Chapter 6

Behavioral organization

6.1 Introduction and motivation

In the previous lectures, the topic of generating individual behaviors, either by hand or using evolutionary methods, have been studied. However, in more complex robotic brains, there is more than one behavior available: The brains of such robots contain a **behavioral repertoire** from which behaviors are dynamically chosen for activation. Thus, a central issue in BBR is behavioral organization (also known as **behavioral coordination**, **behavior arbitration**, or **action selection**), i.e. the process of determining which behavior to activate at any given time. This problem is particularly relevant for **motor behaviors** involving locomotion (such as behaviors involved in navigation), but the problem is also relevant for non-motor behaviors (e.g. thought processes).

However, the approach of dividing the overall behavior of a robot into several simple behaviors is, of course, not the only possible way of generating complex behaviors. An alternative approach is to generate all behaviors simultaneously (e.g. through artificial evolution) within a single decision-making system (e.g. a neural network).

A brief discussion of this approach will now be given, followed by a more detailed description of some methods for behavioral organization.

6.2 Complex behaviors without explicit arbitration

As pointed out in [11] and [45], a common problem when evolving complex behaviors is that the difference between the initial, randomly generated, robotic brains and those capable of carrying out the desired complex behavior, is so large that the EA without exception finds itself stuck in a local optimum. One obvious solution is to break down the problem into simpler tasks, for which the EA *will* be able to find an adequate solution, and then either to fuse several

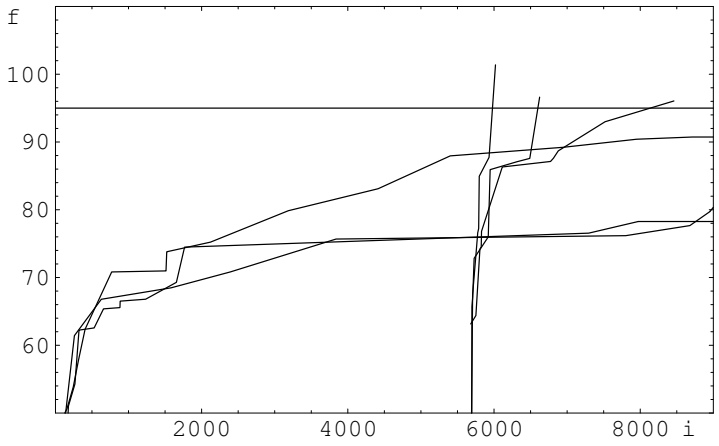

Figure 6.1: *Maximum fitness as a function of the number of evaluated individuals during the evolution of a complex behavior, divided into two constituent behaviors [45]. The three short and almost vertical curves show the results from runs in which the constituent behaviors were first evolved separately, and then combined to form the complete robotic brain.*

evolved robotic brains into one complete brain [45], or to make the problem progressively more difficult until the desired level of complexity is reached, a procedure called **incremental evolution** [11].

An example, taken from [45], is shown in Fig. 6.1. Here, the task was to evolve a complex robotic brain, capable both of cleaning an arena *and* avoiding moving obstacles. Two different approaches were used, namely (1) attempting to evolve the complex behavior directly, starting from random robotic brains (which, in this investigation, were represented by sequences of `IF-THEN-ELSE`-rules, as described in Fig. 4.3 in Chapter 4), and (2) first evolving *cleaning* and *evasion* separately, then fusing the two generalized FSMs by adding random connections between them and, finally, continuing the evolution until the complex behavior emerged. In Fig. 6.1, the results of the investigation are shown. The three lower curves, extending over the whole interval, show the maximum fitness as a function of the number of evaluated individuals, whereas the three short, almost vertical curves show the results obtained when first evolving the **constituent behaviors** separately. In the latter case, the two curves have been shifted to right by an amount that corresponds to the number of evaluated individuals needed to reach acceptable performance of the two constituent behaviors. As is evident from the figure, the approach of first evolving constituent behaviors separately is, by far, the most successful one, quite rapidly reaching the empirically determined fitness threshold for acceptable performance of the complex behavior (indicated by the horizontal line). By contrast, the attempts to evolve the complex behavior directly were not successful: In no case was the fitness threshold reached.

A similar conclusion was reached by Gomez and Miikkulainen [11], who studied the evolution of complex *prey capture* behavior in contests of pursuit and evasion. In their study, a simulated robot was required to capture a prey, moving with speed s, which was given a head start of n moves, a task denoted E_n^s. It was found that the complex behavior $E_4^{1.0}$ could not be generated by direct evolution. However, using several intermediate steps, first evolving $E_0^{0.0}$, then $E_2^{0.0}$ etc., the $E_4^{1.0}$ behavior could be achieved easily.

Thus, the conclusion reached in [11] and [45] is that a complex behavior can be evolved incrementally. However, there are other ways of achieving complex behaviors: Floreano and Mondada [10], by contrast, point out that complex behavior can, at least in certain situations, be achieved simply by increasing the complexity of the environment, and *decreasing* the complexity of the fitness function, in order to avoid canalizing the evolving robotic brains toward suboptimal solutions. In [10], the task of navigation described in Chapter 4 was extended also to include the limited energy storage capacity of the battery of the robot. Thus, the behaviors of *homing navigation* (toward a light source close to the charging station) and *battery charging* had to be included in the brain of the robot. Rather than using incremental evolution, Floreano and Mondada simplified their fitness measure (see Chapter 4), removing the requirement of straight-line motion, and allowed the evolving individuals to extend their life by periodically returning to the charging station, in which case their simulated batteries were instantaneously recharged. Indeed, during evolution, individuals were found that could return to the charging station, thereby extending their life, even though no fitness was directly associated with doing so. In fact, the ideas presented in [10] bear some resemblance to the dichotomy of task behaviors and auxiliary behaviors, discussed in connection with the utility function method described below.

It should be noted, however, that the task used in [10] was quite simplified (e.g. by allowing instantaneous recharging), and that the authors did not attempt to compare with incremental evolution through which, perhaps, the complex behavior could have evolved even faster. Nevertheless, the results reported above show that there are several ways of achieving complex behaviors.

In addition, Floreano and Mondada carried out a painstaking analysis of their evolved neural network, in order to determine how it generated the complex behavior. While their analysis was impressive, it does, perhaps, point to a weakness with respect to the use of neural networks. The advantages of neural networks, i.e. noise tolerance, absence of bias introduced by the user etc., have been eloquently presented by several authors (e.g. [5], [10], and [30]). However, neural networks are also notoriously difficult to interpret, and a complex behavior based on neural networks will therefore, most often, have to be used as a black box.

Furthermore, the issue of *scaling* presents difficulties for the approach de-

scribed in this section: The studies described above involved very few behaviors, and quite simple situations in which, for example, a sequence of tasks of increasing complexity could be defined quite easily. In cases involving more behaviors, the alternative approach of using a repertoire of distinct behaviors and a system for organization is preferable, and the remainder of this chapter will be devoted to such approaches.

6.3 Methods for behavioral organization

A researcher aiming to generate a method for behavioral organization has a considerable amount of freedom and, not surprisingly, a very large number of such methods have appeared in the literature, such as e.g. the pioneering **subsumption method** which was developed by Rodney Brooks [3] and marked the starting point for research in behavior-based robotics.

Unlike, say, an physicist or a biologist, a robotics researcher is not constrained by observations of nature to the same extent. Clearly, even a robot must obey the laws of physics, and biology is often used as a source of inspiration, but the details of a method for behavioral selection can, basically, be devised in an infinite number of ways: Anything that works is an acceptable method for generating a robotic brain. At a first glance, this might seem as an advantage, but, in view of the very many methods that have appeared in the literature, and the considerable difficulty in comparing them (resulting in confusion), one might argue that it is, in fact, a strong *disadvantage*.

In any case, the many methods for behavioral organization that have been proposed can be divided into two main categories, which will now be described.

