

Chapter 3

Behavior-based robotics

The quest to generate intelligent machines has now (2007) been underway for about a half century. While much progress has been made during this period of time, the intelligence of most autonomous robots in use today reaches, at best, the level of insects, rather than the level of humans. Indeed, during the last twenty years, many of the efforts in robotics research have been inspired by rather simple biological organisms, with the aim of understanding and implementing basic, survival-related behaviors in robots, before proceeding with more advanced behaviors involving, for example, high-level reasoning. These efforts have been made mainly within the paradigm of **behavior-based robotics** (BBR), an approach to machine intelligence which differs quite significantly from traditional **artificial intelligence** (AI). This chapter begins with a brief discussion of the differences between BBR and AI. Next, a more detailed introduction to BBR will be given. Finally, the topic of generating basic robotic behaviors (by hand) will be considered.

3.1 Approaches to machine intelligence

The field of **machine intelligence** was founded in the mid 1950s, and is thus a comparatively young scientific discipline. It was given the name **artificial intelligence** (AI), as opposed to the natural intelligence displayed by certain biological systems, particularly higher animals. The goal of AI was to generate machines capable of displaying human-level intelligence. Such machines are required to have the ability to reason, make plans for their future actions, and also, of course, to carry out these actions.

However, as noted above, despite a near half-century of activity in this area, no machines displaying human-level intelligence are, as yet, available. True, there are machines that display a limited amount of intelligent behavior, such as vacuum-cleaning and lawn-mowing robots, humanoid robots of var-

ious kinds, and even elevators, automatic trains, TV sets and other electronic equipment. However, the intelligence of these machines is very far from the human-level intelligence originally aimed at by AI.

To put it mildly, the construction of artificial systems with human-level intelligence turned out to be difficult, for a number of reasons. First, intelligence is hard to define. In a way, intelligence is a concept which is easy to spot when one sees it (and it is even easier to spot its opposite), but it is still difficult to produce a clear and unambiguous definition. Second, and perhaps more important, human-level intelligence is, of course, extremely complex, and therefore hardly the best starting point. The complex nature of human brains is difficult both to understand and to implement, and one may say that the preoccupation with *human* level intelligence in AI research has probably been the most important obstacle to progress.

So, is it at all possible to generate intelligent behavior in artificial systems? It is, but one must take a less narrow view of intelligence, and this is exactly what is done in BBR, which is the main topic of this course.

3.2 Behavior-based robotics

The concept of BBR was introduced in the mid 1980s, and was championed by Rodney Brooks [3] and others. Nowadays, the behavior-based approach is used by researchers worldwide, and it is often strongly influenced by ethology, as discussed in Chapter 2.

3.2.1 Comparison with classical AI

BBR approaches intelligence in a way that is very different from the classical AI approach, and a schematic illustration of the difference is given in Fig. 3.1. In classical AI, the flow of information is as shown in the left panel of the figure. First, the sensors of the robot sense the environment. Next, a (usually very complex) **world model** is built, and the robot reasons about the effects of various actions within the framework of this world model, before finally deciding upon an action, which is executed in the real world.

Now, this procedure is very different from the distributed form of computation found in the brains of biological organisms, and, above all, it is generally very *slow*, strongly reducing its survival value. Clearly, this is not the way most biological organisms function. As a good counterexample, consider the evasive maneuvers displayed by noctuid moths, as they attempt to escape from a pursuing predator (e.g. a bat) [8]. A possible way of achieving evasive behavior would be to build a model of the world, considering many different bat trajectories, and calculating the appropriate response. However, even if the brain of the moth were capable of such a feat (it is not), it would most

