Chapter 4

# Evolutionary robotics

## 4.1 Introduction and motivation

The aim of **evolutionary robotics** (ER) is to use EAs to evolve robotic brains[1] (or bodies, or both), rather than designing them by other means, e.g. by hand. There are many reasons for using artificial evolution to generate robotic brains [12], [13], one of the most important being that it is very difficult to design such systems by hand in any but the simplest cases. In a non-controlled setting, i.e. any realistic environment, there will always be many sources of noise, as well as both stationary and moving obstacles and, perhaps, other robots as well. Trying to predict which situations may occur in such an environment is a daunting task, and hand-coded robotic brains are therefore generally non-robust and prone to failure. On the other hand, properly evolved robotic brains, i.e. those generated using either physical robots or simulations involving realistic noise at all levels, are often able to cope quite well with their environment, even though, in fairness, it should be mentioned that the robotic brains evolved so far are quite simple compared to the ultimate goal of truly intelligent machines. However, the complexity of evolved robotic brains is steadily increasing, and EAs are now used not only for constructing simple behaviors, but also for generating complex systems for behavioral organization, as will be discussed in Chapter 6. Furthermore, evolution (whether natural or artificial) is often able to find solutions that are remarkably simple, yet difficult to achieve by other means.

Another advantage with EAs in connection with robots, is that their ability to function even with very limited feedback. This is important in robotics, where it is often known *what* a robot should do, but perhaps not *how* it should

[1] As mentioned earlier, the term **robotic brain** will be used instead of the term **control system**, since the latter term is indicative of the more limited types of systems considered in classical control theory.

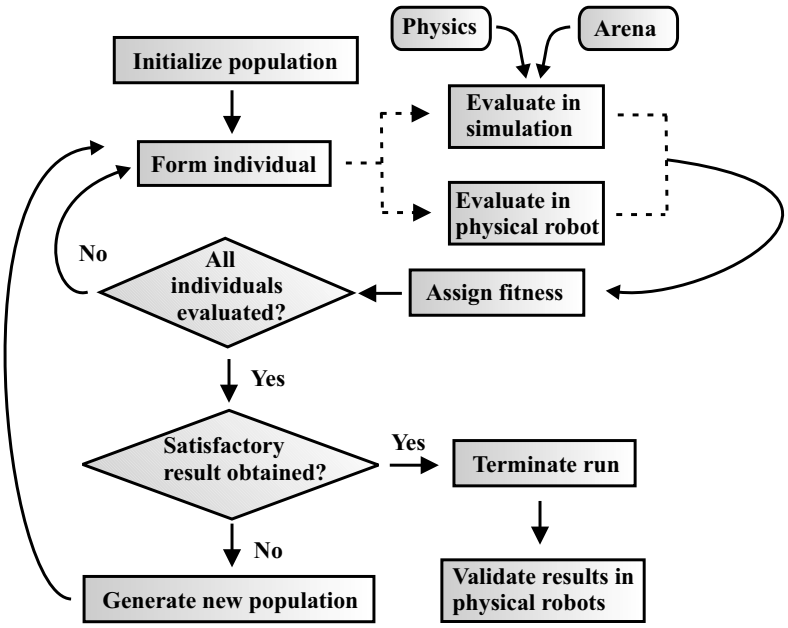


**Figure 4.1:** *The flow of an ER investigation. Note that some steps, e.g. formation of individuals and new populations, are shown in a strongly simplified way.*

do it, e.g. in what order different behaviors should be activated, at what point a task should be suspended in order to engage in self-preserving activities such as obstacle avoidance etc. In other words, it is difficult to specify directly the adequacy of any single action: Feedback is often obtained long after an action is performed, resulting in a **credit assignment problem**, i.e. the problem of assigning credit (or blame) to previous actions. EAs are very well suited to this kind of problems, where only rather vague guidance can be given (an example will be given below).

Of course, as with all methods, there are also some drawbacks with ER, one of the most serious being that, in view of the many candidate robotic brains that must be examined before a satisfactory one is found, it is usually required to resort to the use of simulations, rather than evolving in actual, physical robots. Making realistic simulations is indeed difficult, and there exists a **reality gap** [17] between simulated and physical robots. However, it is possible to overcome the reality gap, and the need for using simulations does not, in fact, introduce fundamental limits on the results obtained from ER.

The flow of an ER investigation is shown in Fig. 4.1, and it is quite similar to the flow of a general EA. The figure is simplified: The procedure of forming individuals can involve anything from setting a few parameters to running a complex developmental process, generating both the brain and the body of a robot. The dashed arrows in the figure indicate a choice: As mentioned above,

in many ER investigations, simulations are used and, as indicated in the figure, the evaluation of an individual requires simulating the physics of the arena in which the evaluation takes place. The other option is to use actual, physical robots, and, in this case, the most common approach is to generate individuals in a computer, upload them one by one on a single robot, evaluate the robot, and return its fitness. There are other approaches as well (not shown in the figure), such as **embodied evolution** [9], [47], where the entire evolutionary process takes place on a population of physical robots. Incidentally, if evolution is carried out in physical robots according to the scheme shown in Fig. 4.1, some restrictions are placed on the fitness measure, since it must be possible for the robot to assess its fitness and return the result to the computer, at least in cases where the EA is to operate without continuous human supervision. Thus, the fitness measure must be based on quantities that are readily available to the robot, such as e.g. sensors readings, motor speeds etc. [10].

The final step in Fig. 4.1, i.e. the validation of the results in physical robots, is perhaps the most important step. A robotic brain that only functions in simulation is of little use. Note that validation should be carried out even if the evolution has been performed using a single physical robot. This is so, since all physical robots have individual characteristics. For example, there is no such thing as two completely identical IR sensors, even if they come from the same manufacturer and have the same product number.

If the validation does not give satisfactory results, there are two different options, either to extend the EA run, or to attempt to adjust the robotic brain (or body) by hand. A useful approach in extended runs may be to switch from simulations to physical robots. Even a short extended run may then lead to improved results [25].

