TRENDS IN EVOLUTIONARY ROBOTICS*

Lisa A. Meeden
Computer Science Program
Swarthmore College
Swarthmore, PA
USA
meeden@cs.swarthmore.edu

Deepak Kumar
Department of Math & Computer Science
Bryn Mawr College
Bryn Mawr, PA
USA
dkumar@brynmawr.edu

Abstract

A review is given on the use of evolutionary techniques for the automatic design of adaptive robots. The focus is on methods which use neural networks and have been tested on actual physical robots. The chapter also examines the role of simulation and the use of domain knowledge in the evolutionary process. It concludes with some predictions about future directions in robotics.

*Appeared in *Soft Computing for Intelligent Robotic Systems*, edited by L.C. Jain and T. Fukuda, Physica-Verlag, New York, NY, 215–233, 1998.

1 Introduction

To be truly useful, robots must be adaptive. They should have a collection of basic abilities that can be brought to bear in tackling a variety of tasks in a wide range of environments. These fundamental abilities might include navigation to a goal location, obstacle avoidance, object recognition, and object manipulation. However, to date, this desired level of adaptability has not been realized. Instead, robots have primarily been successful when deployed in constrained environments to perform deterministic tasks. The result has been that robots have had very limited, task-specific competencies which do not generalize to new situations.

The main advantage of imposing strict constraints within a robotic domain is to enable the use of a predictive model. This implies that a particular set of sensor readings can be accurately translated into a representation of the current state of the world. Furthermore, the result of doing a particular action can be known in advance of actually executing it. This allows a robot to plan its behavior.

Even working within these simplified conditions, a third of the cost of an industrial robotic system is its programming [43]. Yet, the most exciting potential applications involve much more complex and dynamic environments than have typically been attempted so far (for example outdoor, subsea, and other planet environments). We should expect that the burden on the human designer of control software will only increase as we try to move towards these more advanced applications. One hope is that the uncertainty and variability that occur in physical sensors and actuators might lessen as our hardware technology improves, thus alleviating the programming problems that arise. However, some experiments have shown that using higher resolution sensors introduces more variation, not less as one might expect [42].

The use of evolutionary computation is one of the most promising avenues for overcoming the bottleneck of the human engineer in the robot design process. In fact, it has been proposed that an evolutionary approach to the design of robots will eventually superceed design by hand [5]. The fundamental idea of this approach is to maintain a population of possible robot control architectures. The initial population is typically a collection of randomly configured architectures. Each architecture is evaluated according to an objective fitness measure and the better the robot performs using that architecture the more offspring it is allowed to produce in the next generation of the population. Over a number of generations, the fitness of the population increases and successful architectures are created. A human engineer must develop the evolutionary framework, but the actual design of the robotic systems is then automatically generated.

The most accurate fitness measure for a potential architecture is to allow it to control the actual robot operating in the physical world. Yet this can be a painstakingly slow process. Consider that each action can require a second of time to execute and that each fitness measure typically involves hundreds of actions. With a sizable population, the processing of a single generation may require an hour's time. When hundreds of generations are necessary to achieve reasonable results, one run of the evolutionary process may require several days. Two techniques have been employed to speed up the evolutionary process: the use of simulations for faster fitness evaluations and the seeding of the initial population with domain knowledge to decrease the overall number of generations required.

A central question when adopting the evolutionary computation approach is: What type