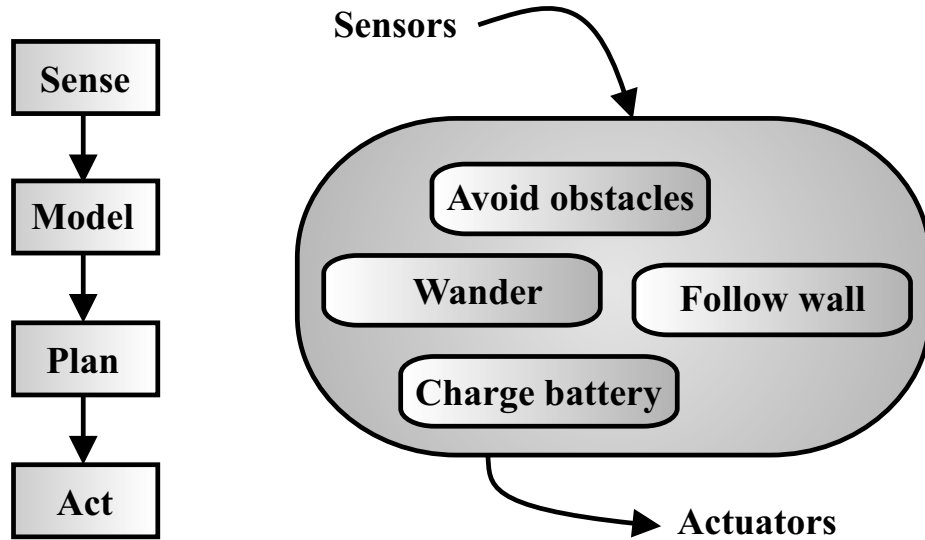


Figure 3.1: A comparison of the information flow in classical AI (left panel) and in BBR (right panel). For BBR, any number of behaviors may be involved, and the figure only shows an example involving four behaviors.

likely find itself eaten before deciding what to do. Instead, as will be discussed below, moths use a much simpler procedure.

As is evident from the left panel of Fig. 3.1, classical AI is strongly focused on high-level **reasoning**, i.e. an advanced cognitive procedure displayed in humans and, perhaps, some other mammals. Attempting to emulate such complex biological systems has proven simply to be too complex as a starting-point for research in robotics: Classical AI has had great success in many of the subfields it has spawned (e.g. pattern recognition, path planning etc.), but has made little progress toward the goal of generating truly intelligent machines, capable of autonomous operation. For a comprehensive introduction to classical AI, see e.g. [35].

BBR, illustrated in the right panel of Fig. 3.1 is an alternative to classical AI, in which intelligent behavior is built in a bottom-up fashion, starting from simple **behaviors**, many of which may be running simultaneously in a given robotic brain, giving suggestions concerning which actions the robot ought to take.

Returning to the example of the moth, its evasive behavior is very simple indeed, and is in fact based on only a few neurons and an ingenious positioning of the ears on its body. This simple system enables the moth to fly away from an approaching bat and, if it is unable to shake off the pursuer, start to fly erratically (and finally dropping toward the ground) to confuse the predator [8].

3.2.2 Behaviors and actions

An attempt should be made to define the concepts of **behaviors** and **actions**, since they are used somewhat differently by different authors. Here, a behavior will be defined simply as a sequence of actions performed in order to achieve some goal. Thus, for example, an obstacle avoidance behavior may consist of the actions of stopping, turning, and starting to move again in a different direction. Note, however, that the definition is not very strict, and that the terms *behavior* and *action* remain somewhat blurred.

3.2.3 Intelligent behavior and reasoning

The example with the moth shows that **intelligent behavior** does *not* require reasoning, and in BBR one generally uses a more generous definition of intelligent behavior than that implicitly used in AI. Thus, in BBR, one may define intelligent behavior as *the ability to survive, and to strive to reach other goals, in an unstructured environment*. This definition is more in tune with the fact that most biological organisms are capable of highly intelligent behavior in the environment for which they have been designed, even though they may fail quite badly in novel environments (as illustrated by the failure of e.g. a fly caught in front of a window). An **unstructured environment** is an environment that changes rapidly and unexpectedly, so that it is impossible to rely on pre-defined maps, something that holds true for most real-world environment. For example, if it is the task of a robot to transport equipment in a hospital, it must first be able to avoid (moving) obstacles. Putting a complete plan of the hospital into the robot will not be sufficient, since the environment changes on a continuous basis. The robot may pass an empty room, only to find the room full of people or equipment when it returns.