### 4.1.1 A simple example

As a simple example of an ER investigation, the evolution of a simplified cleaning behavior will be studied. Consider an arena of the kind shown in the left panel of Fig. 4.2. The large cylinder represents a simple, differentially steered, simulated two-wheeled robot, whereas the smaller (stationary) objects are considered to be garbage, and are to be removed by the robot. The aim is to use an evolutionary algorithm to generate a brain capable of making the robot clean the arena.

#### Cleaning behavior in simulation

The first choice that must be made is whether to evolve robotic brains in simulation, or directly in hardware For the simple problem described here, the choice was made to use simulated robots during evolution, and then attempt to transfer the best robotic brain found in simulation to a physical robot.

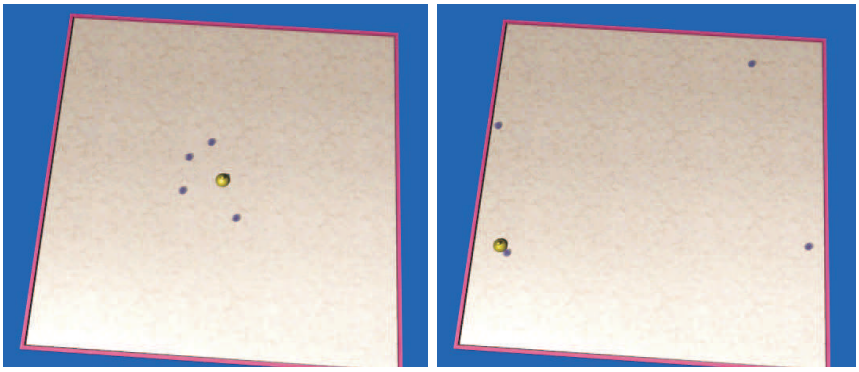

**Figure 4.2:** *A simple, simulated cleaning robot (the large, circular object) in action. The initial state is shown in the left panel, and the final state in the right panel.*

Next, a representation must be chosen for the robotic brains. Ideally, the EA should be given as much flexibility as possible but, in practice, some limitations must generally be introduced. In this problem, the robotic brains were represented as sequences of `IF-THEN-ELSE` rules as shown in Fig. 4.3 and the EA acted directly on the rules rather than on a chromosomal encoding of them. In the beginning of the ER simulation, the number of rules was small. However, the EA was allowed to change the number of rules during the simulation. The simulated robot was equipped with very simple sensors that could distinguish garbage objects from walls, but not much more.

The next step in the application of an EA is to choose a suitable fitness measure, i.e. a performance measure for the evolving robotic brains. In the particular case considered here, the aim of the robot was to place the garbage objects as far from the center of the arena as possible. Thus, the fitness measure was simply chosen as the mean square distance, counted from the center of the arena, of all the garbage objects at the end of the evaluation.

Furthermore, each robot was evaluated against several (usually five) different starting configurations, with garbage objects placed in different positions, to avoid a situation where the evolved robot would learn only to cope with a given configuration.

Next, an EA was set up, in which a population of robotic brains (in the form of rule sequences, as mentioned above) was generated. As the EA progressed, evaluating robotic brains, and generating new ones using selection, crossover, and mutations, better and better results were obtained. In early generations, the simulated robots did little more than run around in circles near their starting position. However, some robots were lucky enough to hit one or a few garbage objects, thereby moving them slightly towards the walls of the arena. Before long, there appeared robots that would hit *all* the garbage objects.

The next evolutionary leap led to purposeful movement of objects. Here, an

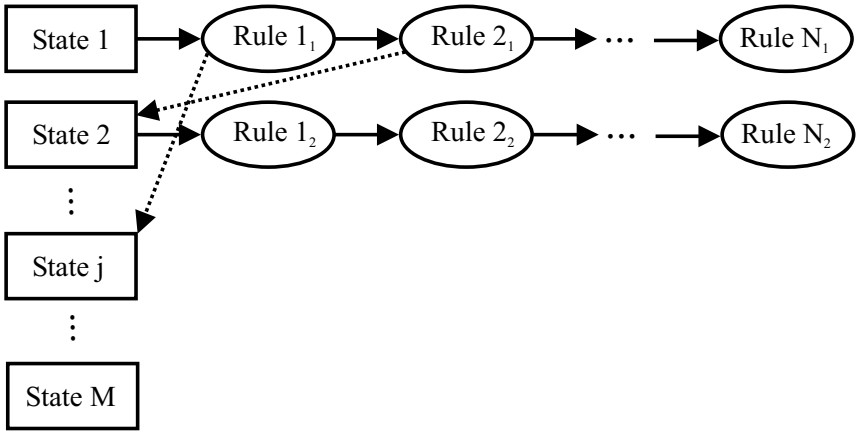


**Figure 4.3:** *The structure of the robotic brains used in the cleaning robot. Each robotic brain consisted of $M$ states, as well as conditional jumps between the states. In this structure, if a given rule is satisfied, a jump is made to the target state. Thus, for example if rule $1_1$ is satisfied, the robot will find itself in state $j$ in the next time step. If no rule is satisfied, the robot will remain in its current state. Rules take e.g. the form IF $s > s_0$, where $s$ is the reading of a proximity sensor and $s_0$ is (an evolvable) constant. Note that, for clarity, rules are only shown for states 1 and 2 in the figure.*

interesting method appeared: Since both the body of the robot and the garbage objects were round, objects could not easily be moved forward; Instead, they would slide away from the desired direction of motion. Thus, a method involving several rules was found, in which the robot moved in a zig-zag fashion, thus managing to keep the garbage object in front.

Next, robots appeared that were able to deliver a garbage object at a wall, and then return towards the center of the arena in a curved, sweeping motion, in order to detect objects remaining near the center of the arena.

Towards the end of the run, the best evolved robots were able to place all, or almost all, garbage objects near a wall, regardless of the starting configuration.

### 4.1.2 Cleaning behavior in a Khepera robot

Needless to say, the aim of ER is to generate real, physical robots capable of performing useful tasks. Simulation is a useful tool in ER, but the final results should be tested in physical robots.

