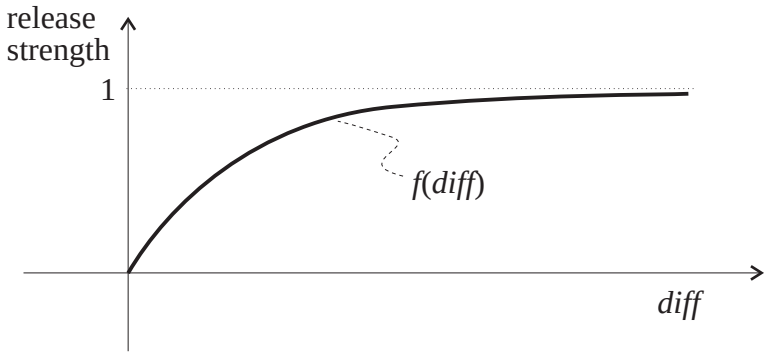

Figure 3: How the activation of `turn_left` should depend on *diff*.

and f is a suitable function that should look like in figure 3. The three factors in the rhs. of (10) account for (A), (B), and (C), in that order.

Since time constants like k will be used in nearly every formula to come, I will not start trying keeping them cleanly apart by naming or indices. Thus, if some k appears in some formula, it's a constant local to that formula.

A completely analogous consideration can be made for the recharge mode. The only difference is that we have to use white light signals WL_{left}, WL_{right} instead of ML_{left}, ML_{right}. This gives us the following activation dynamics for the case of a pure recharge mode:

$$\dot{\alpha} = T_{recharge}(\alpha, WL_{left}, WL_{right}) \tag{12}$$

Assembling the pure mode activation dynamics (10) and (12) in DD fashion lets us arrive at the following instantiation of (9):

$$\dot{\alpha} = \mu_{work} T_{work}(\alpha, ML_{left}, ML_{right}) + \mu_{recharge} T_{recharge}(\alpha, WL_{left}, WL_{right}) \tag{13}$$

Mode regulation is achieved through varying μ_{work} and $\mu_{recharge}$. If $\mu_{work} = 1$ and $\mu_{recharge} = 0$, `turn_left` reacts to sensor input according to a pure work mode. If $\mu_{work} = 0$ and $\mu_{recharge} = 1$, pure recharge mode is on. These are the "clean" cases that one probably has in mind when one designs (10) and (12).

However, in principle μ_{work} and $\mu_{recharge}$ can assume all combinations of values between 0 and 1 during operation time, so `turn_left` can react to white and modulated light in "mixed" modes. It is up to the designer to en- or discourage mode mixing. A simple mechanism for fixing the propensity toward mode mixing will be presented in the next section.

This regulation of μ_{work} and $\mu_{recharge}$ is effected top-down from the complex behaviors `work` and `recharge`. We will treat this topic in detail in the next section.

The activation dynamics as introduced so far would work reasonably in many cases, but things can go awry when *all* of the μ_i in (9) drop to zero. This would result in the activation α just staying where it last was, which might mean that a behavior stays active even though it does not participate in the currently active (if any) mode. This case is provided for in a standard way by adding a decay term to (9), which brings α down to zero after the μ_i have gone to zero.

$$\dot{\alpha} = \mu_1 T_1(\alpha, \ldots) + \ldots + \mu_m T_m(\alpha, \ldots) - \alpha k \prod_{i=1,\ldots,m} (1 - \mu_i)^r \tag{14}$$

In this final form of the activation dynamics, k is a constant that determines the decay rate. The exponents r determines the degree of nonlinearity for when the decay sets in. A standard value would be $r = 2$. Note that decay is not a "background" process that is active all the time. Rather, decay of α sets in only when other influences are absent. Thus, the decay rate k can be selected appreciably great, in order to ensure a rapid fading out of α when it becomes irrelevant.

The example (13) should be augmented by a decay term as in (14).

3.3 Input to the target and activation dynamics

Now we will deal with the notorious dots appearing in the equations of the target dynamics (4) and the activation dyanmics (14). These dots represent input to the dynamical systems concerned.

An obvious source of input is from sensors. We will denote it by $s(t)$.

The DD scheme has a single, iron rule concerning input to behaviors: *The only input that comes top-down from higher-level behaviors is the mode regulation parameters μ_i in the activation dynamics.*

This rule implicitly allows to use *any* kind of input for a behavior as long as it doesn't come from higher-level behaviors. In particular, an elementary behavior can receive information about what is going on in other elementary behaviors. For instance, an elementary behavior B_1 could receive from another elementary behavior B_2 the activation $\alpha_2(t)$, the derivative of the activation $\dot{\alpha}_2(t)$, the representation of the target trajectory $\dot{\mathbf{x}}_2(t)$ and its derivative, or the target trajectory $\mathbf{g}_2(t)$ itself, or higher derivatives thereof, or time-delayed versions, or whatever one finds useful in a particular instance.

Among all of these, the activations of other elementary behaviors are probably the most universally useful. Therefore, we will explicitly include them into the system equations (4) and (14), tacitly assuming that other parameters of the kind just mentioned can be received if necessary.

Also, we assume as a standard case that the activation dynamics of a behavior depends on the target dynamics and vice versa. I.e., in an elementary behavior B_j we standardly note down $\mathbf{x}_j(t)$ as an input for the activation dynamics, and $\alpha_j(t)$ as an input for the target representation dynamics.