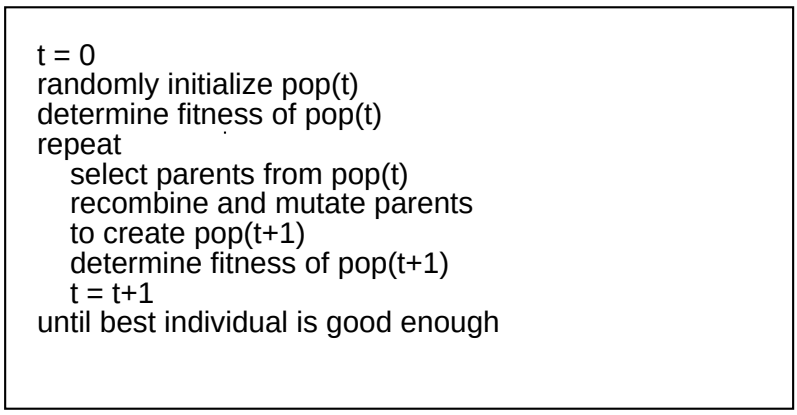

Figure 1: General Evolutionary Computation Algorithm

of robot control architecture should be evolved? There are a number of options: high-level code [40], machine code [37], parameter settings for a hand-designed system [39], situation-action rules [7], and entire rule-based strategies [15]. Perhaps the most innovative direction, however, is the combination of evolutionary computation with artificial neural networks. Neural networks allow the evolutionary process to operate at a very low level, placing minimal constraints on the possible solutions. When a higher-level architecture is used designer bias is more prevalent [5]. In addition, neural networks are robust, noise tolerant and can be used for local learning starting from the points discovered by the global evolutionary process [36].

The remainder of this chapter is organized as follows. In Section 2, further background is given on evolutionary computation and neural networks. In Section 3, some specific research projects combining evolutionary computation and neural networks to control robots are examined. In Section 4, the current debate on the role of simulations is reviewed. In Section 5, the ways in which domain knowledge has been incorporated in evolutionary robotic systems is discussed. Finally in Section 6, speculations about future trends in robotics are explored.

2 Background

2.1 Evolutionary Computation

The majority of the current implementations of evolutionary computation descend from three branches of study: genetic algorithms, evolutionary programming, and evolution strategies [1]. Of these, the genetic algorithm approach developed by Holland, is the basis for most evolutionary robotics applications [22, 32]. Other related techniques that arose out of genetic algorithms are classifier systems [13] and genetic programming [26].

All evolutionary computation methods attempt to mimic the process of natural evolution. The general structure of an evolutionary computation algorithm is shown in Figure 1 and depends on three operators: selection, recombination, and mutation. Selection is usually implemented as a probabilistic process using the relative fitness of an individual to determine its selection probability. In this way, fitter individuals are more likely to participate in producing the next generation. Recombination is the means of mixing the genetic material

of two parents to produce an offspring. Mutation creates random alterations in the genetic material of an offspring.

Consider the effects of each of these operators. Selection alone tends to fill the population with copies of the best individual from the initial population. Mutation alone induces a random walk through the search space. Selection and mutation (without recombination) creates a parallel, noise-tolerant, hill-climbing search. Selection and recombination (without mutation) tends to cause the process to converge on good, but sub-optimal solutions. Together the three operators form a flexible, robust, global search method.

2.2 Artificial Neural Networks

One of the most common artificial neural architectures is a layered, feedforward network. Each unit in one layer is linked by weighted connections to each unit in the next layer. Various amount of activation are applied to the units in the initial layer, representing the input to the network. This activation then flows across the connections to higher layers, with the weights mediating the amount of activation that is passed on to successive units. The final pattern of activation present on the last layer is considered to be the output produced.

A supervised learning algorithm such as back-propagation can be repeatedly applied to adjust the weights of a network enabling it to learn to associate arbitrary pairs of input and output patterns [41]. If the input pattern is interpreted as representing sensory information and the output pattern as an action, a network can be used to control a robot. As a result of this training process, the network learns to recode the incoming sensory patterns into new patterns at the intermediate layers so that the appropriate output action is produced. These intermediate layers are termed hidden because they do not have direct access to the environment.

One drawback of the standard feedforward architecture is that it is difficult to represent time in an efficient manner. Certainly for controlling a robot timing information is crucial. A common method of accommodating time has been to represent time as space. This entails extending the input layer to act as a buffer of the past. A better approach is to deal with time implicitly rather than explicitly by using a recurrent architecture. One such recurrent architecture, called an Elman network, allows every unit in a hidden layer to have weighted connections to every other unit in the layer, including itself [9]. This gives the network a memory of its own internal recodings of its past sensory inputs. A more radical recurrent architecture abandons a layered topology and allows units to connect to one another in arbitrary ways.

In Section 3, we will see examples of feedforward, Elman, and arbitrary network architectures being employed to control robots.