The simulations for the cleaning robot discussed above were, in fact, strongly simplified, and no attempt was made to simulate a physical robot exactly. Nevertheless, the best robotic brain obtained in the simulations was adapted for a Khepera robot (see `www.k-team.com`) the adaptation consisting mainly of rescaling the parameters of the robotic brain, such as e.g. wheel speeds and sensor readings, to appropriate ranges. Quite amazingly, the evolved robotic

brain worked almost as well in the Khepera robot as in the simulated robots. Thus, in this particular case, the transition from simulation to real robots was quite simple, probably due to the simplicity of the problem. However, despite its simplicity, the problem just described still illustrates the power of EAs as a method for generating robotic behaviors.

## 4.2 Issues in evolutionary robotics

Before one starts evolving a robotic brain for a given task, there are several issues that must be considered. In this section, a brief introduction to (some of) those issues will be given, and further descriptions will be given in connection with the examples in the following section. There are many issues that will *not* be covered, however, such as e.g. **interactive evolution**, in which the used provides a **subjective fitness measure** to guide the EA, **evolvable hardware** and **reconfigurable hardware**, where the EA is used for changing the structure of the hardware, and **co-evolution**, i.e. simultaneous evolution of two (or more) populations.

### 4.2.1 Selecting a representation

The choice of representation for the evolving robotic brains, i.e. whether to use neural networks, finite-state machines, `IF-THEN-ELSE`-rules, or some other architecture clearly influences the results that can be obtained. In the ER literature, a common choice, motivated by biological considerations, is to use neural networks (ANNs).

However, ANNs also have several drawbacks. First of all, they essentially function as black boxes and commonly defy attempts at interpretation. Second, in a situation where, say, two elementary behaviors have been generated, and a composite robotic brain (displaying both behaviors) is desired, it is not evident how ANNs should be combined, mainly due to the distributed nature of their computation. Of course, it is possible to maintain two separate neural networks in the robotic brain, and use some method for behavior selection to activate the appropriate network (this topic will be considered further in Chapter 6). However, in that case, some of the biological motivations for the use of ANNs disappear.

In the author's experience, more transparent representations, such as e.g. `IF-THEN-ELSE`-rules, work quite well, in most cases, even though they may not be able to provide the continuous input-output-mapping that ANNs do.

To summarize, it is difficult to make general statements concerning the optimal choice of representations for ER investigations, and the final choice for a given investigation often involves a considerable amount of experimentation.

### 4.2.2 Fitness measures

Perhaps even more than the choice of representation, the results of an ER investigation are influenced by the choice of fitness measure, of which there are many different aspects. Fitness measures can be divided into **explicit** and **implicit** ones, the difference being that an explicit fitness measure attempts to capture many details of the robot's behavior, whereas an implicit one focuses on the overall behavior of the robot. An example of the distinction between explicit and implicit fitness measures will be given in connection with the study of simple navigation behaviors below. A related dichotomy is that of **local** and **global** fitness measures (see the example on box-pushing below). A local fitness measure is one in which the fitness of the robot is (in principle) adjusted at each time step, depending on the actions taken by the robot, whereas a global fitness measure only takes into account the initial and final situations (e.g. the position of the robot).

Second, a distinction can be made between **internal** and **external** fitness measures. An internal fitness measure uses only variables that are available to the robot itself, such as e.g. the readings of its sensors, whereas an external fitness measure may take into account other aspects (such as the exact position of the robot) as well. The distinction is particularly important in the case of evolution in hardware in which case the only information available normally is that acquired by the robot (unless a human observer is available to assess the performance of each evaluated robot).

A related issue concerns the number of **tests** carried out in the evaluation of a particular robotic brain. As was indicated in the simple example above, it is often crucial to carry out multiple tests in order to prevent the EA from finding solutions that only take into account the particular conditions prevailing in a given test.[2]

When several tests are used in the evaluation of a robotic brain, the problem of combining the results into one scalar measure (that can be used by the EA) arises. Such combinations can be carried out in many different ways. A common choice is to consider the average fitness $f_{\text{avg}}$ (over the tests)

$$f_{\text{avg}} = \frac{1}{N}\sum_{i=1}^{N} f_i, \tag{4.1}$$

where $N$ is the number of tests, and $f_i$ is the fitness value obtained in test $i$. Alternatively, the minimum fitness

$$f_{\text{min}} = \min_i f_i, \tag{4.2}$$

[2]Of course, in realistic simulations of autonomous robots, there is noise present on many different levels, making all tests slightly different. However, these differences may not be sufficient to evolve general-purpose behaviors. In the simple cleaning example considered above, the robot would easily adapt its motions to the particular locations for the garbage object, if only one test were to be used.

can be used. The advantage of the latter measure is that it focuses on the *worst* performance of the robot, thus forcing the EA to find robotic brains capable of solving the robot's task in a general manner.

### 4.2.3 Simulations vs. evolution in actual robots

Evolving behaviors is often a time-consuming process, particularly in cases where the evaluation of a single individual takes a long time. In general, an EA needs at least several hundred evaluations (and sometimes thousands or even millions) in order to arrive at a useful result. Thus, at a first glance, evolving behaviors in simulation appear to be a more feasible approach than evolution in hardware and simulations are indeed often used in ER. However, it is important to keep in mind that a simulation is, at best, only a caricature of the real world and, at worst, an outright misrepresentation of it. This view has been championed, among other, by Brooks [4], who takes a particularly pessimistic view of the possibility of transferring results from simulations to reality. There are several reasons for believing that simulations will not transfer well to reality [25], a problem known as the **reality gap** [17]. First of all, it is difficult to generate an accurate physical model of the world, including the motions of real-world objects, variations in lighting etc. Second, real environments are invariably noisy, on all levels. Thus, noise is present in sensor readings, actuator signals, motion etc. Third, there is also a variation in supposedly identical hardware components. For example, Miglino et al. [25] mention a measurement on two sensors of the same type, in which it was found that the range of one sensor was nearly twice as large as the other, and the diffence in angular variation between the two sensors was also of the same magnitude.