Assembling these remarks, and assuming that there are n elementary behaviors, we get the following full version of (4) and (14) for the elementary behavior B_j:

$$\dot{\mathbf{x}}_j = G_j(\mathbf{x}_j, \mathbf{s}_j(t), \alpha_1(t), \ldots, \alpha_n(t)) \tag{15}$$

$$\begin{aligned} \dot{\alpha}_j = {} & \mu_1 T_{j,1}(\alpha_1, \ldots, \alpha_n, \mathbf{x}_j, \mathbf{s}_j) + \ldots \\ & + \mu_m T_{j,m}(\alpha_1, \ldots, \alpha_n, \mathbf{x}_j, \mathbf{s}_j) \\ & - \alpha_j k \prod_{i=1,\ldots,m} (1 - \mu_i)^{r_j} \end{aligned} \tag{16}$$

3.4 The product term

The target dynamics $\mathbf{g}(t)$ and the activation dynamics $\alpha(t)$ are combined in the following product assignment:

$$\mathbf{u}(t_\nu) = k\alpha(t_\nu)(\mathbf{g}(t_\nu) - \mathbf{z}(t_\nu)), \tag{17}$$

where $\mathbf{z}(t_\nu)$ is the perceived current state of the actuators, $\mathbf{g}(t_\nu)$ is the target trajectory as delivered by the target dynamics, $\alpha(t_\nu)$ is the current activation as delivered by the activation dynamics, k is a gain constant particular to each behavior, and $\mathbf{u}(t_\nu)$ is the incremental signal issued from the behavior to the actuators which induces them to increment (if $\mathbf{u}(t_\nu) > 0$) or decrement ($\mathbf{u}(t_\nu) > 0$).

It is important to understand how (17) works mathematically. Since many things come together in this product term, I will explain it with some care.

The first thing to note about (17) is that it is evaluated at discrete time steps t_ν. This is a technical conveniance, not a fundamental property. It makes (17) blend in more easily with standard robot control circuitry, which typically works with a discrete update cycle. The corresponding differential (i.e., continuous) version of (17) would read $\dot{\mathbf{u}} = \alpha(t)(\mathbf{g}(t) - \mathbf{z}(t))$.

In simple robots (e.g. typcial PDL-based Lego vehicles), actuators are more often than not controlled in an open loop, which means that $\mathbf{z}$ coincides with the signal issued to the actuator. The evaluation of (17) would in this case proceed in the following steps:

1. At time t_ν, read the current value of the actuator quantities $\mathbf{z}$, and compute the current values of $\mathbf{g}$ and α.

2. Calculate the value of the rhs. in (17) and assign it to $\mathbf{u}$.

3. At the end of the ν-th cycle, i.e. at time $t_{\nu+1}$, add $\mathbf{u}$ to the value of the actuator quantities $\mathbf{z}$, which they had at the begin and during the cycle.

The second thing to note about (17) is that it concerns vectors (remember that a behavior tends to affect many actuators). In order to understand this equation, however, it will suffice to assume that we have only a single 1-df actuator to deal with, in which case (17) reduces to a scalar assignment $u(t_\nu) = k\alpha(t)(g(t_\nu) - z(t_\nu))$.

In order to understand the essence of (17), let us assume that $k = 1$ and $\alpha \equiv 1$, so we can ignore these parameters for a moment. (17) represents the core of a regulatory loop which tries to make the actuator follow the target trajectory. It evaluates, at each time step, how much the perceived state $z(t_\nu)$ of actuators differs from the desired target state $g(t_\nu)$, by computing the difference $g(t_\nu) - z(t_\nu)$. $u(t_\nu)$ is then set equal to this value, and sent to the actuator, which increments its state if $u(t_\nu) > 0$ and decrements if $u(t_\nu) > 0$. This is a regulatory loop: if the perceived state $z(t_\nu)$ is smaller than the target state, the difference is positive; hence, $u(t_\nu)$ is positive; hence, the state of the actuator increments; hence, the perceived state z should increase its value, which is what one expects from a regulatory mechanism.

This is the basic idea. It implements a particularly simple, but not necessarily the most effective type of regulatory feedback loop. However, due to the fact that α and g are time-varying, this loop already is loaden with more intricacies than we might expect at

first sight. Before we explore the temporal properties of (17) a bit further, I will say some words about the "perceived" current state z of the actuator.

It is important to be aware of the distinction between an actuator's state and the state of affair that one actually wishes to govern by the actuator. For instance, when we wish to maintain a constant forward velocity in a simple two-motor Lego vehicle, we have to distinguish between the motors' current state (which can e.g. be specified in terms of voltage) and the current robot velocity, which is the state of affair that we actually want to control. Due to slippage, the motor state and the velocity of the robot can be quite different things. At some place or other, in the design of robot control systems one has to account for this possible mismatch, since one can only directly influence actuators, although one wishes to control the external state of affairs. In the DD design scheme, the load of this mismatch handling is placed on the target dynamics subsystem. Using its external sensor input, it has to adapt the target trajectory, which prescribes the actuator state. For instance, if the Lego robot deviaties to the right due to slippage on the right wheel, the target trajectory subsystem of a `move_straight_forward` behavior would have to increase the value of the right motor target trajectory. The regulatory loop in the product term, then, only has to take care of making the motor follow the target trajectory.