3.2.4 Features of behavior-based robots

Different behavior-based robots generally share certain basic features, even though not all features are present in all such robots. To start with, behavior-based robots are first provided with the most basic behaviors such as obstacle avoidance or battery charging, needed to function in an unstructured environment. More complex behaviors are added at a later stage. Second, several behaviors are generally in operation simultaneously, and the final action taken by the robot represents either a choice between the suggestions given by the various behaviors, or an average action formed from the actions suggested by several behaviors. Third, BBR is mostly concerned with autonomous robots, i.e. robots that are able to move freely and without direct human supervision. Finally, the concept of **situatedness** is a central tenet of BBR: Behavior-based robots do not build complex, abstract world models. Instead, behavior-based

robots are generally situated (i.e. operate in the real world), and many behaviors in BBR are **reactive**, i.e. have a direct coupling between sensors and actuators. However, **internal states**, which provide the robot with, for example, motivation (see Chapter 2) or short-term memory, are of course allowed and are often very useful, but, to the extent that such states are used, they are not in the form of abstract world models.

Note that, in BBR, it is often necessary to use simulations before an implementation in an actual robot is attempted. While the use of simulations represents a step away from the requirement of situatedness, the simulations used in BBR differ strongly from the construction of abstract world models: Just like a physical behavior-based robot, a simulated robot in BBR will rely only on the information it can gather through its simulated sensors, and the reading of a simulated sensor, in itself, never provides information that the physical counterpart to the sensor would be unable to provide. However, no simulation can capture all the facets of reality. Thus, it is important to test, in real robots, any results obtained through simulations.

In general, the brain of a behavior-based robot is built from a set of basic behaviors, known as the **behavioral repertoire**. The construction of such a brain can be considered a two-stage process: First the individual behaviors must be defined (or evolved). Next, a system for selecting which behavior(s) to use in any given situation, must be constructed as well. Clearly, in any robot intended for complex applications, the behavioral selection system is just as important as the individual behaviors themselves. For example, returning to the example of the hospital robot, it is clear that if it starts to run out of power, it must reprioritize its goals and quickly try to find a power supply, even if, by doing so, it comes no closer to achieving its task of delivering objects. However, the topic of behavior selection (also known as behavior coordination or behavioral organization), will be considered in Chapter 6. In this chapter, the aim is simply to generate basic behaviors.

3.3 Generating behaviors

In general, a behavior-based robot is first equipped with the most fundamental of all behaviors, namely those that deal with survival. In animals, survival obviously takes precedence over any other activity: Whatever an animal (possibly a human) is doing, it will suspend that activity if its life is threatened. What does survival involve in the case of a robot? In order to function, a robot must, of course, be structurally intact, and have a non-zero energy level in its batteries. Thus, examples of survival-related behaviors are *collision avoidance*¹ and *battery charging*. However, even more importantly, particularly for large

¹Names of behaviors will generally be put in *italics*.

robots, is to avoid harming *people*. Thus, the purpose of collision avoidance is often to protect people in the surroundings of the robot, rather than protecting the robot itself. Indeed, one could imagine a situation where a robot would be required to sacrifice itself in defense of a human (to some extent, this is what robots used by bomb squads do already today). These ideas have been summarized beautifully by the great science fiction author Isaac Asimov in his three laws of robotics, which are stated as follows

First law: A robot may not injure a human being, or, through inaction, allow a human being to come to harm.

Second law: A robot must obey orders given it by human beings, except where such orders would conflict with the first law.

Third law: A robot must protect its own existence as long as such protection does not conflict with the first or second law

While Asimov's laws certainly serves as an inspiration for researchers working on autonomous robots, a full implementation of those laws would be a daunting task, requiring reasoning and deliberation by the robot on a level beyond the reach of the current state-of-the-art. However, in a basic sense, BBR clearly deals with behaviors related to Asimov's laws. In the following sections, a few specific examples of behavior implementations, using the Matlab model of the simple, differentially steered robot introduced in Chapter 1, will be given. Only behaviors implemented by hand will be considered here, the evolution of behaviors being deferred to the next chapter.