#### Evolution in hardware

The obvious alternative to evolving behaviors in simulation is, of course, to evolve in physical robots, and this approach has been used by many authors (see e.g. [9], [10], [14], [47], [48]). However, evolution in hardware is also associated with numerous difficulties. First of all, the evaluation of individuals generally takes longer time than in actual robots than in simulated robots (see e.g. [10] or [14] for timing estimates). In order to arrive at a result in a reasonable amount of time, it is therefore often necessary to resort to an EA with a small population size, and to run it only for a small number of generations [48]. Second, supplying the robots with continuous power is an omnipresent problem: In real robots, the amount of available energy is finite, and so recharging will be necessary at regular intervals. This problem can be overcome by the introduction of a powered floor [9], or by requiring the robots periodically to return to a charging station. Charging batteries is a rather slow process, but the transfer of energy to robots can be speeded up using capacitors instead

of batteries [31]. Third, in many applications, e.g. the evolution of gaits for bipedal robots, the evolutionary process must be monitored continuously.

The most common approach to evolution in hardware is to evaluate individuals in a serial fashion, uploading them one after the other in a single robot [10], [30]. In this case, all evolutionary processes, such as e.g. reproduction and formation of new individuals take place on a computer, and the resulting individuals are then placed in the robot for evaluation.

An alternative method, called **embodied evolution** [9], [47] is to evolve a population of real robots. In embodied evolution, infrared communication can be used for exchanging genetic material between robots. Such exchanges take place only when two robots are within infrared communication range. The probability of a robot sending a gene, can be set proportional to its current performance, and the probability of a robot accepting a gene from another robot (and thus to overwrite its own gene), can be set proportional to one minus the probability of sending the gene.

### Improving simulations

The comparison between simulations and evolution in hardware shows that there are advantages and disadvantages with both approaches. A substantial amount of work has been carried out in order to improve the quality of simulations (see e.g. [15], [16], [17], [30], and [29]). Some of the most important aspects to keep in mind when setting up ER simulations [17] is to (1) base the simulation on empirical data, rather than e.g. artificial sensor models without a real-world counterpart, (2) add the correct level of noise (see below) in all parts of the simulation, (3) use a representation that is noise-tolerant, e.g. an ANN.

Sensors can be implemented in several different ways in ER simulations. One approach is simply to measure sensor readings on an *actual* robot, and store them (for example, in a lookup table) for later use in a simulator [25], [29]. Noise can be added in different ways, either by slightly perturbing the measured sensor readings, or by using actual sensor readings taken from a slightly different position and angle than that currently held by the robot, a procedure called **conservative position noise** [25]. In this approach, sensor readings must be taken at many different positions, rendering the procedure quite time-consuming in all but the simplest environments. An alternative approach is to set up a physical model of the sensors, and to determine the values of the model parameters through system identification. For example, Jakobi et al. [17] used this approach to model the IR sensors on a Khepera robot. They used a ray tracing technique which in which $n$ ray tracings were carried out (in different directions), and the resulting sensor reading was then

modelled as (cf. Chapter 1)

$$s = \sum_{i=1}^{n} \cos \beta_i \left( \frac{a}{d_i^2} + b \right), \tag{4.3}$$

where $\beta_i$ is the angle at which ray $i$ emanates from the sensor, and $d_i$ is the distance to an object along the ray. $a$ and $b$ were determined empirically [17].

Jakobi et al. [17] also studied the effects of noise, and found that the smallest performance difference between simulated robots and physical robots was obtained when the noise level in simulations was set approximately equal to the empirically determined noise level in the real world. Interestingly, it was also found that, in simulations with high noise levels, simulated robots could make use of noise to achieve good results in simulations that could not be reproduced in physical robots, showing that noise levels should not be set too high.

Jakobi [15], [16] has introduced a simulation procedure called **minimal simulations**, which recognizes the inevitable discrepancies between the simulated and real worlds. Thus, in a minimal simulation, the various aspects of the interaction between a robot and its environment are divided into **base set aspects** and **implementational aspects**, where the former have a physical counterpart in the real world, whereas the latter do not. Thus, in a minimal simulation, the implementational aspects should be varied randomly from evaluation to evaluation, thus rendering them useless to evolution, and forcing the EA to focus on the base set aspects. An evolved robotic brain that only relies on base set aspects is called **base set exclusive**. In addition, a certain degree of variation is introduced also in the base set aspects, in order to capture the fact that even in the base set aspects there will be discrepancies between the simulated and real worlds. A robot that can cope also with such discrepancies, is termed **base set robust**.

Thus, summarizing, it is evident that the choice between evolution in simulation and evolution in hardware is a non-trivial one. Simulations have the disadvantage of never being able to capture all aspects of the real world. However, carefully designed simulations will nevertheless lead to results that can be transferred to physical robots. Indeed, one may, perhaps, consider the results obtained in a simulation as a first iteration toward the desired results. Of course, a hybrid approach can be used, in which evolution is first carried out in simulations and then briefly continued in hardware. Miglino et al [25] report successful results from such an approach.

## 4.3 Evolving single behaviors

In this section, some examples of the evolution of single behaviors will be given. However, already at this stage, it can be noted that the definition of

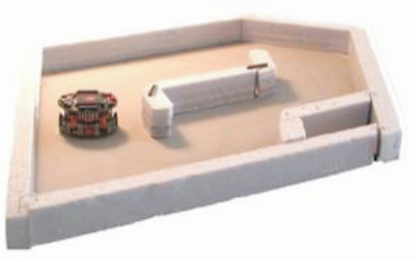

**Figure 4.4:** *The environment used in the evolution of navigation in [10]. Reproduced with kind permission of Prof. D. Floreano.*

behaviors is somewhat fuzzy. For example, in the first example below, i.e. the evolution of navigation, it is normally required that the robots be able not only to move, but also to avoid collisions. Thus, the question of whether to consider motion without collisions as a single behavior or as a combination of two behaviors arises. However, in this particular example, the two aspects of the robot's behavior - motion and obstacle avoidance - are so closely integrated with each other, and can rather easily be achieved through a clever choice of fitness function, that they can perhaps be considered as part of a single behavior.

While the examples are hopefully interesting in their own right, the reader should pay attention to a few particularly important aspects of any application of ER, namely (1) the representation used for the evolving systems, (2) the level of complexity of the simulator (if applicable), and (3) the fitness measure used in the simulations.