Even if the regulatory task of (17) is reduced to maintaining the actuator state, this may be far from trivial. One of the difficulties is that the current actuator state may be hard to ascertain. This is the case, for instance, when propriosensors give delayed responses. In such cases, coming up with sound values for the "perceived" current actuator state, $z(t)$, may involve complex estimation procedures, e.g., Kalman filters. Interestingly, there are indications that in mammals some processing within the cerebellum is devoted to this task [5] [12]. Fortunately, the current state of motors in Lego vehicles can be measured without much ado, which is why we won't have to bother about $z(t)$ very much.

In order to get a better understanding of the temporal properties of (17) when α and g are time-varying, let us review the properties of the mathematical archetype of (17). This is the one-dimensional dynamical system (18), which describes the exponential approximation of a limit point g:

$$\dot{x} = \tau(g - x) \tag{18}$$

Solution curves for this system are shown in fig. 4a1-a3. They reveal the effect of choosing different values for the constant τ. The greater τ, the faster trajectories converge to g. The constant τ is often called *time constant*. The symbol τ is customary in dynamical systems theory for time constants. In the DD framework, however, I use the symbol α instead, as a mnemotechnic aid to indicate that the time constant here determines the **a**ctivation of a behavior.

When g is allowed to vary in time, i.e. when we consider a system

$$\dot{x} = \tau(g(t) - x), \tag{19}$$

different time constants lead to qualitatively different behaviors. When τ is sufficiently great, the convergence rate of trajectories to $g(t)$ is big enough to ensure that $g(t)$ is nicely followed by the system trajectories. Of course, the faster $g(t)$ is allowed to vary, the greater τ must be in order to warrant a target following dynamics like in fig. 4b1. If τ is not sufficiently great, a cleanly following isn't possible any longer (fig. 4b2). If, finally, τ

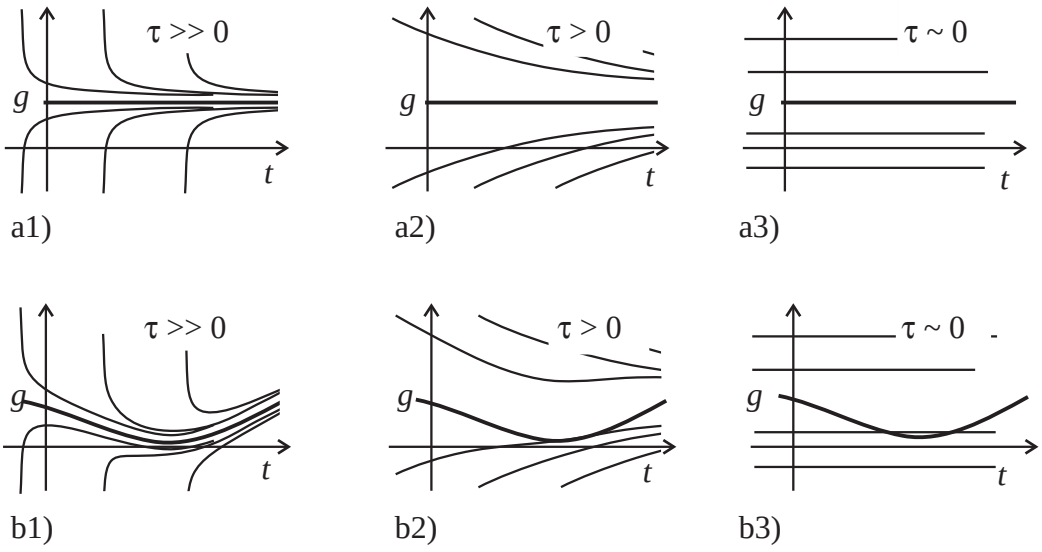

Figure 4: The effects of the time constant τ on exponential approximation of a fixed point g (a1 – a3) and of a time-varying target trajectory g (b1 – b3).

is about 0, system trajectories degrade to constants which don't take any notice of $g(t)$ (fig. 4b3).

The time constant τ from fig. 4b1 corresponds to the activation factor $\alpha(t_\nu)$ in the product term (17). The situation in (17) differs from the situation in (19) in that the latter is a differential equation with x appearing on both sides, whereas in (17) instead of a single system variable we have an output signal u on the lhs and a perceived actuator state z on the rhs, with a possibly complex physical connection between u and z. Yet, in principle (19) and fig. 4b tell us how we should understand (17).

For the purposes of behavior control in robots, the cases from fig. 4b1 and b3 are of particular interest. When the activation factor is sufficiently great, the product term (17) yields an overall dynamics of target following like in fig. 4b3. When, by contrast, the activation factor is about zero, the actuator is not affected by the behavior at all, although the target dynamics $g(t)$ might show strong fluctuations. The case shown in fig. 4b2 is what system analysts and designers are afraid of: the system behavior is unruly and cannot be understood either in terms of the target trajectory $g(t)$ or in terms of simple constants. Thus, if one wishes to design behavior systems such that they behave in an understandable way, one should make sure that the activation dynamics yields either sufficiently large values of α, or values that are near 0, but not values that vagabond in between. For the sake of simplicity, we will assume that "sufficiently large" means "about equal to 1" – this normalization can be made to work by setting the gain k in (17). Thus, the task of engineering a transparent activation dynamics implies that we have to look for more or less bistable, $0-1-$valued dynamics for α. This is not necessarily the most effective kind of activation dynamics; it is just a starting point for making things transparent for a human designer.