2.3 Combining Evolutionary Computation and Artificial Neural Networks

In applying evolutionary computation to neural networks for the purpose of robot control, two main methods have been used. The first method is to fix the topology of a network architecture and to then use evolutionary computation to determine the weights. Unlike back-propagation, which is a gradient descent procedure for finding an appropriate set of weights, evolutionary computation should avoid getting stuck in local minima. The second method is to allow evolutionary computation to actually determine aspects of the network's

topology. The weights may then be set through a separate learning algorithm or can also be determined by evolutionary computation.

In Section 3, we will see examples of both of these approaches, evolved weights and evolved topologies, being used to develop robot controllers.

3 Examples of Evolved Neural Network Robot Controllers

Conducting evolution on physical robots is a time consuming process, and as a result most of the applications attempted so far have been fairly modest in scope. Often, for reasons of practicality, the robots used are quite small. This allows the task environment to be set up on a desktop with the robot tethered to a computer for data collection and tethered to an electrical outlet for power. One popular platform for conducting evolutionary experiments is the Khepera robot [33]. Khepera is circular in shape and miniature (diameter 55 mm, height 30 mm, and weight 70g). It has two DC motors which power two wheels. It's standard sensory apparatus consists of eight infra-red proximity sensors.

Of the five applications described below, the Khepera was used in two: battery recharging and trash collection. Another small robot, called the Gantry, was used for locating visual targets. A miniature car, called carbot, was used for light seeking and avoiding. Only the driving task involves a full size robot, a car called NAVLAB, which is part of an ongoing project at Carneigie Mellon University [38, 25].

3.1 Battery Recharging [12]

The environment was an empty rectangular arena with a gray floor. A light emitter was placed over one corner, and under this light a circular patch of the floor was painted black. This represented the battery recharging area. When the robot happened to pass over this black area its virtual battery would be instantly recharged. The Khepera robot was equipped with some additional sensors for this task. First it had three light sensors, two of which gathered ambient light, and one which pointed down at the floor. Second it had a virtual battery level sensor.

The goal of the evolutionary process was to determine an appropriate set of weights for a fixed Elman-style recurrent network with twelve input units for the sensors, five hidden units, and two output units for the motors. The process was run continuously for ten days with each generation lasting approximately three quarters of an hour.

Each individual set of weights was evaluated by a very simple fitness function that consisted of two components: one to maximize speed and the other to avoid the walls. When the robot was in the recharging area, fitness was defined to be 0. Notice that there is no explicit reward for recharging the battery; in fact in terms of direct fitness it is detrimental to spend time in the recharging area. However, there is an implicit benefit to recharging. A fully charged battery allowed the robot to move for 50 time steps, and each individual was allowed an upper limit of 150 time steps for evaluation. Therefore by recharging the battery, the robot could gain more time to accumulate fitness.

Analysis of the resulting behavior revealed that the robot exhibited very different strategies depending on its battery level. When the battery level dropped to about one third full charge, the robot began executing a trajectory that led it to the recharging area. Otherwise,

it moved at nearly full speed along a slightly bended trajectory that avoided the recharging area.

Although a very general fitness function was used, the evolved robot controller was able to locate the battery recharging area and to instigate timely homing maneuvers when the battery level was low. To extend this generality even further, Floreano and Mondada suggest that it would be interesting to redo this experiment but eliminate the fitness function entirely and simply select those individuals that live longer. This would make the artificial evolution process more similar to natural evolution.

3.2 Trash Collection [35]

In order to pickup trash, the Khepera robot was equipped with a gripper module. The environment was a rectangular arena containing five pieces of randomly distributed trash (white cardboard cylinders). The robot's task was to find these targets, pick them up, carry them to the boundary of the arena, and drop them outside.

Nolfi compared five different architectures, the best of which was able to determine how to modularize the task. It was a two-layer feedforward network where the value of each motor output depended on a competition between two separate modules. The number of available modules was fixed at two per output node, but the interactions between the modules was determined by the adapted weights.

The fitness measure primarily depended on how many targets were successfully released outside the arena. To help bootstrap the process, the fitness measure also included a component to reward a robot's ability to simply pickup targets. It proved difficult for the robot to learn how to react to new targets when it was already carrying a target. To alleviate this problem, the training experience was manipulated to make this type of occurrence more frequent.

During the training process, the controllers were evaluated in a simulator. In spite of this, each evolution run still required approximately ten hours. After training was complete, the resulting control systems were downloaded into the physical robot and tested in the real environment.