6.3.1 Taxonomy of methods for behavioral organization

The purpose of any method for behavioral organization is to determine when different behaviors should be activated, and it is therefore natural to categorize methods of behavioral organizations based on the procedure they use for selecting behaviors. There are two main categories, namely **arbitration methods** and **cooperative methods**[1]. In arbitration methods, exactly one behavior is active, and the selection of which behavior to activate is generally a function both of present sensor readings and the internal state of the robot. In cooperative methods, the action taken by the robot is a weighted combination of the actions suggested by several behaviors. Behavioral organization methods can be further divided into subcategories belonging to either of the two main categories just introduced. However, such a detailed classification is of dubious

[1]Cooperative methods are sometimes also called **command fusion methods.**

value, and will not be considered here. Instead, two examples of behavioral coordination methods, one from each category, will be given.

6.3.2 The utility function method

The **utility function method** [43] is a biologically inspired arbitration method. As its name implies, in this method, the selection of behaviors is based on the concept of utility described in Chapter 5. Utility functions as a common currency, used for guiding the selection of behaviors. As an example, consider a floor-sweeping robot, which is given a fitness increment for each square meter of floor it sweeps. Clearly, the robot should try to sweep as much floor as possible, in order to maximize its fitness. However, if the robot runs out of battery energy, it will no longer be able to move. Thus, the utility of a behavior that forces the robot to suspend, temporarily, its floor-sweeping activities to charge its batteries will rise as the energy in the battery decreases, even though the battery charging behavior does not lead to any *direct* increase in fitness. Hence, in order to receive as much fitness as possible over a long period of time, the robot must, in fact, maximize *utility*, which, as was shown in Chapter 5, is exactly the strategy employed by rational agents.

Utility also provides a means of allocating limited resources in an optimal way. The life of any animal (or robot) inevitably involves many trade-offs, where less relevant behaviors must be sacrificed or at least postponed in order to perform the most relevant behaviors, i.e. those associated with largest utility value.

Organizing behaviors using the utility function method

In the utility function method [43], the robot is equipped with a behavioral repertoire, in which each behavior B_i is assigned a utility function U_i, which depends on the values of the **state variables** of the robot. Three kinds of state variables are defined, namely **external variables**, denoted s, representing e.g. sensor readings, **internal physical variables**, denoted p, representing e.g. the energy in the batteries of the robot, and **internal abstract variables**, denoted x, which are dimensionless variables used in the behavioral selection procedure, and which roughly correspond to signalling substances (e.g. hormones) in biological systems. Thus, the most general form of a utility function is

$$U_i = U_i\left(s_1, \ldots, s_{n_\mathrm{e}}, p_1, \ldots, p_{n_\mathrm{p}}, x_1, \ldots, x_{n_\mathrm{i}}\right), \tag{6.1}$$

where n_e, n_p, and n_i denote the number of external, internal physical, and internal abstract variables, respectively. However, in most cases, each utility function will only depend on a subset of the set of available variables.

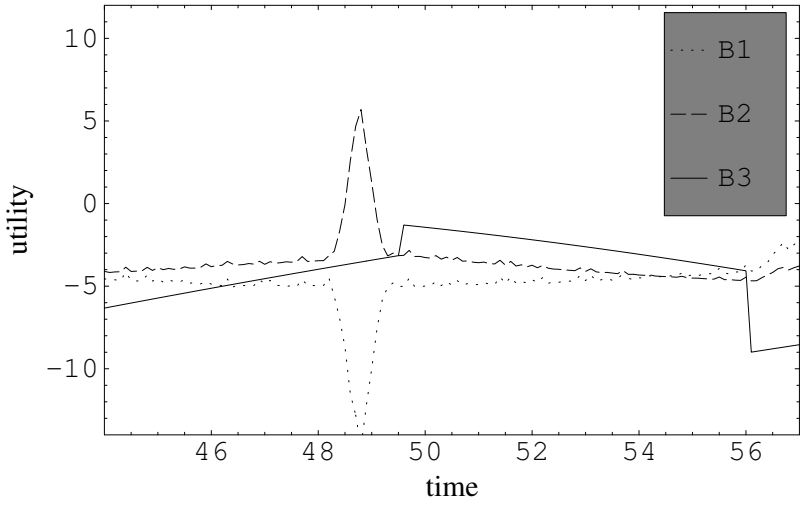

Figure 6.2: *The utility functions for a behavioral organization problem, shown as functions of time during part of the evaluation of a simulated robot.*

In the utility function method, behaviors are divided into two categories. **Task behaviors** are directly related to the task of the robot and increase its fitness, if successfully executed. An example of a task behavior is the floor-sweeping behavior described in the example above. **Auxiliary behaviors**, on the other hand, do *not* change the fitness of the robot, but are nevertheless necessary for the robot to be able to perform its duties. The battery charging behavior (for the floor-sweeping robot described above) is an example of an auxiliary behavior.

Furthermore, a time variable t_i, referred to as the **behavior time** of behavior B_i, is defined for all behaviors. t_i is set to zero every time B_i is activated, and then follows the same rate of increase as the global time variable t, which measures the time since the robot was initialized. As soon as behavior B_i becomes inactive, t_i is again set to zero, where it remains until the behavior is activated again.

Behavioral selection is straightforward in the utility function method; At regular intervals, the values of the utility functions are computed, using the latest available readings of the state variables as input, and the behavior with the highest utility value is executed. This is illustrated in Fig. 6.2, in which is shown the variation in time of three utility functions obtained for a behavioral organization problem involving three behaviors. In the part of the simulation shown in the figure, behavior B_2 is initially active, since its utility value exceeds that of the other behaviors, and remains active from $t = 44$ (arbitrary units) until around $t = 49.5$, when B_3, which in this case happened to be an auxiliary battery charging behavior, was activated. Around $t = 56$, B_1 was

activated.

The problem, of course, is to determine the shape of the utility functions. In the utility function (UF) method, the optimization of utility functions is normally carried out using EAs. In general, the utility functions depend on several state variables, and should provide appropriate utility values for any combination of the relevant inputs. Thus, determining the exact shape of the utility functions is a formidable task, and one for which EAs are very well suited. In principle, GP can be used, in which case any function of the state variables can be evolved. However, it is often sufficient to make an ansatz for the functional form of each utility function, and then implement the EA as a standard GA for the optimization of the parameters in the utility function.

For example, for a utility function U_i (associated with a behavior B_i) that depends on the two variables s and p, a common ansatz is

$$U_i(s,p) = a_{i,00} + a_{i,10}s + a_{i,01}p + a_{i,20}s^2 + a_{i,11}sp + a_{i,02}p^2, \tag{6.2}$$

where the $a_{i,jk}$ are constants. When optimizing the utility functions the $a_{i,jk}$ (and the corresponding constants in all other utility functions), should be encoded in the chromosomes used by the GA. The fitness gained while executing the task behavior(s) is used as the optimization criterion.

A problem that may occur when selecting behaviors based on utility functions is **rapid behavior switching** in which the robot keeps swapping back and forth between two (or more) behaviors, and thus failing to achieve its goals. One of the purposes of the internal abstract variables is to prevent such rapid switching. Thus, an internal abstract variable x_i, which takes non-zero values only in a specific behavior B_i, can be introduced. When the robot switches from another behavior to B_i, x_i is immediately set to a non-zero value (e.g. 1), and if the utility functions are properly chosen, the utility of B_i will then be raised sufficiently to avoid immediate switching to another behavior. As an example consider, in Fig. 6.2, the jump in utility for B_3 at the moment when it becomes active. The internal abstract variables may depend both on other state variables and on the behavior time t_i, and the exact variation can be optimized by the GA as well. For example, a possible ansatz for an internal abstract variable x_i is

$$x_i = \begin{cases} b_{i,1} + b_{i,2}\mathrm{e}^{-|b_{i,3}|t_i} & \text{If } B_i \text{ is active} \\ 0 & \text{Otherwise} \end{cases} \tag{6.3}$$

where the $b_{i,j}$ are constants. Thus, with this ansatz for the internal abstract variables, the chromosomes used by the GA will encode not only the constants $a_{i,jk}$, but the constants $b_{i,j}$ as well.