It will often be the case that several elementary behaviors issue signals u to the same actuator. A strategy for transparent engineering is to ensure that at a given time, only one

of the behaviors tries to control the actuator, i.e., only one of the behaviors is activated. This means that one has to design winner-take-all mechanisms for the activation dynamics of behaviors that can be simultaneously relevant in a situation.

Again, this strategy for neatly selecting one behavior at a time is helpful for keeping matters transparent, but does not necessarily lead to optimal solutions. However, it is interesting to note that in animal ethology, it is standardly observed that behaviors are mutually inhibitory, i.e. that there is a strong winner-take-all principle ruling between them [1, 10, 2].

For the sake of convenience, I repeat all equations (cf. (8), (13)) from our `turn_left` example, in their final form:

target dynamics (simplified form, without representation system $\mathbf{x}$*)*:

$$\begin{aligned} g_{left}(t) &= V_{left}(t) - D \\ g_{right}(t) &= V_{right}(t) + D \end{aligned}$$

activation dynamics:

$$\begin{aligned} \dot{\alpha} &= \mu_{work} T_{work}(\alpha, ML_{left}, ML_{right}) + \mu_{recharge} T_{recharge}(\alpha, WL_{left}, WL_{right}) \\ &\quad -k\alpha(1-\mu_{work})^r(1-\mu_{recharge})^r \end{aligned}$$

product term:

$$\begin{aligned} u_{left}(t_\nu) &= k\alpha(t_\nu)(g_{left}(t_\nu) - V_{left}(t_\nu) \\ u_{right}(t_\nu) &= k\alpha(t_\nu)(g_{right}(t_\nu) - V_{right}(t_\nu) \end{aligned}$$

where $u_{left}(t_\nu)$ and $u_{right}(t_\nu)$ are voltage increments added to the current voltage at the left and right motor after the time cycle ν, k is a suitable, fixed time scaling factor whose value will depend on the duration of a time cycle (the k's in both equations above should be equal), and $V_{left}(t_\nu)$, $V_{right}(t_\nu)$ are the voltages at the left and right motor at the beginning of the time cycle ν.

4 Complex behaviors

The DD scheme allows to construct multi-level architectures with any number of levels. Since higher levels are formally similar to each other, we can restrict this presentation to the case of a two-level behavior system (the bottom level of elementary behaviors plus the first level of *complex* behaviors). Thus, in this section the phrase "all complex behaviors" actually means "all behaviors from the first level of complex behaviors", etc.

From a formal point of view, complex behaviors are quite similar to elementary ones. Like the latter, they consist of a target and an activation subsystem, whose effects are combined in a product term. The central difference between elementary and complex behaviors is that the former control actuators, whereas the latter control behaviors from the next lower level (elementary behaviors in the two-level case which we are focussing).

Also, the meaning of "control" differs in the two cases. Whereas elementary behaviors issue signals to actuators that directly put them to work, complex behaviors issue "mode instructions" to elementary behaviors. They don't "call" elementary behaviors to execute; rather, they instruct them under which situative circumstance they should activate themselves. To put it in a nutshell, elementary behaviors *control* actuators, while higher-level behaviors *configure* lower levels, by determining their mode. This is one of the central ideas in the DD scheme. It distinguishes it from other hierarchical architectures for action selection, where higher-level behaviors actually call lower-level behaviors to execute.

Although complex behaviors come with a fully-fledged target dynamics in the general case, I will restrict this section to the case where the target dynamics is trivially $\equiv 1$. This means that the output of a complex behavior solely reflects its activation dynamics, which leads to the further simplification that we can dispense with the product term. Therefore, all we shall care about in this section is the activation dynamics of complex behaviors. This makes sense for two reasons. First, didactically: we can concentrate on the fundamental idea of "configuring" the level of elementary behaviors. Second, for the kind of Lego vehicles that are the current primary aim of the DD design scheme, non-trivial target dynamics in complex behaviors seem to be unneccessary.

Before we get going, a notational convention will be helpful: we will use dashed symbols when we are dealing with complex behaviors. Thus, α' will denote the activation of a complex behavior B', etc.

4.1 Mode regulation through complex behaviors

Now that we have simplified matters considerably, the "configuration" of the elementary level by mode assignment through complex behaviors is easily explained. Assume that there are m complex behaviors $B'_1, \ldots, B'_m$. Each of them represents a "pure" mode. Recall that the activation dynamics of each elementary behavior essentially is a sum of m terms $\mu_i T_i(...)$, where $i = 1, \ldots, m$ (plus a decay term that we do not have to consider here, see (14)). The basic idea is to have μ_i follow the activation α'_i of the complex behavior B'_i: we simply put, in each elementary behavior's activation system,

$$\mu_i(t) := \alpha'_i(t). \tag{20}$$

This is how the elementary level is "configured" by the activation of complex behaviors. Before we turn to the question of how the activation of complex behaviors is determined, some remarks should be made concerning (20).