### 4.3.1 Navigation

Navigation, i.e. the problem of moving in an environment without colliding with obstacles, is clearly a basic competence of any robot, and it has been studied extensively in the ER literature [10], [24], [36].

#### Basic navigation

In [10], Floreano and Mondada evolved basic navigation. The authors specifically pointed out the difficulties in constructing accurate simulations, and chose instead to evolve navigation behavior in a real Khepera robot. The aim was to evolve collision-free navigation in an environment with stationary obstacles, shown in Fig. 4.4. In the experiments, the authors used a simple, fixed-length chromosome to encode the weights of neural networks of fixed structure. The networks consisted of a single layer of synaptic weights, connecting the 8 IR sensors of the Khepera robot to two output neurons, connected to the

motors. In addition, recurrent connections were introduced within the output layer.

The fitness contribution for each time step during the motion of the robot was chosen as

$$\Phi = V\left(1 - \sqrt{|\Delta v|}\right)(1 - i), \tag{4.4}$$

where $V$ denotes the average rotation speed of the wheels, $\Delta v$ is the difference between the (signed) speeds of the wheels, and $i$ is the value of the IR sensor with the highest reading. $V$, $|\Delta v|$, and $i$ were all normalized to the range $[0, 1]$, so that $\Phi$ also was constrained to this range. The complete fitness function $f$ was obtained by summing the values of $\Phi$ obtained for each time step, and then dividing the results by the number of time steps. The same fitness measure was used also by Nolfi et al. [29].

The first factor ($V$) in Eq. (4.4) promotes high speed, whereas the second factor, $1 - \sqrt{|\Delta V|}$, promotes straight-line motion, and the third factor $(1 - i)$ rewards obstacle avoidance. Note that the different factors in the fitness measure counteract each other, to some extent. For example, the environment was such that the robot was required to turn quite often in order to avoid collisions, even though the second factor in the fitness measure would discourage it to do so. This is an example of what could be called an **explicit fitness measure**, since explicit punishments (in the form of a reduced increase in fitness) are given if the speed is low, if the robot does not move in a straight line, or if the robot approaches an obstacle. The alternative is to use an **implicit fitness measure** for *some* of the aspects of the robot's motion. For example, in [17], Jakobi et al. used a fitness measure similar to the one given in Eq. (4.4), but without the factor $(1 - i)$, since it was found that this factor is unnecessary in a sufficiently cluttered environment, where obstacle avoidance is an implicit requirement for any form of motion.

Floreano and Mondada report that the best evolved robots successfully navigated through their environment, and that some robots also showed intelligent behavior beyond what was explicitly encoded in the fitness function. For example, some of the best robots learned to modulate the speed, keeping it at around three quarters of the maximum allowed speed in the vicinity of obstacles.

While evolving robotic behaviors in hardware has obvious advantages, a possible disadvantage is that it is often quite time-consuming. Indeed, in the experiments reported in [10], each generation lasted around 40 minutes. In order to evolve successful robots, around 50-100 generations were needed.

In addition to basic navigation, Floreano and Mondada also evolved *homing navigation*, in which the robot was required periodically to recharge its (simulated) batteries. However, this was a more complex task, and it will be considered in Chapter 6.

While some recurrent connections were included in the output layer of the

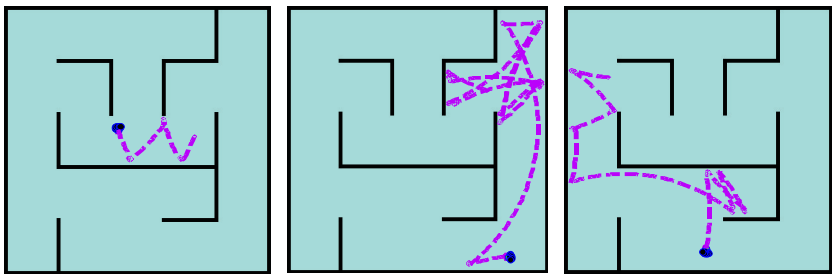

**Figure 4.5:** *The maze used by Nelson* et al. *[27] for the evolution of navigation, reproduced with kind permission of the authors. The figure shows some results obtained in simulation. Qualitatively similar results were obtained using a physical robot, EvBot, shown in Fig. 4.6.*

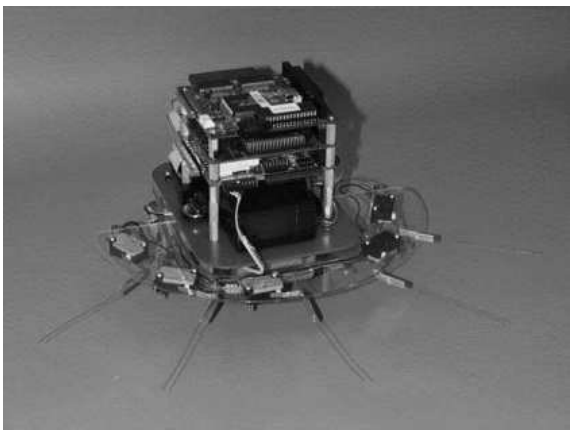

**Figure 4.6:** *The EvBot robot used in [27]. The robot is equipped with an array of tactile sensors, and can also be fitted with other sensors, such as video cameras. Reproduced with kind permission of Nelson* et al.

neural networks used in [10], Nelson *et al.* [27] made more explicit use of temporal processing of information, motivated by the fact that robots commonly show suboptimal performance using reactive behaviors based on very poor sensory inputs.

Thus, Nelson *et al.* considered neural networks with several layers containing recurrent couplings as well as time-delayed couplings. The neural networks were evolved in simulations involving robots equipped with 5 simple tactile sensors. The task of the simulated robots was to navigate in a maze, shown in Fig. 4.5. The fitness measure was taken essentially as the distance travelled, with a penalty for situations in which the robot became stuck.

It was indeed found that robots equipped with memory, in the form of recurrent couplings and time-delay elements, performed the navigation task

better than purely reactive robots. The best results were obtained for networks of moderate complexity, containing 1-3 hidden layers with 5-10 neurons per layer. The best networks obtained during evolution were transferred, with retained functionality, to an actual robot (EvBot, shown in Fig. 4.6), equipped with five tactile sensors.