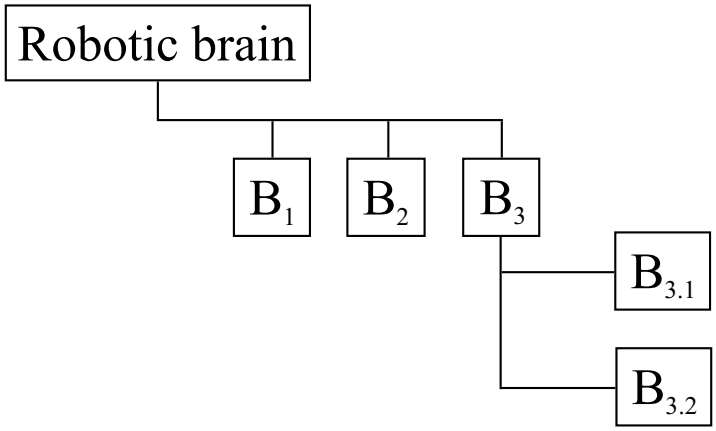

Figure 6.3: *An example of a robotic brain involving five behaviors on two hierarchical levels, taken from [44].*

Behavioral hierarchies

The presentation above is somewhat simplified, in that it does not consider the fact that, in order to keep the level of complexity of the constituent behaviors in a robotic brain at a reasonable level, the behaviors should often be divided into sub-behaviors. Battery energy maintenance is a case in point: maintaining the energy in the batteries of a robot requires that (at least) two separate procedures be executed: one for finding an energy source, and one for connecting to the energy source and carrying out the actual charging. Evidently, the entire energy maintenance sequence *could* be considered as one single behavior. However, generating such a behavior, and making it reliable, for example in the case of sudden interruptions due to obstacles in the path of the robot, would indeed be a daunting task. An alternative procedure is to introduce one behavior for locating an energy source, and one for battery charging, and to consider these behaviors as sub-behaviors to an overall energy maintenance behavior. Thus, in such a case, the robotic brain would have a hierarchical structure, an example of which is shown in Fig. 6.3.

This figure shows the structure of the robotic brain for a simple, two-wheeled guard robot considered in [44]. The behavioral repertoire consisted of a total of five behaviors, namely *straight-line navigation* (B_1), *obstacle avoidance* (B_2), *energy maintenance* (B_3), *corner seeking* $(B_{3.1})$, and *battery charging* $(B_{3.2})$. In the arena where the robot operated, the battery charging stations were placed in the corners. Thus, the *corner seeking* behavior corresponded to charging station localization.

As shown in the figure, the five behaviors were placed on two hierarchical levels. The utility function is able to cope with multiple hierarchical levels of behaviors, by comparing utility values on a level-by-level basis. Thus, in the

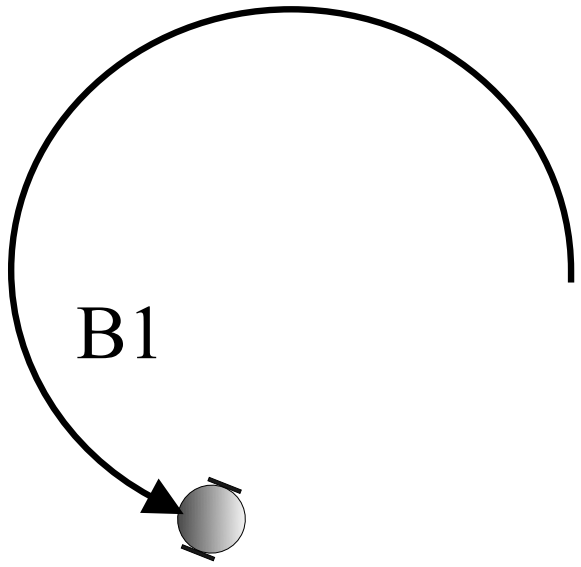

Figure 6.4: *The motion of the robot (seen from above) in Example 1 while executing* B_1

case shown in Fig. 6.3, it was determined which of the three utility functions $U_1, U_2,$ and U_3 took the highest value. If it happened to be U_3, the comparison of $U_{3.1}$ and $U_{3.2}$ would determine which of these two behaviors was active.

The procedure is actually slightly more complex than this since the switch from, say, B_2 to B_3 (and then to one of the sub-behaviors $B_{3.1}$ or $B_{3.2}$) may require modification of the internal abstract variables (if any) in B_3, in order to avoid rapid behavior switching as discussed above. In practice, this is accomplished by executing an *enter procedure* on each hierarchical level. Thus, when U_3 exceeded U_2, the enter procedure for U_3 was executed (thus possibly further raising U_3 to prevent rapid switching back to B_2), Next, the enter procedure of either $B_{3.1}$ or $B_{3.2}$ was called, depending on their relative utility values, and the robot then proceeded by executing the active behavior, i.e. either $B_{3.1}$ or $B_{3.2}$.

Example 1: Circular navigation

Consider the very simple example of a two-wheeled, differentially steered robot (e.g. a guard robot), whose task it is to move as far as possible along a circular path. The robot, which is illustrated in Fig. 6.4, is equipped with two behaviors, namely *circular navigation* (B_1) and *battery charging* (B_2).

In the first behavior, the robot sets the torque of its two motor to slightly different values, thus executing a circular motion, as illustrated in Fig. 6.4. In B_2, the robot simply sets the torque of the two motors to zero, eventually reaching a standstill. It is assumed that the charging of the batteries (taking

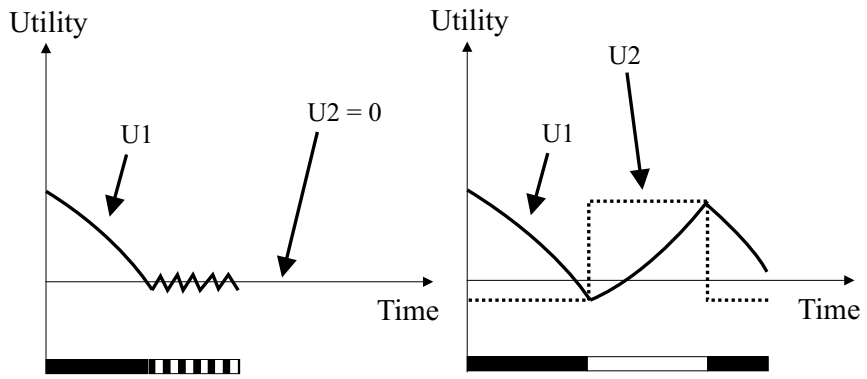

Figure 6.5: *Utility variation and behavior activation. In the left panel U_2 is identically zero, resulting in behavioral dithering (as indicated by the rectangles at the bottom: A black rectangle indicates that B_1 is active and a white one that B_2 is active). By constrast, in the right panel, U_2 depends on an internal abstract variable x.*

place e.g. via a conducting floor) starts after a time t_c after the motor torques have been set to zero, and then continues until B_1 is activated. If the batteries become full, no further charging occurs, but the robot does not start to move until B_1 is activated again.

The evaluation of the robot continues during a time interval of length T, unless the robot runs out of battery energy, in which case the evaluation is terminated immediately. For this robot, B_1 is the task behavior, for which a fitness measure should be specified. In this simple case, the distance moved while executing B_1 is a suitable fitness measure. However, since the robot consumes energy while executing B_1 it must, from time to time, also activate B_2. In order to do so, it must be able to determine the utility values of B_1 and of B_2, at all times. Clearly, in this simple case, the battery energy E is a very important variable. Thus, for B_1, the following ansatz can be made for the utility function: (making the somewhat arbitrary choice $p = 2$ for the polynomial degree)

$$U_1(E) = a_{00} + a_{10}E + a_{20}E^2. \tag{6.4}$$

Now, since the selection of a behavior (for activation) depends only on the relative utility values of the available behaviors (i.e. the behavior with the highest utility value is activated), one might be tempted to set $U_2 \equiv 0$ in this case. However, this would lead to a more subtle problem: Assume that the coefficients determining U_1 have been set such that U_1 decreases as the battery energy falls. If U_2 were identically zero, B_2 would become active as soon as U_1 dropped below zero, provided that the coefficients were such that it does.