- Simply setting μ_i equal to α'_i is possible since we use simplicistic complex behaviors without target dynamics and hence, without product term. In the general case, where behaviors have target dynamics and a product term, μ_i is controlled in a closed loop by B'_i's product term just like actuators are controled by the product term of elementary behaviors. In the general case, therefore, μ_i may well deviate from α'_i, and we have to distinguish between them. However, in our simplicistic case, we may rewrite the activation dynamics (14) of an elementary behavior, as follows:

$$\dot{\alpha} = \alpha'_1 T_1(\alpha, \ldots) + \ldots + \alpha'_m T_m(\alpha, \ldots) - \alpha k \prod_{j=1,\ldots,m} (1 - \alpha'_j)^r \tag{21}$$

- Loosely speaking, the dynamical system (21) is likely to undergo *bifurcations* when it changes from one mode to another. This is to say that it will behave qualitatively differently in different modes. However, the notion of bifurcations is not neatly defined for open systems, and should be used with caution when one is talking to a mathematician. Yet, physicists will understand.

- (21) is formally a linear combination of modes (again, disregarding the decay term that is irrelevant when the modes are relevant). However, it would be misleading to think of modes a entities that can be "linearly superimposed". The mixed mode that occurs when $\alpha'_1 = \alpha'_2 = 1$ cannot be in general understood in terms of the pure modes $\alpha'_1 = 1, \alpha'_2 = 0$ and $\alpha'_1 = 0, \alpha'_2 = 1$. In general, the two pure modes will be qualitatively different, and one cannot just "mix" qualitatively different things. One might say, modes don't mix, they get entangled. It is probably good advice for designers to prevent modes from getting entangled. This may be achieved with suitable winner-take-all mechanisms, one of which will be described below in an example. At least this is good advice when one sets out for behavior systems whose behavior one can understand (and modify and maintain and sell ...). It is unclear whether "opaque" systems, which can e.g. be obtained through evolutionary techniques rather than by explicit design, might be more effective. Be this as it may, the scheme (21) certainly admits mode mixing, so one can go for it if one wishes.

- When an elementary behavior does not participate in the complex behavior B'_i, T_i is the zero function. The term $\alpha'_i T_i(\ldots)$ can then be omitted from (21). For instance, in the VUB AI Lab scenario, an elementary behavior `sit_still` would probably participate in the complex behavior `recharge` but not in `work`. Thus, there would be no term $\alpha'_{recharge} T_{recharge}(\ldots)$ in the activation dynamics of `sit_still`.

4.2 Activation dynamics

Now we turn to the activation dynamics of a complex behavior. Like in elementary behaviors, it is a dynamical system:

$$\dot{\alpha}' = T'(\alpha', \ldots) \tag{22}$$

Again, the notorious dots "..." will suffer some explanation. They represent various kinds of time-varying input to the system (22). The iron rule stated in section 3.3 applies to complex behaviors, too. Recall that it explicitly disallows input from higher levels, and implicitly allows everything else. Thus, the dots in (22) might stand for the activation parameters $\alpha'_1, \ldots, \alpha'_m$ of (other) first-level complex behaviors, or derivatives thereof, or any kind of elementary-level quantity discussed in section 3.3, and, of course, sensor input.

As a general rule (with many exceptions), sensor input to complex behaviors should tend to come from slow internal sensors (like battery level) as opposed to fast external sensors feeding into elementary behaviors (like modulated light sensors).

For most practical purposes in Lego vehicles, the following explicit version of (22), for a complex behavior B'_i, will do:

$$\dot{\alpha}'_i = T'_i(\alpha'_1, \ldots, \alpha'_m, \mathbf{s}'_i(t)) \tag{23}$$

4.3 An example

Let's design a rudimentary activation dynamics for the two complex behaviors (and hence, two pure modes) `work` and `recharge` of our Lego vehicle. We wish them to have the following properties:

(A) There should be an adjustable winner-take-all competition between the activations α'_{work} and $\alpha'_{recharge}$. It should be possible, in the one extreme, to make them mutually exclusive ($\alpha'_{work} \sim 1$ iff $\alpha'_{recharge} \sim 0$ and vice versa); or, in the other extreme, to let the one take no notice of the other at all (such that α'_{work} and $\alpha'_{recharge}$ can assume all values between 0 and 1 independently); or anything in between.

(B) Assume that there is a battery level sensor BL that reads 1 when the battery is full, 0 when it's empty, and behaves linearly in between. Then, the tendency for being in working mode should correlate positively with BL, while the tendency for being in recharging mode should correlate negatively.

(C) We want recharging to happen opportunistically when the robot accidentally passes the charging station nearby, even when everything else being equal it should be in working mode.

(D) There should be an emergency mechanism that puts the robot into pure recharging mode when the battery level BL drops below an alarm value *alarm*.

Let us first turn to requirement (A). A transparent and simple method to come up with an adjustable winner-take-all mechanism is to model the activation dynamics of `work` and `recharge` with two variables each, i.e. using $\alpha'_{work,1}, \alpha'_{work,2}$ for `work` and $\alpha'_{recharge,1}, \alpha'_{recharge,2}$ for `recharge`. The first of these variables ($\alpha'_{work,1}$ and $\alpha'_{recharge,1}$) are the ones which eventually adjust modes in elementary behaviors; it is they that appear, for instance, in (21). The second ones in each pair ($\alpha'_{work,2}$ and $\alpha'_{recharge,2}$) are auxiliary variables that appear only inside the dynamical systems responsible for the activation of `work` and `recharge`.