3.3.1 Behavioral architectures

Before implementing a behavior one must, of course, select a **behavioral architecture**², i.e. a general framework for the implementation. Unlike researchers in many other fields of science, a robotics researcher has a great deal of freedom in the selection of the models and methods used. This is so, since, in robotics, anything that works (robustly) is, per definition, correct. While, for example, biological organisms may serve as an inspiration for the implementation of robotic behaviors, there is absolutely no need to follow biology exactly. This state of affairs is sometimes an advantage, and sometimes not: One problem is the fact that there is really no single, standard benchmark method to which a new method can be compared. However, in practice, the number of frequently used behavioral architectures is quite limited. For example, in connection with evolutionary robotics, it is common to use (continuous-time)

²The word *architecture* is here used in a sense similar to that implied by the term *computer architecture*, i.e. a manner of organizing the components (in the case of BBR, the basic actions taken by the robot) in a certain way.

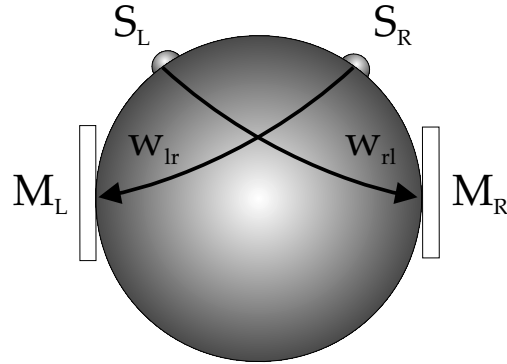


Figure 3.2: A Braitenberg vehicle equipped with elementary pursuit behavior.

artificial neural networks, even though other architectures are certainly used as well. For hand-coded behaviors, i.e. behaviors that are defined completely by hand, neural networks are rarely used, simply because they are, by virtue of the distributed nature of their computation, very difficult to generate by hand. Instead, other architectures are used, and here architectures based on simple *if-then-else* rules, combined with Boolean state variables, will be considered. This architecture is particularly suited for hand-coded behaviors, and can also be represented in the form of **finite state automata**. First, however, behaviors implemented as *very simple* neural networks will be discussed briefly.

3.3.2 Braitenberg vehicles

Even very simple networks can make a robot display some form of basic intelligent behavior. This was noted by, among others, Valentino Braitenberg [2]. He described, in an informal way, how intelligent artificial creatures could be constructed incrementally starting from very simple ones. The term **Braitenberg vehicle** will be used here to denote an artificial creature with a direct sensor-actuator mapping, even though Braitenberg did describe much more complex creatures with memory, shape and pattern recognition etc.

As an example, consider a creature with two light sensors (e.g. infrared) placed in its direction of motion, and two actuators (e.g. wheels), one on each side of its body. If the sensors are connected to the actuators in the manner shown in Fig. 3.2, the creature will be a pursuer, i.e. it will tend to move toward a detected object, since the left sensor (S_L) will activate the motor (M_R) controlling the right wheel.

Using the equations for a differentially steered robot, from Chapter 1, and neglecting the motor model (i.e. assuming that the torques generated by the brain of the robot are applied directly to the wheel axes), the equations of mo-

tion can be written

$$M\dot{V} + \alpha V = A(\tau_L + \tau_R) = A(w_{lr}S_r + w_{rl}S_l), \quad (3.1)$$

$$\bar{I}\ddot{\varphi} + \beta\dot{\varphi} = B(-\tau_L + \tau_R) = B(-w_{lr}S_r + w_{rl}S_l), \quad (3.2)$$

where w_{lr} and w_{rl} are the (positive) weights connecting the sensors to the motors. The ability of this robot to pursue a target is of course limited, to put it mildly, and dependent on the values of the connection weights. If the connection weights are made too large, the robot will tend to spin, whereas if the weights are too small, it will have difficulties turning.