To be specific, consider the case in which the battery energy is normalized to the range $[0, 1]$, and where U_1 is given as $U_1 = -0.5 + E^2$, so that B_2 would become active at $E = 0.707\ldots$. As soon as B_2 becomes active, the battery

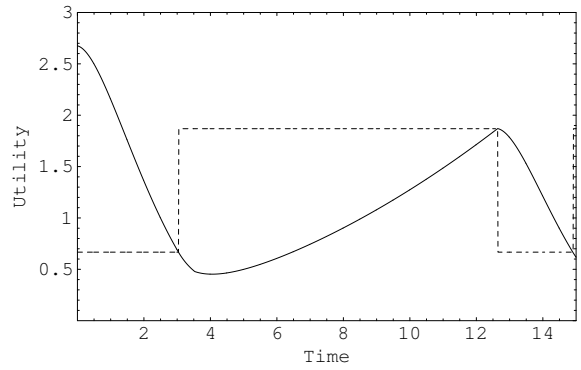

Figure 6.6: *The actual utility functions found by EA for the circular navigation example. The solid line shows U_1 and the dashed line represents U_2.*

energy would rise, resulting in U_1 becoming positive, and thus activating B_1. However, at this point, E would again fall, thus directly making $U_2 > U_1$, again activating B_2 etc. Put differently, the robot would find itself dithering endlessly between B_1 and B_2. This situation is illustrated in the left panel of Fig. 6.5. Clearly, a different ansatz is needed for U_2, and this is where the concept of internal abstract variables becomes useful. Consider a variable x that takes the value 1 whenever B_2 is active and the value 0 whenever B_1 is active[2]. Now, an ansatz of the form

$$U_2(x) = b_{00} + b_{10}x + b_{20}x^2, \tag{6.5}$$

can be made[3] for U_2. Now, as U_2 becomes larger than U_1, as a result of a drop in battery energy, x (and, therefore, U_2) will make an instantaneous jump. Provided that the coefficients in Eq. (6.5) have been set appropriately, this would make U_2 much larger than U_1, thus keeping B_2 active for some time, as illustrated in the right panel of Fig. 6.5. In this simple problem, it would be possible to set correct values for the parameters (a_{ij}, b_{ij}) by hand. Nevertheless, an EA was applied to this problem. Appropriate values for the six parameters were quickly found, and the resulting *evolved* utility functions are illustrated in Fig. 6.6.

Example 2: A simple exploration robot

As a second example of an application of the utility function method, a simple exploration robot will be considered. The two-wheeled robot, shown in

[2]The variation of x could be more complex. However, this simple specification will suffice for the present example.

[3]Clearly, the ansatz $U_2 = b_{10}x$ would be sufficient in this case. However the form in Eq. (6.5) was chosen for generality: It would be applicable even in cases with more complex variation of x.

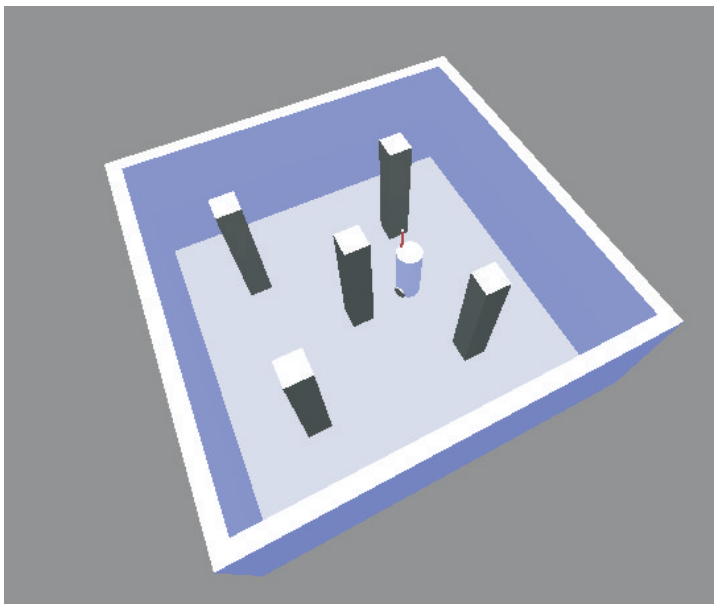

Figure 6.7: *A two-wheeled exploration robot in a simple arena with 5 stationary obstacles.*

Fig. 6.7, is differentially steered, and equipped with a two-dimensional laser range finder for measuring the distance to the nearest objects in a certain sector. In this example, the sector of measurement will be assumed to cover a narrow range centered around the current direction of motion of the robot. The task of this robot will be to explore a given arena, also shown in Fig. 6.7, while avoiding collisions with stationary obstacles.

Obviously, in a more realistic application, the robot would have to be able to find a charging station and charge its batteries when necessary, avoid moving obstacles (in addition to stationary ones), and carry out purposeful motion rather than the more or less random wandering that will be considered here. In such cases, a rather complex robotic brain involving many behaviors, probably distributed on various hierarchical levels, e.g. as illustrated in Fig. 6.3, would be needed. The utility function method would easily handle such a situation, but, for simplicity and clarity, a greatly simplified situation will be studied in this example, where the capacity of the robot's battery is assumed to be infinite. In fact, only two behaviors will be included in the robotic brain, namely *straight-line navigation* (B_1) and *obstacle avoidance* (B_2). In B_1, the brain of the robot sends equal signals to the two motors of the robot, making it move in a straight line (after an initial transient, in case the robot was turning at the time of activation of B_1). In B_2, equal signals, but of opposite sign, will be sent to the two motors, until the minimum distance (as measured by the laser range finder) to the nearest obstacle exceeds a certain minimum value.

The fitness measure is basically taken as the amount of time spent execut-

ing B_1. However, the robot receives a fitness increase *only* if it executes B_1 continuously for at least T_0 seconds. The value of T_0 was chosen, somewhat arbitrarily, as 0.2 s. With this fitness measure, the robot has a strong incentive for executing B_1 as much as possible, without dithering between the two behaviors. On the other hand, it is also necessary for it sometimes to activate B_2, in order to avoid collisions, since the evaluation is terminated if the robot collides with a wall or with some other obstacle. Thus, the robot is faced with a behavioral selection problem, involving a trade-off between carrying out the assigned task (by executing B_1) and surviving (by executing B_2). Clearly, in this simple case, it would not have been very difficult to write down a procedure for behavioral selection by hand. However, as indicated above, this simple example of a behavioral selection problem was chosen primarily for its (intended) pedagogical value, not for its (rather limited) usefulness.

In order to apply the utility function method, an ansatz is needed for each utility function. Here, U_1 will be taken as a p^{th} degree polynomial in the variables s_1, s_2, and s_3 which represent the (inverted) readings along three rays of the laser range finder (in the directions 0, $\pm$ 30 degrees, relative to the forward direction). The inverted reading y along a ray equals $R - z$, where R is the maximum range (4.0 m, in this case) of the laser range finder, and z is the measured distance to the nearest object along the ray. Thus, y will be in the range $[0, R]$. For B_2, the utility function (U_2) depends on two variables: s_{avg}, which equals the average reading of all rays of the laser range finder and x, which is an internal abstract variable. x can be interpreted as a hormone whose level is raised if the robot senses fear, e.g. as a result of an imminent collision. The variation of the variable x is slightly simplified in this example, however: it takes the value 0 if B_1 is active, and the value 1 if B_2 is active.