The role of $\alpha'_{work,2}$ and $\alpha'_{recharge,2}$ is to collect and assemble the influences from requirements (B),(C),(D), yielding a kind of "vote" for the current appropriateness of working and recharging. These "votes" for recharging and working are then passed into the equations for $\alpha'_{work,1}$ and $\alpha'_{recharge,1}$, where a winner-take-all mechansim of adjustable efficiency determines the final arbitration between `work` and `recharge`.

The following equations yield the desired dynamics for $\alpha'_{work,1}$ and $\alpha'_{recharge,1}$:

$$\begin{aligned} \dot{\alpha}'_{work,1} &= k_1(\sigma_{0,\beta_1}(\alpha'_{work,2} - \alpha'_{recharge,2}) - \alpha'_{work,1}) \\ \dot{\alpha}'_{recharge,1} &= k_1(\sigma_{0,\beta_1}(\alpha'_{recharge,2} - \alpha'_{work,2}) - \alpha'_{recharge,1}) \end{aligned} \tag{24}$$

In these equations, k is a gain factor that adjust the overall time scale, and σ_{0,β_1} is a sigmoid function with turning point in 0 and steepness β_1. For large β_1 one gets a sharp winner-take-all characteristics.

Now let's turn to the "voting" part, i.e. to the dynamics of $\alpha'_{work,2}$ and $\alpha'_{recharge,2}$. For the complex behavior `work`, we will not invest much ingenuity and simply let its activation "vote" follow the battery level, the idea being that the propensity to work is proportional to the energy reserve. This accounts for (B) in the case of `work` and gives us

$$\dot{\alpha}'_{work,2} = k_2(BL - \alpha'_{work,2}). \tag{25}$$

Since BL changes very slowly, the time scale adjustment factor k_2 can be set quite small while still guaranteeing that $\alpha'_{work,2}$ essentially is equal to BL all the time.

In the case of `recharge`, we invest a bit more ingenuity in order to account for (B), (C), and (D).

For (C), the robot must be able to sense that the charging station is near. Assume that $CL(t)$ is a sensor quantity representing the overall sensor input from the white light sensors that respond to the light source mounted on the charging station. Thus, CL should yield a rough and highly nonlinear estimate of the robot's distance to the charging station – provided it is roughly aligned toward it. CL will thus have an appreciable size only when the robot is near the charging station and more or less directed towards it, which makes CL a good trigger for opportunistic visits at the charging station. We may further assume that CL maximally reads 1. Figure 5 gives a sketch of how CL relates to the distance from the charging station.

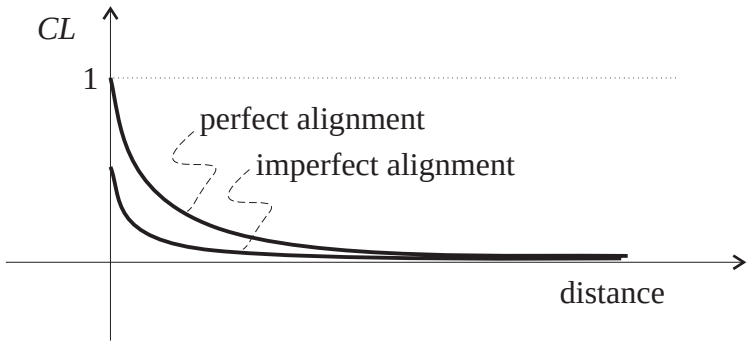

Figure 5: Distance from the charging station, as represented in the sensor quantity CL.

Now we possess all ingredients for meeting the requirements (B), (C), and (D). Each of them is accounted for by a separate additive component in the following equation:

$$\begin{aligned} \dot{\alpha}'_{recharge,2} &= k_2(1 - BL - \alpha'_{recharge,2}) && (26)\\ &\quad + k_3 CL(t)(1 - \alpha'_{recharge,2}) && (27)\\ &\quad + k_4(\sigma_{0,\beta_2}(alarm - BL) - \alpha'_{recharge,2}) && (28) \end{aligned}$$

In this equation, the first term (26) is the analogue of (25) and needs no further explication. It accounts for (B).

(27) accounts for the opportunism requirement (C): when CL becomes appreciable, indicating that the charging station is near, the term (27) pushes $\alpha'_{recharge,2}$ towards 1, overriding the slow process (26). The constant k_3 essentially determines the crucial distance (i.e. the crucial value of CL) when this happens. It is easy to further refine (27) by an additional factor that makes the crucial distance depend on the battery level (opportunistic recharging should be harder to trigger when the battery level is quite high), but this shall not concern us here.

Finally, (28) yields the alarm mechanism (D). The constant k_4 should be much greater than k_2 and other similar time factors that one might use inside the activation dynamics

of `work` in order to ensure that (28) really becomes the all-overriding influence when BL drops below *alarm*. Accordingly, β_2 should be set to yield a very steep sigmoid.

5 All parts assembled

I shall now collect everything we have got so far and present it with a bit more indices than in the preceding sections. Furthermore, I will add another level of complex behaviors, so that it becomes clearly visible how multi-level architectures can be designed.