### Wandering behavior

Miglino et al. [24] evolved an exploratory behavior (also called *wandering*) in simulations. The resulting behaviors were then tested in actual (Lego) robots. In this case, the brains of the robots were represented by very simple neural networks, with two inputs from a front sensor and a rear sensor, two hidden units, and two outputs controlling the two motors. In addition, a single memory neuron was used, connecting the hidden layer to itself through a recurrent coupling. All signals were binary (a Heaviside step function was used as the threshold function in the neurons), and the simple neural networks used in this study could, in fact, be represented as finite-state machines or `IF-THEN-ELSE`-rules.

In order to achieve wandering behavior, the authors used a fitness measure which rewarded movements that would take the robot to locations where it had not been before. The robot was placed in an arena of size $2.6 \times 2.6$ meters, consisting of squares with 10 cm side length. The central $20 \times 20$ square had white color, and the remaining squares were black, so that the robot could determine whether or not it was close to a boundary.

The fitness of the evolving robotic brains was incremented by one every time the robot visited a square it had not previously visited, and the final fitness measure was taken as the fraction of squares visited.

An important issue in ER is the fact that an EA will try to exploit the particularities of the encountered situations as much as possible. Thus, in the evolution of wandering behavior, if a single starting position was used in all evaluations, the EA would quickly find a way to optimize the motion of the robot based on the given starting position. In order to avoid such problems, Miglino et al. evaluated each robotic brain 10 times, starting from a random location in the grid.

The authors report that the best evolved robotic brains managed to make the corresponding robot visit an average of slightly more than half of the available squares. The evolutionary process exhibited three clear stages, during which the maximum fitness was essentially constant. The stages corresponded to different levels of complexity. For example, in the first stage, the evolved neural networks made no use of the memory unit, whereas in the second stage they did.

The neural networks obtained in the simulations where then implemented in actual Lego robots. While the behaviors of the actual robots were similar to

those of the simulated ones, there were some clear differences, some of which were attributed to noise (absent in the simulations). Thus, some simulations were made with noise added, and it was found that a closer correspondence between the behavior of real robots and simulated robots could be found if the latter were evolved at intermediate levels of noise.

### 4.3.2 Box-pushing

Simple *Box-pushing* can be used as a metaphor for behaviors needed, for instance, in transportation robots, and it has been investigated by several authors, e.g. [20], [40], [39], [49]. The results from two such investigations will now be discussed briefly.

#### Single-robot box pushing

Lee et al. [20], evolved box-pushing in simulations, and transferred their results onto a Khepera robot. In order to reduce the differences between simulations and real-world implementations of box-pushing, the authors used the interesting approach of sampling the *actual* sensors on the Khepera, storing the results in lookup tables that were then used in the simulations. In the simulations, the task of the robot was to push a small box towards a light source.

In fact, the simulations reported in [20] involved the evolution of organization of two behaviors, namely *box-pushing* and *box-side-circling*. The latter behavior was used in order to place the robot in the correct direction, so that it could use the *box-pushing* to move the box toward the light source. Here, however, only the box-pushing behavior will be discussed, behavioral organization being deferred to Chapter 6.

Lee et al. used GP to evolve tree-like structures achieving box-pushing, using the readings of the 8 IR sensors on the Khepera as input. The authors used the measure

$$e = \sum_{t=1}^{T} \alpha\,(1 - s(t)) + \beta\,(1 - v(t)) + \gamma w(t), \tag{4.5}$$

where $s(t)$ denotes the average of the normalized activations of the two front sensors (sensors 2 and 3) on the (simulated) Khepera robot, $v(t)$ the normalized forward speed of the wheels, and $w(t)$ the normalized speed difference between the two wheels. Note that the measure $e$ should be minimized, and is thus an error measure rather than a fitness measure in the traditional sense. However, an EA can of course easily be modified to strive to minimize an objective function. The measure $e$ penalizes low activation of the front sensors (i.e. loosing the object), low speed, and curved motion.

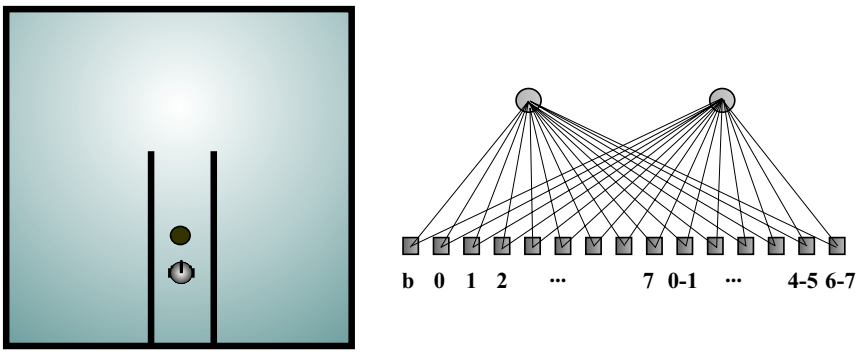


**Figure 4.7:** *The left panel shows a schematic view of the environment used by Sprinkhuizen et al. [39] in the evolution of box-pushing behavior. Note the light source placed at the end of the corridor. The right panel shows the structure of the FFNNs used.*

The EA used by Lee et al. made use of subpopulations in order to maintain the diversity of the population, and thus to avoid costly computations of almost identical individuals. Successful box-pushing behavior was achieved in 50 generations, using a total population size of 100.