Thus, the utility functions U_1 and U_2 will be polynomials of three and two variables, respectively, i.e.

$$U_1(s_1, s_2, s_3) = a_{000} + a_{100}s_1 + a_{010}s_2 + a_{001}s_3 + a_{110}s_1s_2 + \ldots, \qquad (6.6)$$

and

$$U_2(s_{\text{avg}}, x) = a_{00} + a_{10}s_{\text{avg}} + a_{01}x + a_{20}s_{\text{avg}}^2 + a_{11}s_{\text{avg}}x + a_{02}x^2 + \ldots \qquad (6.7)$$

Thus, the task of the evolutionary algorithm will be to determine the coefficents a_{ijk} and a_{ij} so as to maximize the fitness of the robot in evaluations of a given maximum duration T (here set to 50 s).

The two behaviors B_1 and B_2 were implemented in the format defined by the `UF Library` simulation package, which implements the utility function method [42].

Next, an executable application was built, and the simulations for this exploration robot were carried out, using utility function polynomials of degree $p = 2$. Due to the simplicity of the problem, the evolutionary algorithm rather

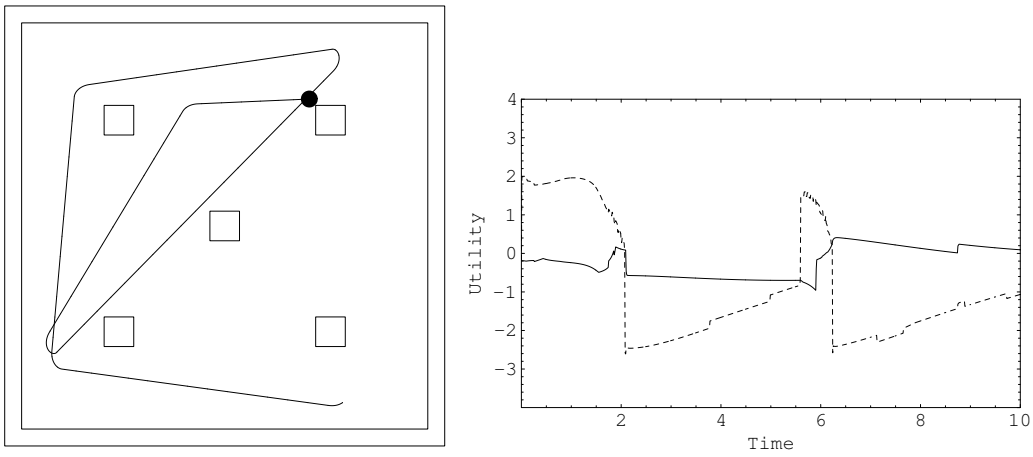

Figure 6.8: *Left panel: The motion of the robot, whose initial position is marked by a black disk. Right panel: The variation of the utility functions for the first 10 seconds of the motion. The solid line shows U_1, and the dashed line U_2.*

quickly found robotic brains that were able to keep the robot from colliding with obstacles for the full duration of an evaluation, while spending as much time as possible executing B_1. In the left panel of Fig. 6.8, the motion of an evolved individual is shown, seen from above. The right panel of the same figure shows the variation of the utility functions for the first 10 seconds of the evaluation. At the initial moment of the evaluation, the robot was placed right in front of an obstacle, forcing it first to activate B_2 in order to rotate to a different direction (this part of the motion is not visible in the left panel of Fig. 6.8, which only displays translational motion). Once a clear path was found, at $t \approx 2$ seconds, the robot began executing B_1. At $t \approx 5.5$ seconds, another obstacle was encountered, and the robot again activated B_2 for a fraction of a second, before resuming the execution of B_1.

Example 3: An analytically solvable problem

In simple cases, the problem of finding utility functions can, in fact, be solved analytically. As an example, consider a greatly simplified situation in which two behaviors are to be combined into a functioning robotic brain. The first behavior (hereafter: B_1) is a task behavior, e.g. floor-sweeping, whereas the second behavior (B_2) is responsible for re-charging the batteries of the robot. The goal for the robot is to work as much as possible with its task, i.e. to gain as much fitness per unit time as possible, without running out of energy.

In this simple example, the first behavior is specified through the fitness

function

$$f_1(t_1) = \frac{t_1}{1 + \left(\frac{t_1 - a}{b}\right)^2}, \tag{6.8}$$

where t_1 is the behavior time for B_1, and a and b are constants. It is common in any task behavior that some time elapses before the robot is able to do something useful, and that the benefit of carrying out the behavior also exhibits diminishing return after some time. Both features are captured by the fitness function defined in Eq. (6.8). The exact shape of $f_1(t_1)$ was chosen simply because it is easy to integrate, which is needed for this analytical example but otherwise, of course, unnecessary. For B_2, which is an auxiliary behavior, the fitness function $f_2(t_2)$ is set identical to 0.

As for the state variables, an internal physical variable $p_1 = E$, measuring the (normalized) energy stored in the batteries, is introduced. In B_1, E varies as

$$\frac{\mathrm{dE}}{\mathrm{dt_1}} = -c < 0, \tag{6.9}$$

where c is a constant, whereas in B_2, E varies as

$$\frac{\mathrm{dE}}{\mathrm{dt_2}} = \alpha E \text{ if } E < 1, \text{ 0 otherwise}, \tag{6.10}$$

where α is a constant. Thus, it is assumed that it is more difficult to recharge the battery the emptier it is.

The second state variable introduced is an internal abstract variable, denoted x. This variable is identically zero in B_1; By contrast, when B_2 is initiated, x is set to 1, and it then falls off exponentially with time, i.e.

$$x = 1 \text{ at } t_2 = 0, \dot{x} = -\beta x \text{ for } t_2 > 0, \tag{6.11}$$

where β is a constant. This choice of variation with time for x is somewhat arbitrary, but will be sufficient for this simple example. Proceeding to the utility functions, a very simple ansatz is used:

$$U_1 \equiv 0, \tag{6.12}$$

$$U_2(E, x) = (1 - x)(\epsilon_1 - E) + x(\epsilon_2 - E), \tag{6.13}$$

where ϵ_1 and ϵ_2 are two constants to be determined ($\epsilon_2 > \epsilon_1$).

Of course, a genetic algorithm could, in principle, be used for determining the optimal values of ϵ_1 and ϵ_2. However, if it is assumed (as we will do) that $\beta = \alpha \equiv k$, the present example is actually analytically tractable.

It will be assumed that the robot starts its motion in B_1, with energy $E = \epsilon_1$. Now, as $\mathrm{d}E/\mathrm{d}t_1 < 0$ in B_1, E will dip below ϵ_1 instantaneously, thus making

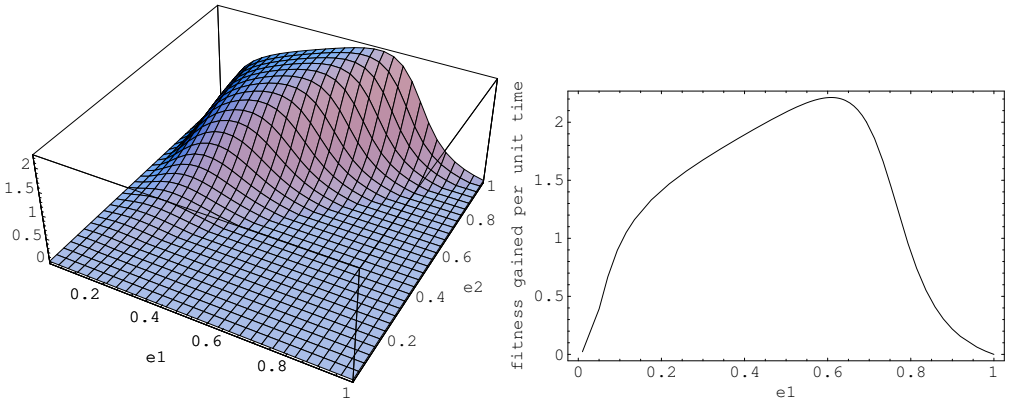

Figure 6.9: *Left panel: the fitness gained per unit time as a function of the two optimization parameters ϵ_1 and ϵ_2 in a case with $a = 10.0$, $b = 3.0$, $c = 0.02$, and $k = 0.03$. Right panel: A slice through the surface in the left panel, for $\epsilon_2 = 1$.*