Still, with the proper choice of connection weights, a simple pursuer can be made using only two neurons. Strictly speaking, the connections between the sensors and the actuators are not neurons at all, but input elements, since they only mediate the signals from the input and thus do not perform any computation.

In the case just described, the training of the neural network would be very simple: it would amount to setting the values of only two weights. Normally, the neural networks used for robot control are much more complex, however, and are often recurrent. As mentioned above, such networks are very difficult to generate by hand, and are instead most often used in connection with artificial evolution, a topic that will be considered in the next chapter. For the remainder of this chapter, hand-coded behaviors will be considered.

3.3.3 Hand-coded behaviors

Writing *reliable* behaviors for autonomous robots is more difficult than it might seem at a first glance, particularly for robots operating in realistic (i.e. noisy and unstructured) environments, for which it is hard to predict all possible situations that the robot might encounter. In the BBR paradigm, as has been mentioned before, robotic brains commonly consist of several simple behaviors which together define the overall behavior of the robot. However, there is no well-defined method for deciding the exact composition of a behavior: For example, in some applications, navigation and obstacle avoidance may be parts of the same behavior whereas, in other application, the behaviors *straight-line navigation* and *obstacle avoidance* may be separate behaviors, as illustrated beautifully by the kinesis of *E. Coli* in Chapter 2, an example that also illustrates an important principle, namely that of keeping individual behaviors very simple.

The writing of hand-coded behaviors will now be illustrated by means of an example. The example will make use of the `ARSim` Matlab robot simulator for differentially steered robots, introduced in Chapter 1.

The program flow during execution of a behavior is given in Fig. 1.14 in Chapter 1. Thus, in each time step of the simulation, (1) the robot probes the

state of the environment using its sensors. With the updated sensor readings, the robot then (2) selects an action, i.e. generates motor signals (one for each motor), which are then sent to the motors, and (3) the position and velocity are updated. Finally, (4) the termination criteria (for example, collisions) are considered.

Note that, for such a simulation to be realistic (i.e. implementable in a real robot), the time required, in a corresponding real robot, for the execution of steps (1) - (2) must be *shorter* than the simulation time step. By default, the Matlab simulator (ARSim) uses an updating frequency of 100 Hz (i.e. a time step of 0.01 s), which is attainable by the simple (e.g. IR) sensors used here³. Furthermore, in the simple behaviors considered here, the deliberations in step (2) usually amount to checking a few *if-then-else*-clauses, a procedure that a modern processor completes within a few microseconds. At the end of step (2), the motor signals can be sent to the motors which, of course, also must be able to update their output within a time interval no longer than the time step, as illustrated in Fig. 1.15 in Chapter 1. Cases where a motor with a slower updating frequency is used can easily be handled, simply by limiting the updating frequency of the motor signals, regardless of the signals sent to the motor as a result of the execution of step (2). Finally, an updating frequency of 100 Hz is not unrealistic for a DC motor. However, also in this case, a state variable can be introduced, limiting the allowed updating frequency (to a lower value).

The writing of basic behaviors for autonomous robots will be exemplified in the form of a *wandering* behavior, which allows a robot to explore its surroundings, provided that no obstacles are present. Other basic behaviors, such as e.g. *obstacle avoidance*, *wall-following* etc. will be considered in the exercises.

Example: wandering

The task of robot navigation, in a general sense, is a very complex one, since it normally requires that the robot should know its position at all times which, in turn, requires accurate **positioning** (or **localization**), a procedure which will not be considered in this chapter. However, simpler aspects of the motion of a robot can be considered without the need to introduce localization. For example *wandering* is an important behavior in, for instance, an exploration robot or a guard robot that is required to cover an area as efficiently as possible. Below, a few versions of wandering, with varying degrees of complexity, will be considered.