Sprinkhuizen-Kuyper et al. [39] evolved box-pushing in simulations, using a Khepera simulator involving a slightly more complex environment (see the left panel of Fig. 4.7) containing walls. In this case a simple, one-layer, FFNN was used to represent the brain of the simulated robots. However, in addition to the 8 input signals provided by the IR sensors and a bias signal, the authors added six simple **edge detectors**, consisting simply of the differences between the readings of adjacent sensors, as shown in the right panel of Fig. 4.7. All 15 input signals were connected directly to the two outputs, which, in turn, provided the motor signals. Sprinkhuizen-Kuyper et al. noted that the fitness of an individual can be defined in several different ways, and the concepts of global, local, internal, and external fitness measures were employed. A **global fitness measure** was defined as one only taking into account the difference between the final state and the starting state of the robot, in a given evaluation, whereas a **local fitness measure** assesses a robotic brain based on its performance at each time step. An **internal fitness measure** was defined as one only based on the information available to the robot, through its sensors, whereas an **external fitness measure** is based on information that is not directly available to the robot, such as e.g. its position. The authors defined four fitness measures, using all possible combinations, i.e. global external, global internal, local external, and local internal. For example, the global external fitness measure was defined as

$$f_{\mathrm{GE}} = d(B_T, B_0) - \frac{1}{2} d(B_T, R_T), \tag{4.6}$$

where $d(B_T, B_0)$ denotes the difference in position of the box between the final

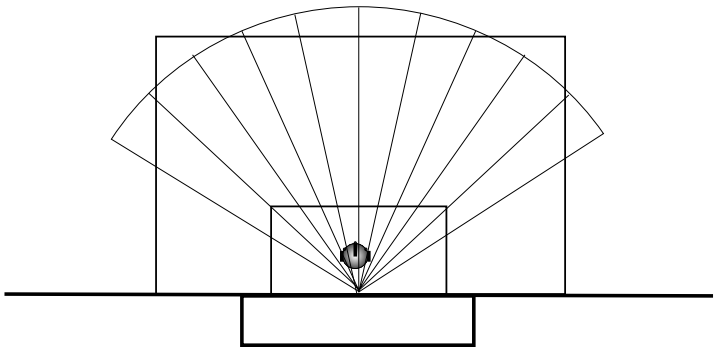

**Figure 4.8:** *A schematic illustration of the 10 different attack directions used by Lazarus and Hu in the evolution of a soccer goalkeeper [19].*

state (at time $T$) and the initial state, and $d(B_T, R_T)$ is the difference in position between the box and the robot at the final state. The second term was introduced as a penalty for robots that did not keep pushing the box until the end of the simulation. The local external fitness measure instead used a sum of terms similar to those given in Eq. (4.6). The global internal fitness measure required the introduction of lights, so that the robot could distinguish good locations from bad ones. Finally, the local internal fitness measure was taken as that used by Lee et al. [20].

Robotic brains were evolved using each of the four fitness measures. Next, each robotic brain was tested for its performance based on the other three fitness measures. It was found that the best performance was obtained with the global external fitness measure, indicating that the best performance is obtained when the EA is allowed to explore its search space quite freely, without risking a penalty for occasional bad moves. The local internal fitness measure did not do very well, however, and its failure was attributed to the fact that this fitness measure would reward robots pushing against a wall (with sliding wheels). The final results were succesfully implemented in a Khepera robot.

### 4.3.3 Behaviors for robot soccer

In recent years, there has been a great interest in robot soccer, manifested in events such as the Robocup tournament (see www.robocup.org). Robot soccer, or indeed any two-person (or two-team) game for robots, leads to interesting questions concerning not only evolution of behaviors in general, but also multi-robot coordination, co-evolution etc. Research on this topic has resulted in a very large number of publications, and only a few brief examples will be given here. The interested reader is referred to www.robocup.org or www.fira.net for further information.

The ultimate aim of Robocup is to generate a team of (humanoid) robotic

football players, capable of beating the best human team, a seemingly distant goal. In addition to humanoid robot soccer, wheeled robots and simulated robots have been used as well in the framework of Robocup. In [19], Lazarus and Hu evolved goalkeeper behavior in simulations using GP. In the simulations, the function set included the standard operators of addition (`add`), multiplication (`mult`) etc., as well as problem-specific operators such as e.g. `kick` and `catch`. The terminal set included e.g. the direction to the ball and the distance to the ball. Each individual goalkeeper robot was tested against attacks from 10 different directions, as shown schematically in Fig. 4.8. An attack consisted of the attacker kicking the ball in a straight line toward the goal. The authors attempted to evolve active goalkeepers, that would not only catch an incoming ball, but would also attempt actively to move toward the ball and intercept it. To this end, a fitness measure involving five parts, namely ball saving, self-localization, ball localization, movements, and positioning was used. When using a multimodal fitness measure, the problem of weighing different fitness cases against each other always appears. In [19], it turned out that the best results were found using equal weights.

In [46], simple ball-following behavior was evolved, also using GP and employing several (in this case, 8) different trials in order to evolve robust solutions. Luke et al. [21] used co-evolution to evolve a simulated robot soccer team.

### 4.3.4 Motion of a robotic arm

As mentioned earlier, BBR (and therefore ER) is mostly concerned with autonomous robots. However, ER has also been used in connection with stationary robots such as robotic arms. An example of such an application is reported in [26], where obstacle avoidance was evolved for an OSCAR-6 robotic arm. The authors noted that, while basic manipulator-eye coordination had been obtained using neural networks trained in a supervised manner (using, for example, backpropagation), such approaches are mainly useful in structured environments without obstacles. In supervised training, the neural network must generally be given feedback for each input-output mapping, something which is difficult to provide e.g. in obstacle-filled environments. Thus, Moriarty and Miikkulainen [26] used an EA to evolve neural networks for controlling the robotic arm, guiding the search by means of a single fitness measure at the end of each evaluation, rather than providing the neural network with a detailed performance measure for every single motion carried out.

The task of approaching a target was divided into two phases, an initial approach phase phase, which brought the end effector to the vicinity of the target, and a final approach phase which used smaller movements to reach within grasping distance of the target. Based on this division, the authors devised a

control system consisting of two neural networks, a primary network for the initial approach phase, and a secondary network for the final approach phase. Each network was represented as a fully connected FFNN, with 9 input units, a single hidden layer with 16 neurons, and 7 output neurons. The 9 input units consisted of 6 proximity sensors, as well as the $x$, $y$, and $z$ components of the distance between the end effector and the target. Six of the output units were used for determining the direction of rotation and magnitude of rotation for three joints in the robotic arm. The output units determining the magnitude of rotation were scaled differently in the primary and secondary networks, in such a way that the motion of the secondary networks used smaller steps. The seventh output unit was used to override the output of the other six outputs, and thus to stop the arm, in emergency situations.