$U_2 > 0$. The robot then starts executing B_2, setting x to 1. The variation of both x and E can easily be determined as

$$x = \mathrm{e}^{-kt_2}, \tag{6.14}$$

and

$$E = \epsilon_1 \mathrm{e}^{kt_2}, \text{ if} E < 1. \tag{6.15}$$

Thus, inserting Eqs. (6.14) and (6.15) in Eq. (6.13),

$$U_2(t_2) = \left(1 - \mathrm{e}^{-kt_2}\right) \epsilon_1 \left(1 - \mathrm{e}^{kt_2}\right) + \mathrm{e}^{-kt_2} \left(\epsilon_2 - \epsilon_1 \mathrm{e}^{kt_2}\right). \tag{6.16}$$

The switch to B_1, at which point x is set to zero, occurs at time $t_{2\to1}$, when U_2 dips below zero. Using Eq. (6.16),

$$t_{2\to1} = \frac{1}{k} \ln \frac{\epsilon_1 + \sqrt{4\epsilon_1\epsilon_2 - 3\epsilon_1^2}}{2\epsilon_1}. \tag{6.17}$$

At this point, the energy stored in the batteries of the robot equals

$$E_0 = \epsilon_1 \mathrm{e}^{kt_{2\to1}} = \frac{\epsilon_1 + \sqrt{4\epsilon_1\epsilon_2 - 3\epsilon_1^2}}{2}. \tag{6.18}$$

From Eq.(6.9), in B_1, the energy in the battery at $t = t_1$ is given by

$$E = E_0 - ct_1. \tag{6.19}$$

As noted above, the switch to B_2 occurs at $E = \epsilon_1$. Using Eq. (6.19) it is seen that this level is reached at

$$t_{1\to2} = \frac{\sqrt{4\epsilon_1\epsilon_2 - 3\epsilon_1^2} - \epsilon_1}{2c}. \tag{6.20}$$

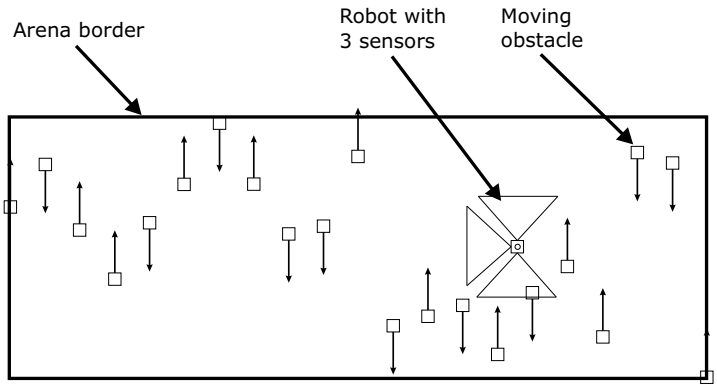

Figure 6.10: *The arena used in [33] (Example 4). The aim of the simulated robot is to move without collisions, from right to left, in the arena.*

The increase in fitness during this time is given by

$$\begin{aligned}\Delta f &= \int_0^{t_{1\to 2}} f_1(t_1)\mathrm{d}t_1 = \int_0^{t_{1\to 2}} \frac{t_1 \mathrm{d}t_1}{1+\left(\frac{t_1-a}{b}\right)^2} = \\ &= ab\left(\arctan\frac{t_{1\to 2}-a}{b} + \arctan\frac{a}{b}\right) + \frac{1}{2}b^2 \ln\frac{(t_{1\to 2}-a)^2+b^2}{a^2+b^2}. \quad (6.21)\end{aligned}$$

This is also the total fitness change during a complete cycle (i.e. through B_2 and B_1), since no fitness variation occurs during the time when B_2 is active. The total fitness gained per unit time is thus given by

$$\frac{\Delta f}{\Delta t} = \frac{\Delta f}{t_{2\to 1} + t_{1\to 2}}, \quad (6.22)$$

where the expressions for Δf, $t_{2\to 1}$, and $t_{1\to 2}$ are given by Eqs. (6.21), (6.17), and (6.20), respectively. Once the parameters a, b, c, and k have been specified, $\frac{\Delta f}{\Delta t}$ is a function of the two optimization parameters ϵ_1 and ϵ_2 only. The resulting surface for $a = 10.0$, $b = 3.0$, $c = 0.02$, and $k = 0.03$ is shown in the left panel of Fig. 6.9. The maximum occurs at $\epsilon_1 = 0.608$, $\epsilon_2 = 1$. A slice through the surface, for $\epsilon_2 = 1$, is shown in the right panel of the same figure.

Example 4: Locomotion in an arena with moving obstacles

In [33], the utility function method was used for generating a behavioral organization system for a hopping robot traversing an arena with moving obstacles, illustrated in Fig. 6.10. The simulated robot, consisting of a foot plate and a leg with two **degrees of freedom** (DOF), implemented as two revolute joints, was equipped with four behaviors: *move forward* (B1), *move backward* (B2), *stop* (B3), and *charge batteries* (B4). The robot was simulated using full newtonian

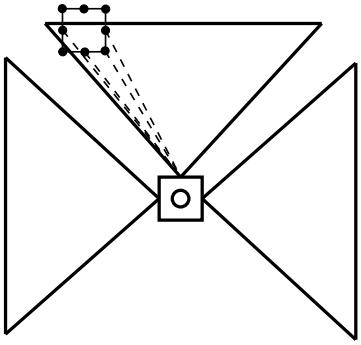

Figure 6.11: *The robot used in [33] (Example 4) as seen from above, equipped with three simple proximity sensors, each with a triangular-shaped detection range. The filled black circles illustrate points on an obstacle that are detectable by the range sensors. Dashed lines indicate detected points in this particular configuration.*

dynamics, implemented in the EvoDyn package developed by Pettersson [32]. In B4, the robot was required to remain at a standstill, charging its batteries using simulated solar cells.

While the UF method itself uses an EA to generate behavioral organization, the constituent behaviors can be generated by any method. In the particular case considered in [33], the behaviors were implemented as continuous-time, recurrent neural networks, and were optimized using an EA. Note, however, that the EA optimizing the individual behaviors should not be confused with the EA used for generating the behavioral organizer through the UF method.

Once the four constituent behaviors had been generated, the behavioral organizer, assigned the task of selecting between behaviors B1-B4, was generated using the utility function method. The simulated robot was equipped with three proximity sensors, as illustrated in Fig. 6.11. In addition, the robot was able to measure the amount of available energy in its batteries. Thus, the ansatz for each utility function was taken as a second-degree polynomial $U_i = U_i(s_1, s_2, s_3, E, x_i)$, where s_1, s_2, and s_3 are the readings of the three sensors, E is the battery energy, and x_i is a behavior-specific internal abstract variable, whose variation was modelled as in Eq. (6.3). The total number of parameters to be optimized by the GA was 96.

The settings of the simulated robot's battery were such that frequent recharging was necessary. Thus, the task of moving through the arena was strongly non-trivial, requiring constant vigilance in order to avoid collisions or an empty battery (in which cases the evaluation was terminated). The fitness measure was simply taken as the distance moved in the forward direction.

Despite the many parameters determining the utility functions and the abstract internal variables, a successful behavioral organizer was found quite quickly. An example is shown in Fig. 6.12.