To succeed at this task, the ability to distinguish between walls and targets is of crucial importance. Some errors made by the robots revealed that this was not a simple distinction, such as attempting to grasp walls or releasing a target over another target. The emergent modular architecture created modules specifically tuned to making this distinction under confusing sensory situations. The capacity to produce sharp switches in strategy in response to fine-grained sensory differences is a key to complex behavior.

3.3 Locating Visual Targets [6]

Several visual tasks were tried, including locating a large static target, locating a smaller static target, tracking a moving target, and locating a triangle target in the presence of a competing rectangle target. For these experiments a specialized piece of robotic equipment was developed called the Gantry robot. Instead of wheels, the robot is suspended from a gantry frame that allows translational movement in the X and Y directions. The robot is 150mm in diameter and can rotate. It is equipped with a camera pointed down at a mirror which is inclined at 45 degrees as well as several bumper sensors.

In this case, the evolutionary algorithm searched for both a network architecture and a visual morphology. Each network had a fixed number of input nodes (for sensors) and output nodes (for motors), but the number of hidden nodes and the number and type of connections (either excitatory or inhibitory) was variable. Connections were allowed to be recurrent between any layer. Furthermore, rather than feeding a raw camera image to the controller, they allowed the method of sampling this image to be evolved along with the network. They achieved this by designating a set of possible receptive fields within the image.

The visual tasks involved finding a light colored target within the dark colored environment. The fitness measure was a function of the distance of the robot from the desired target—the closer the robot to the target the higher its fitness. For each possible architecture and visual morphology, the robot was given four trials starting from the same position but using different orientations. The final fitness was the worst score from these four trials. This allowed them to terminate trials as soon as they bettered a previous trial and thus improved the evaluation time.

Starting the evolutionary process from initially random populations proved to be too slow. Instead they selected one of the members of this initial random population that displayed "interesting" behavior and then cloned it to create a new population. Working with a converged population rather than a random population has been termed Species Adaptation Genetic Algorithm or SAGA [18]. From this starting point, the evolution run was able to develop good solutions within 10 to 15 generations with each generation taking about one and a half hours.

First the robot was evolved to successfully locate large (150cm wide) static targets. The final population from this experiment was used as the starting point for the next experiment of locating smaller (22cm wide) static targets. In less than 10 generations the population adapted successfully to the harder task. Again his new population was recycled and used as the starting point for the even harder task of tracking a moving target. Working in this incremental fashion it should be possible to gradually build up quite complex behaviors (see also [14]). Also allowing the morphology of the robot to be developed along with the controller may simplify the task solution considerably.

3.4 Seeking and Avoiding Light [29]

Carbot is a modified toy car which is approximately 15cm wide and 23cm long and is equipped with two light sensors and several bumper sensors. It was placed in a rectangular arena with a light in one corner. Its task was to constantly keep moving, avoid the walls, and respond appropriately to a light goal. When the goal was positive, carbot had to seek out the light until a maximum light reading was obtained. Once this was accomplished the goal automatically switched to a negative value, indicating that carbot had to avoid the light until a minimum light reading was obtained. The goal varied in this periodic manner throughout the task, seeking was immediately followed by avoiding and so on.

A local and a global method of reinforcement learning were compared for training carbot at this task—a special form of back-propagaion and an evolutionary algorithm. The topology of the network was fixed to be an Elman network with seven inputs for the sensors and goal, five hidden units, and four output units for the motor settings. The aim of both methods was to find an appropriate set of weights. The adaptation process was conducted on a simulation

and the best architectures were tested on the actual robot.

Statistical analyses revealed several quantitative differences between the two learning methods. The back-propagation algorithm out-performed the evolutionary algorithm in the original task. However, the evolutionary algorithm out-performed the back-propagantion algorithm when the task difficulty was increased either by removing the explicit goal (but keeping the periodic structure) or by removing immediate reinforcement feedback. Perhaps even more interesting were the qualitative differences in the behaviors produced by the two methods. Being a local method, back-propagation was more sensitive to the moment-to-moment changes in the environment and thus used the explicit goals to develop unique strategies tuned to each goal. In fact when no explicit goal was present, back-propagation trained networks sometimes created their own goal-like units in the hidden layer. In comparison, the evolutionary algorithm tended to develop a single overall strategy that was applicable to both goals. More importantly, the evolutionary algorithm's ability to find good strategies was quite robust across the experimental variations.