The training of the arms was made by starting the arm with random joint positions, and a random target position, selected from a set of 400 pre-generated positions. An additional 50 target positions were generated for use during testing. In addition, obstacles were placed in one of 12 imaginary boxes located in the path of robotic arm. The motion was executed as a series of discrete moves, and simulations were terminated e.g. if a collision occurred. Fitness was assigned as the percentage of the initial distance (between the position of the end effector and the target) covered during the motion.

The results of the EA showed that the combination of primary and secondary networks could, in fact, control the arm to within industry standards, i.e. with a deviation of one cm or less between the actual and desired end effector positions. Furthermore, using the 50 test cases, it was shown that the evolved solutions were robust; Obstacles were hit only in around 2% of the test cases.

## 4.4 Evolving behaviors with ERSim

There are several program packages available for evolving robotic behaviors. An example is the `UFLib` package [42], which is used mainly for evolutionary optimization of the *selection* (for activation) of behaviors. In this course, however, we will used a slightly simpler simulator, called `ERSim`, which is based on the `ARSim` autonomous robot simulator introduced in Chapter 1 and Appendix A.

### 4.4.1 A simple example

As a very simple, illustrative example of the use of `ERSim`, consider the evolution of fast *straight-line navigation*. Specifically, consider a robotic brain which simply assigns constant values to the motor signals used as input to the motors

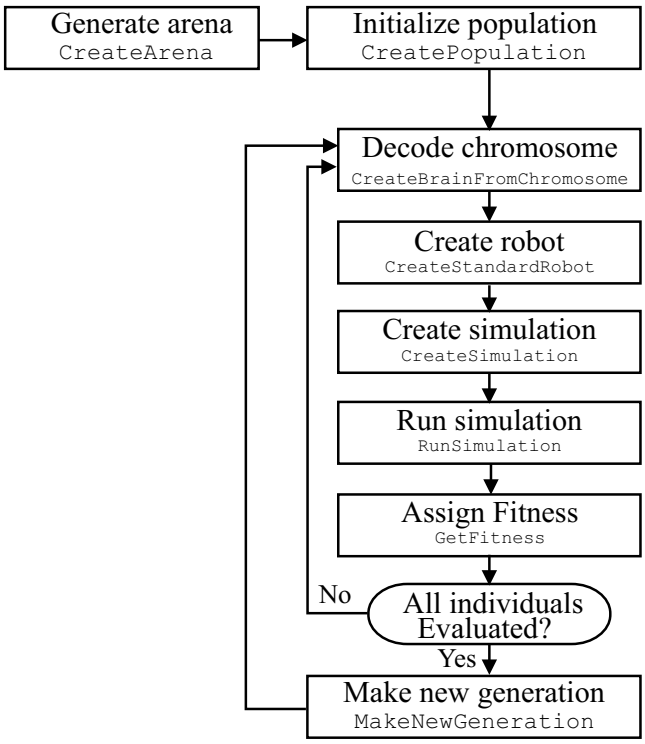


**Figure 4.9:** *The flow of an ERSim simulation. In each box, the bottom row shows the name of the corresponding Matlab function.*

of the (differentially steered) robot, i.e.

$$s_L = s_1, \tag{4.7}$$

$$s_R = s_2, \tag{4.8}$$

where $s_1$ and $s_2$ are constants. Clearly, achieving straight-line navigation is trivial: All that needs to be done is to set $s_1 = s_2 = s_0$, for some value of $s_0$. In order to maximize speed, the value of $s_0$ should, of course, be chosen as large as possible. Even though the problem is trivial, it is sufficient for illustrating the `ERSim` program and the example has therefore been included in the default version of `ERSim`.

The general flow of the program is illustrated in Fig. 4.9 (for a more detailed description, see Appendix B). In the straight-line navigation example, the chromosomes consist of two real numbers, in the range $[0, 1]$. In the decoding procedure the numbers are converted to constants in the range $[-1, 1]$, which are then used as motor signals. The robot starts at rest in $(x, y) = (0, 0)$, heading in the positive $x$-direction. The fitness measure is simply taken as the distance moved in the $x$-direction.

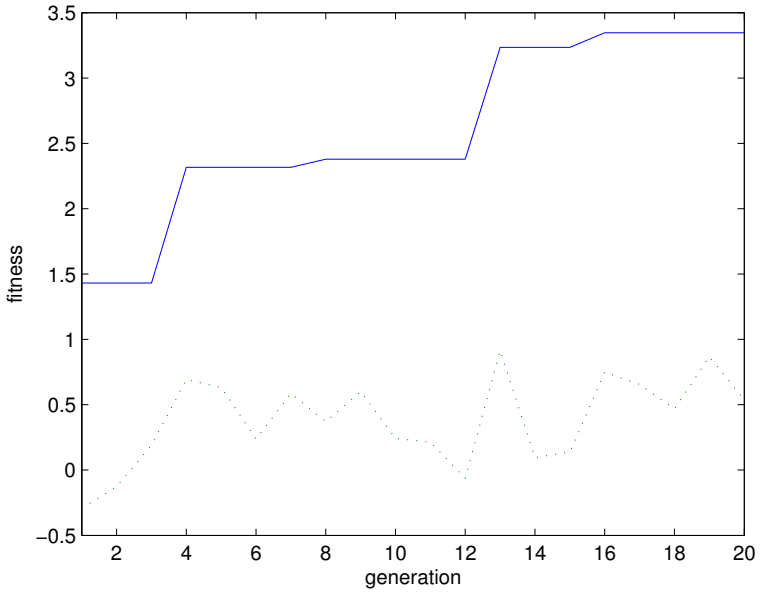


**Figure 4.10:** *The maximum (solid curve) and average fitness values obtained for the first 20 generations in a run of the straight-line navigation example.*

The two curves in Fig. 4.10 show the maximum and average fitness values for the first 20 generations of a run with a population size of 10. Note that the average fitness oscillates quite wildly, since the mutation rate was set to a very high value (0.50). The best individual found near-equal values, close to 1, of the two motor signals.