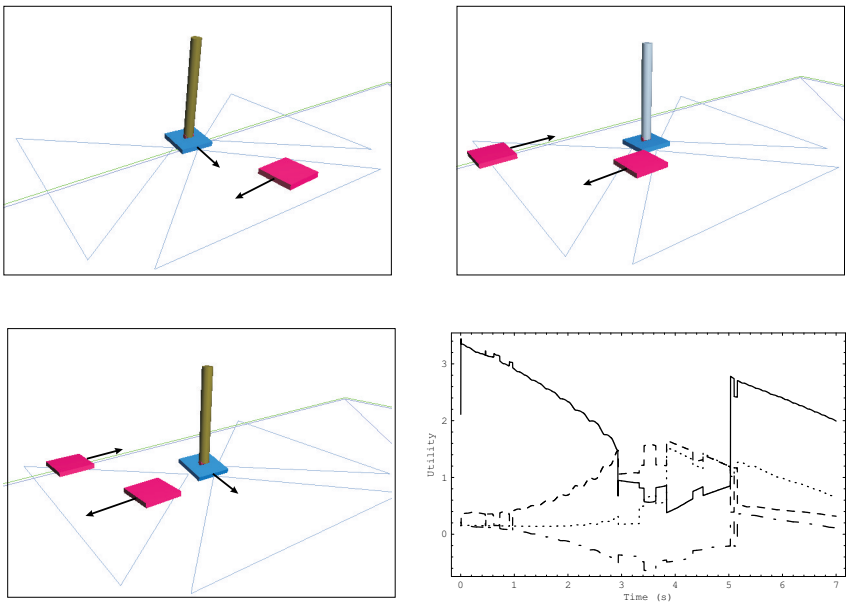

Figure 6.12: *The early stages of a typical run. The bottom right panel shows the variation of the four utility functions in one of the best performing individuals during evolution with the dynamic model. The curves represent the utility values for the behaviors* move forward *(solid curve),* move backward *(dash-dotted curve,),* stop *(dotted curved), and* charge *(dashed curve). Note that the utility of the* move backward *behavior is very low throughout the simulation.*

However, the dynamically simulated robot was unable to traverse the whole arena. The failure could be attributed to the limited quality of the constituent behaviors B1-B4, rather than a failure of the behavioral organization system. Thus, an obvious conclusion was that the constituent behaviors must at least reach some minimum level of quality, in order for a behavioral organizer to perform well.

In order to study the evolution of behavioral organizer in its own right, Pettersson and Wahde also carried out some simulations using a much simpler dynamical model (essentially a point-particle model) for the simulated robots. In this case, the EA found utility functions that would allow the robot to traverse the entire arena without collisions. An example of the motion of the simplified robot is shown in Fig. 6.13.

6.3.3 The potential field method

The **potential field method** [18] is an example of a cooperative method, and is used for robot navigation problems. In the potential field method, the robot moves in the direction suggested by the (negative) gradient of an artificial po-

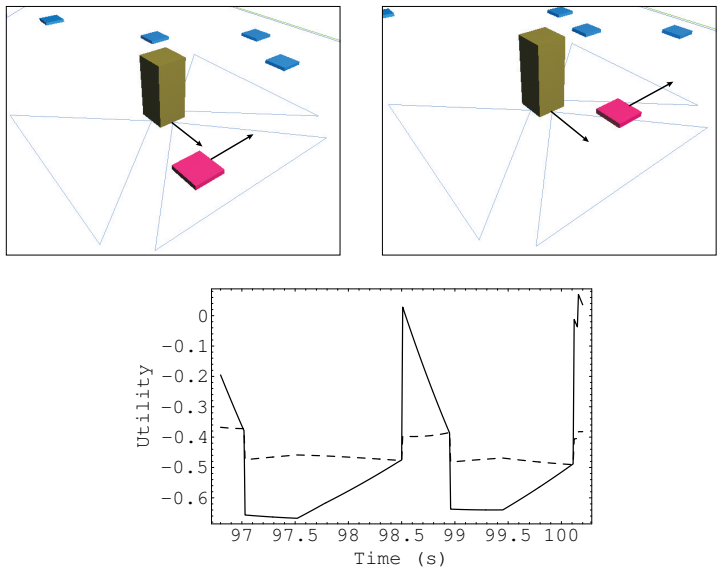

Figure 6.13: *A behavior switching sequence where, at first, behavior B1 (*move forward*) is active. Shortly thereafter behavior B4 (*charge*) is activated due to the detection of an approaching obstacle. As the obstacle passes, and the sensor signal decreases, B1 is activated again. For clarity, only the utility values for B1 (solid line) and B4 (dashed line) are shown in the bottom panel.*

tential field, generated by the objects (such as obstacles) in the vicinity of the robot and by the navigation goal. As shown in Fig. 6.14, the potential field can be interpreted as a landscape with hills and valleys, and the motion of the robot can be compared to that of a ball rolling through this landscape. The navigation goal is usually assigned a potential field corresponding to a gentle down-hill slope, e.g.

$$\Phi^{\mathrm{g}} = k_{\mathrm{g}} \left(\mathbf{x} - \mathbf{x}_{\mathrm{g}} \right)^2 , \tag{6.23}$$

where k_{g} is a positive constant, $\mathbf{x}$ the position of the robot, and $\mathbf{x}_{\mathrm{g}}$ the position of the goal. Continuing with the landscape analogy, obstacles should generate potentials corresponding to steep hills. Thus, the potential of an obstacle can be defined as

$$\Phi^{\mathrm{o}} = k_{\mathrm{o}} \mathrm{e}^{-\frac{(\mathbf{x}-\mathbf{x}_{\mathrm{o}})^2}{w_{\mathrm{o}}^2}} , \tag{6.24}$$

where k_{o} and w_{o} are constants, determining the height and width of the obstacle, respectively, and $\mathbf{x}_{\mathrm{o}}$ is the position of the obstacle. The total potential is given by

$$\Phi = \Phi^{\mathrm{g}} + \sum_{i=1}^{n_{\mathrm{o}}} \Phi_i^{\mathrm{o}}, \tag{6.25}$$

where n_{o} is the number of obstacles. Once the potential field has been defined, the desired direction of motion $\hat{\mathbf{r}}$ of the robot can be computed as the negative

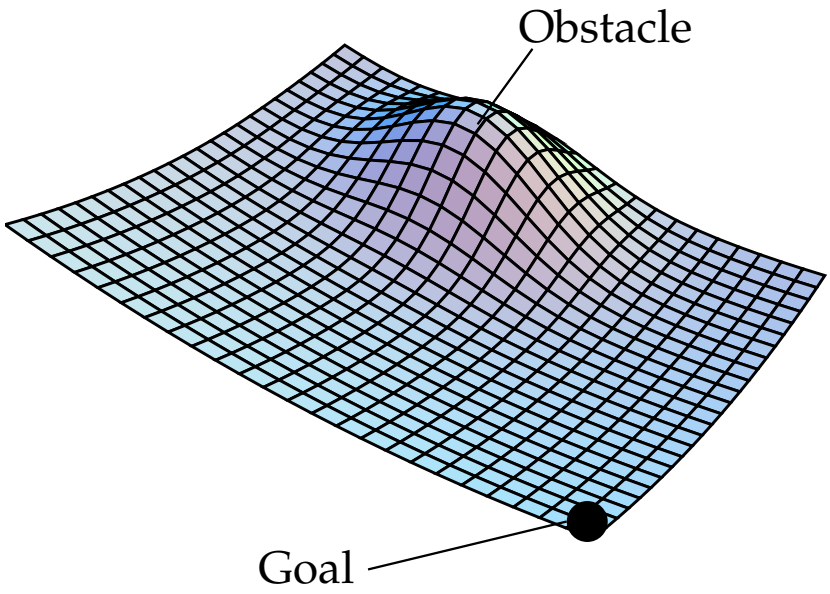

Figure 6.14: *A potential field containing a single obstacle and a navigation goal.*

of the normalized gradient of the field

$$\hat{\mathbf{r}} = -\frac{\nabla\Phi}{|\nabla\Phi|}. \tag{6.26}$$

In order to integrate the equations of motion of the robot, it is not sufficient only to know the desired direction: the magnitude of the force acting on the robot must also be known. In principle, the unnormalized negative gradient of the field could be taken as the force acting on the robot, providing both magnitude and direction. However, in that case, the magnitude of the force would vary quite strongly with the position of the robot, making the robot a dangerous moving object (if it is large). A safer approach is to attempt to keep the speed of the robot at a constant (or slowly varying) reference value V_{ref}, while following the directions suggested by the vector $\hat{\mathbf{r}}$ defined in Eq. (6.26). Using the equations of motion defined in Chapter 1, the motion of the robot can be taken as

$$M\dot{V} + \alpha V = A\tau_{\mathrm{V}}, \tag{6.27}$$

where

$$\tau_{\mathrm{V}} \equiv \tau_{\mathrm{L}} + \tau_{\mathrm{R}} = \frac{\alpha V_{\mathrm{ref}}}{A} + a\left(V_{\mathrm{ref}} - V\right), \tag{6.28}$$