The respective strengths and weaknesses of these two methods are clearly complementary, suggesting that some hybrid of the two could be the most effective method. Because the evolutionary algorithm globally samples the entire space of alternative solutions while back-propagation locally searches the immediate neighborhood of a solution, the most straightforward form of hybrid would be to allow the evolutionary algorithm to find a good starting point in the weight space and then use back-propagation to do the fine tuning. As in nature, the global evolutionary method can determine a good gross solution which the local learning method can then adjust to the current environmental conditions.

3.5 Driving [3]

NAVLAB is an autonomous land vehicle that has been operated in a wide variety of domains including dirt roads, bike paths, two-lane suburban neighborhood streets, and divided highways. A NAVLAB controller must determine an appropriate steering angle when given a video image from a camera mounted on the front of the car. Unlike the previous tasks, for this task there is a clear "right" answer for every situation as determined by human drivers.

A number of different network architectures have been explored for solving this driving task, the most successful of which is a three-layer, feedforward topology. It has a 2-D input retina for the video images, a small hidden layer, and a gaussian representation of the steering angle across thirty output units ranging from "sharp left" to "straight ahead" to "sharp right". Previously back-propagation had been used to determine the weights for these network controllers. Baluja set out to discover whether evolutionary computation could develop better controllers for this task.

Suppose that a "maximal" network describes the maximum connectivity of the networks to be evolved. Then through evolution, different topologies can be tested by selecting which of these possible connections to enable. If a connection is present, then a weight for it is determined as well. The maximal network for these experiments contained a 15x16 input retina fully connected to a five unit hidden layer fully connected to a thirty unit output layer, all strictly feedforward.

In order to reduce search times, Baluja created a novel evolutionary method called Population-Based Incremental Learning (PBIL). This algorithm requires that individuals

in the population be represented with a binary alphabet (network weights can be represented as binary numbers if restricted to a range of values). The goal of PBIL is to produce a real-valued probability vector which can be considered a prototype for highly fit individuals in the population. By sampling this single probability vector an entire population can be stochastically produced and then tested. Based on the test results the values are adjusted to push the probability vector towards the best individual and away from the worst individual. In addition, a mutation operator is used to update the probability vector directly, by shifting each value a small positive or negative amount. Unlike a genetic algorithm, PBIL does not use any recombination operator.

To test possible networks for the driving task, a training set of 1000 images and correct steering angles were collected and saved. From this set, 100 examples are randomly selected for each network evaluation. The network which obtained the lowest error was designated best for that generation and the network with the highest error the worst.

The final networks evolved using the PBIL method kept only about half of the possible connections allowed in the maximal network. When compared with maximal networks trained with back-propagation, the PBIL networks showed a 13% reduction in error. Thus PBIL produced more space efficient and more accurate controllers. However this gain has a cost; to complete an entire PBIL run requires about an hour while a back-propagation run requires only a few minutes of processing time.

3.6 Summary of Examples

With the battery recharging task, Floreano and Mondada demonstrated that interesting, complex behavior can be obtained without an overly explicit fitness function. They further suggested that rather than employing a human engineered measure of some kind, ultimately the best measure may be the most general one—survival of the fittest.

In contrast to this call for more implicit fitness measures, Harvey, Husbands, and Cliff argued that complex behavior may best be obtained through a set of well designed incrementally harder fitness tests. Using related visual location tasks, they showed that by beginning each evolution run from a converged population rather than a random one, a faster more focused exploration was produced.

Another important aspect of Harvey, Husbands, and Cliff's experiments was that the visual morphology of the robot was evolved along with the controller. Because the human sensory apparatus is quite different from a robot's sensory apparatus, the way in which a human might consider processing a visual image will probably not translate well into a robot. Instead, by allowing the evolutionary process to operate on the sensory apparatus, a more efficient robot-based solution can emerge.