Assume that there are l second-level complex behaviors $B''_1, \ldots, B''_l$, m first-level complex behaviors $B'_1, \ldots, B'_m$, and n elementary behaviors $B_1, \ldots, B_n$.

The elementary behavior B_j has a target dynamics as described in (4) and (5):

$$\dot{\mathbf{x}}_j = G_j(\mathbf{x}_j, \mathbf{s}_j(t), \alpha_1(t), \ldots, \alpha_n(t)) \quad (29)$$

$$\mathbf{g}_j(t) = \Gamma_j(\mathbf{x}_j) \quad (30)$$

$\mathbf{s}_j$ is sensor input made available to B_j, and $\alpha_1(t), \ldots, \alpha_n(t)$ are the activations of $B_1, \ldots, B_n$.

The product term of B_j, copied from (17), is

$$\mathbf{u}_j(t_\nu) = k\alpha_j(t_\nu)(\mathbf{g}_j(t_\nu) - \mathbf{z}_j(t_\nu)). \quad (31)$$

Finally, the activation dynamics of B_j, adapted from (21), embellished with indices:

$$\dot{\alpha_j} = \alpha'_1 T_{j,1}(\alpha_1, \ldots, \alpha_n, \mathbf{x}_j, \mathbf{s}_j) + \ldots + \alpha'_m T_{j,m}(\ldots) - \alpha_j k \prod_{i=1,\ldots,m} (1 - \alpha'_i)^{r_j} \quad (32)$$

In this equation, $\alpha_1, \ldots, \alpha_n$ are the activations of $B_1, \ldots, B_n$, $\alpha'_1, \ldots, \alpha'_m$ are the activations of $B'_1, \ldots, B'_m$, $\mathbf{x}_j$ is the "representation" of B_j's target dynamics, $\mathbf{s}_j$ is the sensor input made available to B_j, k is a constant that determines the maximal decay rate, and r_j determines how sensitively the onset of decay depends on all the α'_i being near to zero.

(32) provides a basic format for the activation dynamics of B_j. As discussed in section 3.3, the inputs to the components $T_{j,i}$ can be any quantity found or defined within the elementary level (e.g. derivatives of other quantities). The arguments listed in (32) only propose the quantities that seem to be the most standardly useful ones. Furthermore, like in the example (24)- (28), one can introduce auxiliary variables in order to endow the activation dynamics with interesting properties.

In our simplified version of the DD scheme, complex behaviors consist of nothing but an activation dynamics. For a first-level complex behavior B'_i, we get

$$\dot{\alpha}'_i = \alpha''_1 T'_{i,1}(\alpha'_1, \ldots, \alpha'_m, \mathbf{x}'_i, \mathbf{s}'_i) + \ldots + \alpha''_l T'_{i,l}(\ldots) - \alpha'_i k \prod_{h=1,\ldots,l} (1 - \alpha''_h)^{r'_i} \quad (33)$$

(33) differs from the version given in (23), since now we have to account for second-level complex behaviors that regulate modes of the first level of complex behaviors. Still, (33) should be self-explaining.

Last but not least, the activation dynamics of a second-level complex behavior B''_h has the form

$$\dot{\alpha}''_h = T''_h(\alpha''_1, \ldots, \alpha''_l, \mathbf{x}''_h, \mathbf{s}''_h) \tag{34}$$

What has been said about (32) concerning additional kinds of input and auxiliary variables applies to (33) and (34), too. The DD scheme allows that input into a higher level can come from *all* lower levels. It seems, however, wise to be restrictive. It would probably make the design intransparent if one used input from lower levels than the one immediately lower. In the present case this means that one should not use parameters from the elementary level as input for the second level.

6 Actuators

The part of the DD scheme that lies closest to actuators is the product term (17) of elementary behaviors. It is assumed that actuators are controled by a one-dimensional signal, which is additively updated at each time step by $u(t_\nu)$.

This is certainly far from being a ready-to-use actuator interface. Actuators differ widely from each other – a servo motor, an electromagnet, and a IR emitter have not much in common. Since this paper in its present version aims at readers who work with 2-df Lego robots, I will say a few words about how to interface their motors in a PDL environment, given the product terms of elementary behaviors.

Assume that there are n_0 elementary behaviors $B_1, \ldots, B_{n_0}$ that issues signals to a particular motor. This means that at each PDL working cycle t_ν, the motor quantity M of this motor (which roughly corresponds to the voltage fed to the motor) is updated according to

$$M(t_{\nu+1}) = M(t_\nu) + \sum_{i=1,\ldots,n_0} u_i(t_\nu), \tag{35}$$

where $u_i(t_\nu)$ is the vector component from the lhs. of B_i's product term which is responsible for M.

This PDL way of handling actuator quantities like M effectively endows actuators with a *state* – the expression "$M(t_\nu)+$" on the rhs. of (35) makes the actuator remember what its state was in the preceding working cycle, and the rest of the rhs. only *changes* what state there already is. This has the unwanted side effect that if the motor is in a state where M is appreciable (say $M = a > 0$), and then all behaviors $B_1, \ldots, B_{n_0}$ become inactive, M stays in this state a, which means that the motor keeps on running although no behavior is active that wants this.