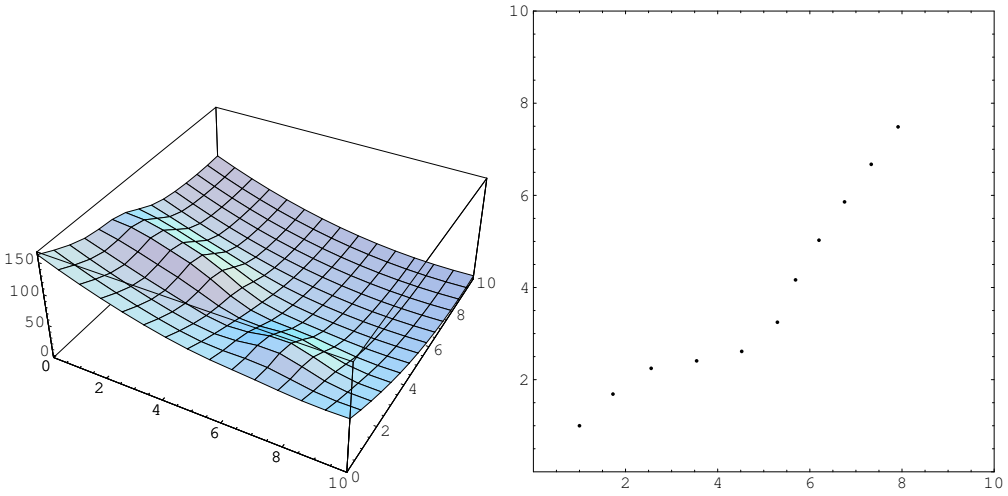

Figure 6.15: *Potential field navigation. The left panel shows a potential field, and the right panel shows the motion (seen from above) of a robot navigating in this field, using the potential field method, towards the goal located at* $(9, 9)$.

where a is a positive constant, and

$$\overline{I}\ddot{\varphi} + \beta\dot{\varphi} = B\tau_{\varphi}, \tag{6.29}$$

where

$$\tau_{\varphi} \equiv -\tau_{\mathrm{L}} + \tau_{\mathrm{R}} = b\left(\varphi_{\mathrm{ref}} - \varphi\right), \tag{6.30}$$

where φ_{ref} is the desired direction of motion, i.e. the direction of the vector $\hat{\mathbf{r}}$, and b is a positive constant. This simple, proportional control law will be sufficient if the reference angle φ_{ref} varies slowly.

Waypoint navigation With any gradient-following method, there is always a risk that the robot will become stuck in a local minimum, and the potential field method is no exception. In Fig. 6.16 a robot is shown in an environment containing three objects forming a wedge-shaped obstacle, and a goal. The corresponding potential field will attract the robot towards the goal, its potential dominating over that from the obstacles at the robot's starting position. However, as the robot approaches the obstacles their repelling potentials will begin to be noticeable. Attracted by the goal, the robot will eventually find itself stuck inside the wedge, at a local minimum of the potential, a situation sometimes referred to as a **locking phenomenon**. In order to avoid locking phenomena, the path between the robot and the goal can be supplied with **waypoints**, represented by small attractive potentials Φ^{p} given by

$$\Phi^{\mathrm{p}} = k_{\mathrm{p}} \mathrm{e}^{-\frac{(\mathbf{x}-\mathbf{x}_{\mathrm{p}})^2}{w_{\mathrm{p}}^2}}, \tag{6.31}$$

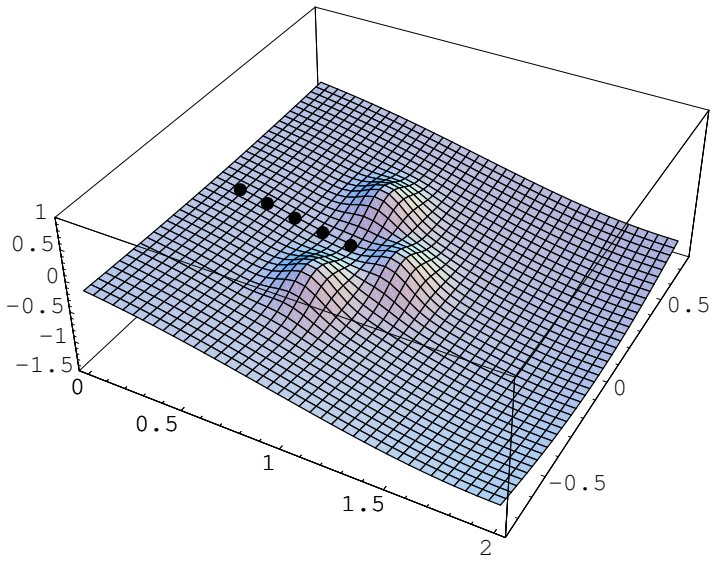

Figure 6.16: *The locking phenomenon. Following the gradient of the potential field the robot, shown as a black dot, moves towards the goal, located at* $(2, 0)$*. However, the three obstacle potentials generate a local minimum, where the robot eventually gets stuck.*

i.e. the same form as the obstacle potentials, but with *negative* constants k_{p}. Of course, the introduction of waypoints leads to the problem of determining where to put them. An analysis of such methods will not be given here. Suffice it to say that the problem of waypoint placement can be solved in various ways to aid the robot in its navigation.

Waypoints are removed once the robot has passed within a distance R from them, to avoid situations in which the robot finds itself stuck at a *waypoint*.

Evolving the potential field Needless to say, the performance of a robot navigating by means of the potential field method will be determined by the values chosen for the parameters V_{ref} (if applicable), k_{g}, k_{o}, and w_{o}, as well as k_{p} and w_{p} if waypoints are used. The parameters can often be set by trial and error. In more complex situations, an evolutionary algorithm can be used for optimizing the potential field. In such a case, the chromosome would encode the parameters just listed, and an ordinary genetic algorithm could be used for finding optimal parameter values.

6.3.4 Other methods

In addition to the two methods just described, many other methods for behavioral organization have been suggested in the literature, starting with the subsumption method developed by Brooks [3]. The methods of **activation net-**

works [22] and **distributed architecture for mobile navigation** (DAMN) [34] are often cited, but will not be described further here.

It should be noted, however, that in many methods suggested to date, it is not uncommon, as pointed out in [1], [23], and [43], that the user must specify the architecture of the behavioral organization system by hand. This is far from optimal, for several reasons: First, behavior-based robots are generally expected to operate in unstructured environments, which cannot easily be predicted, making it difficult to tune parameters by hand. Second, even if the environment happens to be fairly well structured, it is difficult to assess (at least quantitatively) the relevance of some behaviors, especially those that are not directly related to the assigned task of the robot. For example, a battery-driven robot that is assigned the task of transporting goods between two locations, must occasionally recharge its batteries. Clearly, the behavior of charging batteries is important, but it is difficult to judge quantitatively its relevance against the relevance of the assigned task, and thus to assign an optimal reward for it. Third, hand-coding a behavioral organizer represents a step away from the biologically inspired behavior-based architecture. The main alternative to hand-coding is, of course, to use EAs.