A similar conclusion was drawn by Nolfi with respect to modularization. Through a comparison of a number of network architectures for a trash collection task, Nolfi found that providing a network with modularization options was extremely beneficial. Again because of the differences between humans and robots in sensory capabilities, the way in which a human might subdivide a task may not translate well into a robot. Allowing the evolutionary process to consider how to break up a task can lead to simpler robot-centered solutions.

In the light seeking and avoiding task, Meeden demonstrated that an evolutionary algorithm is a robust method for determining a good gross solution to a robot control problem.

She suggested that such a solution can be improved and fine tuned through additional training with a local learning method such as back-propagation. A hybrid of this kind can be robust across large environmental changes and yet sensitive to subtle features.

Finally in Baluja's studies, a fast, new evolutionary computation method called PBIL was employed to create controllers for driving. Both the topology and the weights of the networks were determined by the evolutionary process. The results demonstrated that automatically designed controllers out-performed hand-designed controllers.

Some questions arise from these studies. How large a role should the human designer play in shaping the robot's behavior through the fitness function? Should our models of evolution try to be more true to natural evolution? Or should we as engineers try to influence the evolutionary process more directly?

A related question that also emerges from these studies is: What aspects of the robot control system should be manipulated by the evolutionary process—the parameters, the architecture, or the robot itself? It appears that the more features that are accessible to the evolutionary process then the more successful the adapted controllers will be.

4 Simulation

> There is currently a hot debate among people trying to understand and reproduce intelligent agents, that could be stated as follows: Is the simulation a powerful enough tool to draw sound conclusions, or should a theory or an approach be tested on a real agent, i.e. a robot? [11]
>
> Simulation in robotics, control theory, and AI has mostly been a complete waste of time. Of course there are certain cases in which simulation is inevitable ... what is at issue is whether results "demonstrated" using simulation *only* should be accorded worth. We think not. [27]
>
> However, it does appear that simulations are not quite the dead-end some had suggested. For simpler cases at least it has been shown that they can be made accurate enough. Their attractive qualities of speed and ease of data collection can then be made use of. [24]
>
> It is frankly easier to use robots situated in the real world than it is to try to build some all encompassing super-simulation. [23]

What makes a robot distinct from any other artificial intelligence project is that one must actually deal with hardware and the intrinsic limitations that all physical sensing and acting systems have [20]. By using a simulation, one can sidestep these difficult hardware interactions completely, and this is what is at issue in the simulation debate. For example, the term "robot" is often used loosely to refer to a simulated "agent" or "animat" which may not have any physical counterpart upon which it is modeled [42]. In fact some such simulated "robots" could never be implemented in the real world because they depend on non-existent "sensors" to provide object-level information about the environment. The danger is that simulations simplify the learning problem too much by making the environment and the

robot clean and predictable. Thus there is no guarantee that results obtained in simulation will transfer to the noisy real world. This is termed the "correspondence problem".

Despite the difficulties simulations present they are still very attractive primarily because of their speed. An evolutionary experiment that takes several days on a physical robot may only require several hours on a simulated robot. This ability to obtain results quickly facilitates a more open-ended exploration of robot control ideas. In addition, simulated results are more easily collected, analyzed, and reproduced.

To lessen the correspondence problem, every effort should be made to keep simulations in close step with reality. Some specific suggestions have been made along this line: (1) base the simulation on carefully collected empirical data of a real physical robot; (2) add noise to the simulated sensory readings and the actuator outcomes; and (3) calibrate the simulation through tests on the real physical robot [19].

Calibration tests have provided interesting and somewhat contradictory results. In one case, neural network robot controllers adapted in simulation always performed better when tested in the real world than they had when tested in simulation [31]. The suspected cause of this improvement was that the physical robot's movements and sensor readings were not noisy in the same ways as the simulator's. The physical robot's experience was occasionally noisy while the simulator's experience was systematically noisy, and this was beneficial to learning. In another case, experiments were conducted to determine how the amount of simulated noise affects the correspondence between behavior evolved in simulation and then tested in simulation versus being tested in the real world [24]. Three noise levels were examined: no noise, observed noise, and double the observed noise. The results revealed that networks that evolved in an environment that is less noisy than the real world will behave more noisily in reality. Conversely, networks evolved in an environment that is noisier than the real world will behave less noisily in reality. Furthermore the correspondence between simulation and reality was maximized when the noise level of the simulation most closely matched reality.