The simplest form of wandering, following an asymptotically circular path, can be achieved simply by setting the motor torques of the robot to different,

³In cases where the simulation time step is shorter than the time required to update sensor readings, a situation that may occur, for example, if binocular vision is used, a **readability state** can be added to the simulated sensors, such that they are updated only with the frequency that could be realistically achieved in the real robot, see also Chapter 1.

constant values. In the Matlab `ARSim` robot simulator, for each time step, the motor signals are generated by the brain of the robot. After being rescaled to the appropriate voltage range, the motor signals are then sent to the motors. Next, the motors return the actual torque values, which are used in the integration of the equations of motion (see the `MoveRobot` function). The motor signals are generated in the function `BrainStep`, which for the simple example of a circular path takes the form shown in code listing 3.1.

Code 3.1: *The BrainStep function for the first example of wandering.*

```

1 function b = BrainStep(robot, time);
2
3 b = robot.Brain;
4
5 b.LeftMotorSignal = 1.0;
6 b.RightMotorSignal = 0.5;

```

Note that the variable `time` is included in the `BrainStep` function, even though *this* particular behavior does not use it. Using default parameter settings, the motion of the robot during the first three seconds is shown in the left panel of Fig. 3.3. In order to achieve a slightly more complex behavior, the robot can be made to modify its motor torques with some probability p . In this case, the average number of time steps between motor torque updates will be $\Delta N = 1/p$. The parameter p can be set in the `CreateBrain` function which, for this example, takes the form

Code 3.2: *The CreateBrain function for the second example of wandering.*

```

1 function b = CreateBrain;
2
3 s = [0.5 0.5];
4 p = 0.01;
5
6 b = struct('LeftMotorSignal', s(1), ...
7           'RightMotorSignal', s(2), ...
8           'TurnProbability', p);

```

Note that the initial values of the motor signals are set in the `CreateBrain` function. At this stage, it is important again to note that a simulated robot may only access information that would be accessible to the corresponding real robot. In the simple behavior considered here, no sensory input is really *needed*, but could nevertheless be used. For example, in a different implementation of *wandering* the robot may be required to move a certain *distance* between turns. However, even though the used of the simulator has easy access to the position and direction of the robot at all times, providing this information directly to the robot is *not* permissible, since it would not be available to a real robot. However, if an odometer is added to the simulated robot, it would be permissible to use its (noisy) readings for measuring distance.

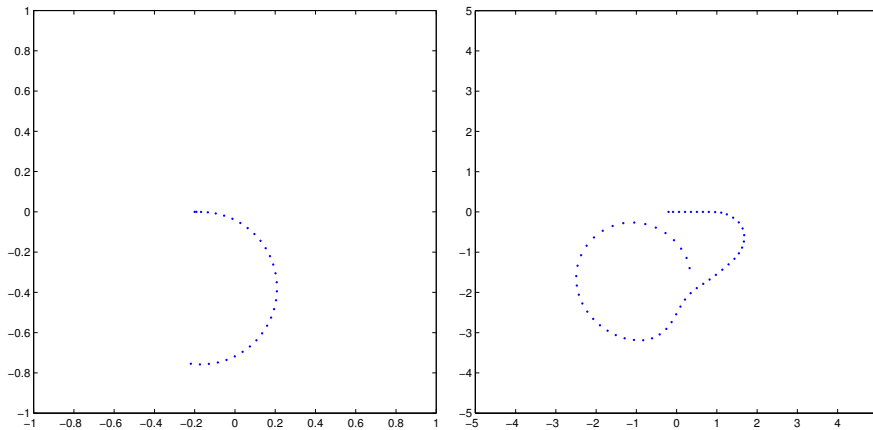


Figure 3.3: Motion of simulated robots, using the motor signal settings defined in code listing 3.1 (left panel) and in code listing 3.3 (right panel). Note the different scales in the two figures.