A method to overcome this problem is to endow the motor with a decay mechanism similar to the one we are already familiar with. Thus, we would change (35) to something like

$$M(t_{\nu+1}) = M(t_\nu) + \sum_{i=1,\ldots,n_0} u_i(t_\nu) - M(t_\nu) \prod_{i=1,\ldots,n_0} (1-\alpha_i)^r, \tag{36}$$

which makes the motor signal M go toward zero when all relevant behaviors are inactive.

7 Translating DE's to PDL processes

The DD scheme describes robot control programs at the level of differential equations. Translating DE's into PDL processes is, however, a straightforward affair.

The DE's we are dealing with in the DD scheme have the basic form

$$\dot{x} = f(x, s(t)), \tag{37}$$

where $s(t)$ is some input (sensoric, in this case) to the system. We assume that time is measured in seconds, and that the temporal scaling of (37) yields the right dynamics with respect to seconds as units of time.

In order to derive a PDL process from (37), we proceed in two steps. First, we construct a discrete process from (37), and second, we rewrite this discrete process as a PDL process.

The basic working cycle of PDL has a duration of 1/40 seconds. Thus, the time increment for the desired discrete process is $b := 1/40$.

The simplest method for discretizing (37) is linear extrapolation (consult textbooks on numerical methods for solving DE's for more sophisticated methods). I.e., we approximate solutions of (37) by curves that are piecewise linear within intervals of length b (cf. fig. 6)

$$\begin{aligned} x(t+b) &= x(t) + b \cdot \dot{x}(t) \\ &= x(t) + b \cdot f(x(t), s(t)) \end{aligned} \tag{38}$$

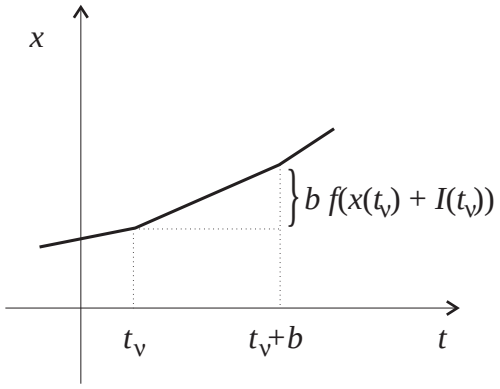

Figure 6: Discretization by linear extrapolation.

The second, and final, step is to rewrite (38) in PDL. This yields the following code:

```
...
quantity X; Sensor;
...
void DISCRETE-DE (void)
{
float s;
s = value(Sensor);
add_value(X, f(value(X), s)/40); /* b = 1/40! */
}
```

Acknowledgments I feel very grateful towards Peter Stuer and Dany Vereertbrugghen who provided the initial kick (they had to kick more often than once); towards Thomas Christaller and Luc Steels for providing the material and theoretic substrate in which this work grounds; and towards all of them and the other members of the VUB AI Lab agent's research group for their deep interest and continuated discussions.

References

[1] G.P. Baerends. On drive, conflict and instinct, and the functional organization of behavior. In M.A. Corner and D.F. Swaab, editors, *Perspectives in Brain Research. Proc. of the 9th Int. Summer School of Brain Research, Amsterdam, August 1975*, pages 427–447. Elsevier, Amsterdam, 1975.

[2] B. Blumberg. Action-selection in Hamsterdam: Lessons from ethology. In D. et al. Cliff, editor, *From Animals to Animats 3: Proc. of the 3rd Int. Conf. on Simulation of Adaptive Behavior*, pages 108–117. MIT Press (Bradford Books), 1994.

[3] H. Haken. *Advanced Synergetics - Instability Hierarchies of Self-Organizing Systems and Devices*, volume 20 of *Springer Series in Synergetics*. Springer, Berlin/Heidelberg, 1983.

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

[5] M.G. Paulin. The role of the cerebellum in motor control and perception. *Brain Behavior and Evolution*, 41:39–50, 1993.

[6] R.F. Port, F. Cummins, and J.D. McAuley. Naive time, temporal patterns and human audition. Research Report 118, Cognitive Science Program at the Indiana University, Bloomington, Indiana, 1994. To appear in van Gelder & Port (1995), Mind as Motion: Explorations in the Dynamics of Cognition, Bradford/MIT Press 1995.

[7] S.S. Robertson, A.H. Cohen, and G. Mayer-Kress. Behavioral chaos: Behind the metaphor. In L.B. Smith and E. Thelen, editors, *A Dynamic Systems Approach to Development: Applications*, pages 119–150. Bradford/MIT Press, Cambridge, Mass., 1993.

[8] G. Schöner, H. Haken, and J.A.S. Kelso. A stochastic theory of phase transitions in human hand movement. *Biological Cybernetics*, 53:247–257, 1986.

[9] L. Steels. Building agents out of autonomous behavior systems. In L. Steels and R.A. Brooks, editors, *The "Artificial Life" Route to "Artificial Intelligence": Building Situated Embodied Agents*. Lawrence Erlbaum, 1993.

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

[11] G.M. Werner. Using second order neural connections for motivation of behavioral choices. In D. et al. Cliff, editor, *From Animals to Animats 3: Proc. of the 3rd Int. Conf. on Simulation of Adaptive Behavior*, pages 154–161. MIT Press (Bradford Books), 1994.

[12] J.J. Wright and D.T.J. Liley. Dynamics of the brain at global and microscopic scales: Neural networks and the EEG. *Behavior and Brain Sciences*, page to appear, 1995.