In the first case additional noise appeared to be beneficial while in the second case it did not. Perhaps these contradictory results could be resolved with further experiments. It may be that twice the observed noise is too drastic a change to realize a benefit, whereas a smaller increment above observed noise would be helpful. This issue of the appropriate amount of noise is just one of the many open questions related to the use of simulations for adapting robots.

In doing evolutionary robotics, one has three options for evaluating possible control systems: use a physical robot, use a simulated robot, or use a hybrid of the two [36]. Using a physical robot is obviously the most desirable but may be too time consuming. Using a simulated robot leads to the correspondence problem. Using a hybrid approach, one can begin the evolutionary process on a simulated robot to quickly develop a high performing population. Assuming that the simulation was developed through close observation of the actual robot, this should provide a good starting point for the slower evolutionary process on the physical robot. Also a simulated robot can be used to quickly prune the set of possible experiments one may want to eventually conduct on a physical robot [8]. For these reasons, the hybrid approach to evolutionary robotics may offer the best compromise between speed and accuracy.

5 Domain Knowledge

The role of domain knowledge in evolutionary robotics can be viewed as a continuum. At one extreme, robots can have complete domain knowledge and require no ability to adapt, while at the other extreme robots can have no domain knowledge and be seen as tabula rasa systems. At a recent workshop on robot learning, most of the research presented incorporated a substantial amount of domain knowledge. [21]. Many of the presenters argued that to construct a successful robotic system, learning must be limited in use to portions of the system where the designer's knowledge is too sketchy to engineer a solution. Learning from scratch was seen as too inefficient for any problem of reasonable complexity.

A middle ground along this continuum is an approach that treats knowledge acquisition as a cooperative effort between the human engineer and the robot itself [17]. Grefenstette developed an evolutionary method known as SAMUEL, which stands for Strategy Acquisition Method Using Empirical Learning [16]. SAMUEL is an evolutionary process that develops entire behaviors which are defined as sets of rules. The rules are expressed in a high-level language to make it easy to incorporate existing knowledge and to make it easy to understand the results of learning. The initial population of behaviors are not random but consist of a variety of rule sets including human generated ones and automatically generated variants of these. By seeding the initial population, it is hoped that the search space will be usefully constrained thus leading to faster search times (refer back to Section 3.3 for another example of this). Another technique used to constrain the search space, is the creation of virtual sensors. For instance, rather than basing rules on sixteen raw sonar values, one could create four virtual sensors—left, forward, right, and backward—which combine the raw values in a meaningful way.

Although beginning with a significant amount of domain knowledge seems more practical there may be a disadvantage. By imposing our perspective on the learning problem we may actually make it harder. Nolfi argues that a designer views a robot task from an observer's point of view or a *distal* perspective, but a robot must solve the task in terms of its sensorimotor system or a *proximal* perspective [35]. There may be no simple mapping from the distal perspective to the proximal perspective. For instance, consider the trash collection task described in Section 3.2. One distal description of this task is the following:

1. Explore the environment avoiding walls.
2. Recognize trash and approach it for pickup.
3. Pickup trash.
4. Move towards a wall avoiding other trash.
5. Release trash over the wall.

Recall that the Khepera robot used in these experiments only had access to infra-red sensors. Using human vision (the distal perspective) it is a simple matter to distinguish a wall from trash, but using the robot's infra-red sensors (the proximal perspective) it is not. For the robot, these two objects can only be distinguished from a relatively small number of close positions. Thus a modular division of the task into distal subtasks such as "move towards a

wall avoiding other trash" is not viable because trash and walls appear the same except from a very local view.

It has been argued that artificial neural networks offer one of the most promising means for investigating robot control because they allow the task demands rather than the designer's biases to be the primary force in shaping the system's development [29]. Yet the designer still has an important role to play in the evolutionary process which includes: determining what aspects of the neural network will be operated on by evolution, the input and output representations, the robot's physical characteristics (which could also be manipulated by evolution), and the robot's environment. After this adaptive process has been set in motion and a successful control system has been produced, its method must be dissected to understand the underlying control principles. The use of evolutionary computation with neural networks inverts the classical order of problem solving in which a high-level understanding comes first and closely guides the search for algorithms [4]. Through the evolution of neural network controllers a solution to the task emerges which is not simply a product of the designer's understanding of the domain.