Anyway, since no sensory input is needed in this behavior, the `BrainStep` function simply takes the form

Code 3.3: *The `BrainStep` function for the second example of wandering.*

```

1 function b = BrainStep(robot, time);
2
3 b = robot.Brain;
4
5 r = rand;
6 if (r < b.TurnProbability)
7     b.LeftMotorSignal = b.LeftMotorSignal - 0.1 + 0.2*rand;
8     b.RightMotorSignal = b.RightMotorSignal - 0.1 + 0.2*rand;
9 end
10
11 if (b.LeftMotorSignal > 1.0)
12     b.LeftMotorSignal = 1.0;
13 elseif (b.LeftMotorSignal < -1.0)
14     b.LeftMotorSignal = -1.0;
15 end
16
17 if (b.RightMotorSignal > 1.0)
18     b.RightMotorSignal = 1.0;
19 elseif (b.RightMotorSignal < -1.0)
20     b.RightMotorSignal = -1.0;
21 end

```

Thus, with a given probability (`TurnProbability`), a small modification of the motor signals is made. The right panel of Fig. 3.3 shows an example of a robot trajectory, taken from a simulation lasting 30 seconds using the random-wandering function defined in code listing 3.3. The instructions on lines 11-21

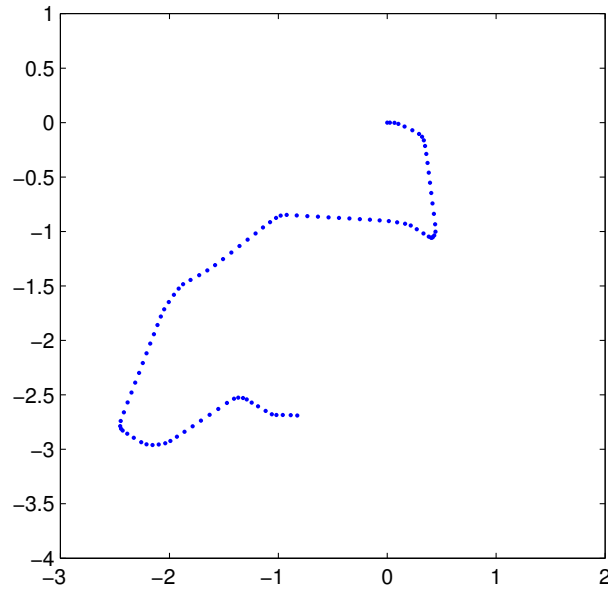


Figure 3.4: Motion of a simulated robot, using the motor signal settings defined in code listing 3.5.

keep the motor signals in the interval $[-1,1]$.

As a final implementation of random walk, consider the following implementations of `CreateBrain` and `BrainStep`:

Code 3.4: The `CreateBrain` function for the third example of wandering.

```

1 function b = CreateBrain;
2
3 s = [0.5 0.5];
4 turning = false;
5 p1 = 0.01;
6 p2 = 0.03;
7
8 b = struct('LeftMotorSignal',s(1),...
9           'RightMotorSignal',s(2),...
10          'Turning',turning,...
11          'TurnProbability',p1,...
12          'StopTurnProbability',p2);

```

As is evident from the code listing, two additional fields have been included in the brain. The `Turning` variable is `true` if the robot is turning, and `false` if it is not. The `TurnProbability` and `StopTurnProbability` variables set the probability of starting or stopping a turn, respectively.

Code 3.5: *The BrainStep function for the third example of wandering.*

```
1 function b = BrainStep(robot , time);
2
3 b = robot.Brain;
4
5 if (b.Turning)
6     r = rand;
7     if (r < b.StopTurnProbability)
8         b.LeftMotorSignal = 0.5;
9         b.RightMotorSignal = 0.5;
10        b.Turning = false;
11    end
12 else
13     r = rand;
14     if (r < b.TurnProbability)
15         s = rand;
16         if (s < 0.5)
17             b.LeftMotorSignal = 0.5;
18             b.RightMotorSignal = -0.5;
19         else
20             b.LeftMotorSignal = -0.5;
21             b.RightMotorSignal = 0.5;
22         end
23         b.Turning = true;
24     end
25 end
```

In each call to `BrainStep`, the robot checks whether or not it is currently executing a turn. If it *is*, it checks whether or not it should stop turning, and vice versa if it is not turning. The motion of a robot using the random wandering behavior implemented in code listing 3.5 is shown in Fig. 3.4. Note that a switch between two actions (for example turning or moving forward) alternatively could be implemented using leaky integrators, as described in the *E. Coli* example in Chapter 2.