6 Future Trends in Robotics

Hans Moravec likens today's robots to simple invertebrates in the global evolutionary sense [34]. He predicts that in the next decade robots should improve to the level of reptiles and within 50 years to the level of mammals. One of the crucial impediments to modeling adaptive behavior in robots, besides the lack of modeling techniques, has been the size and speed of computers. Employing evolutionary techniques for developing neural network based controllers is computationally expensive. Significant progress has been made recently partly due to continuing exponential increase in the computational resources. As Moravec points out, the amount of computational power that a dollar can purchase has increased thousandfold every two decades since the beginning of the century. There has been a trillionfold decline in the cost of computation. If this trend continues, as seems to be the case at present, Moravec predicts that the computational power required for a humanlike robot would be available in a $10 million super computer before 2010 and in a $1000 personal computer by the year 2030.

Work on a humanlike robot has already begun at MIT with the COG project [2]. Their approach is to build a humanoid robot that develops and acts in the real world in the same way that humans develop and act. This human-inspired development plan has so far led to the incorporation of several behaviors: the arms have grasping, withdrawal, and reflexes like those of a child; the arms also have adaptive spring-like behavior; the arms follow smooth motion trajectories; the eyes have foveation behavior that can be used to coordinate hand-eye movements in reaching for objects; and the eyes and the head exhibit saccading motion and gaze control [10, 28]. Even though COG's performance today is below those of conventional robots, it is expected that the developmental approach will eventually pay off. Most of the models incorporated in Cog are based on biological models.

It has been suggested that the combination of artificial intelligence with evolutionary computation represents one of the most innovative research directions that may lead to the development of efficient, robust, and easy-to-use solutions to complex real-world problems [1]. Given the incredible computational power at hand, it is becoming increasingly attractive to experiment with evolutionary methods in robots. Onboard computers in mobile systems

are now powerful enough to run experiments in real-time, but simulations will probably still continue to play a role for some time. It is also becoming feasible to incorporate robotics into school and college curricula [30]. As robots become less expensive and more prevalent we should expect rapid innovations in the future.

References

[1] T. Back, U. Hammel, and H. Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17, 1997.

[2] R. Brooks and L. Stein. Building brains for bodies. *Autonomous Robots*, 1(1):7–25, 1994.

[3] S. Buluja. Evolution of an artificial neural network based autonomous land vehicle controller. *IEEE Transactions on Systems, Man, and Cybernetcs–Part B: Cybernetics*, 26(3):450–463, 1996.

[4] A. Clark. *Associative Engines: Connectionism, Concepts, and Representational Change*. MIT Press, Cambridge, MA., 1993.

[5] D. Cliff, I. Harvey, and P. Husbands. Explorations in evolutionary robotics. *Adaptive Behavior*, 2(1):73–110, 1993.

[6] D. Cliff, P. Husbands, J-A. Meyer, and S. Wilson, editors. *Seeing the Light: Artificial Evolution, Real Vision*, Cambridge, MA, 1994. MIT Press.

[7] M. Colombetti and M. Dorigo. Behavior analysis and training–a methodology for behavior engineering. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 26(3):365–380, 1997.

[8] M. Dorigo and M. Colombetti. Precis of robot shaping: An experiment in behavior engineering. *Adaptive Behavior*, 5(3/4):391–405, 1997.

[9] J. Elman. Finding structure in time. *Cognitive Science*, 14:179–212, 1990.

[10] C. Ferrell. Orientation behavior using registered topographic maps. In *From Animals to Animats: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, 1996.

[11] D. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In D. Cliff, P. Husbands, J-A. Meyer, and S. Wilson, editors, *From Animals to Animats: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 421–430, Cambridge, MA, 1994. MIT Press.

[12] D. Floreano and F. Mondada. Evolution of homing navigation in a real mobile robot. *IEEE Transactions of Systems, Man, and Cybernetics-Part B: Cybernetics*, 26(3):396–407, 1996.

[13] D. Goldberg. *Genetic Algorithms in Search Optimization, and Machine Learning.* Addison-Wesley Publishing Company, New York, NY, 1989.

[14] F. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5(3–4):317–342, 1997.

[15] J. Grefenstette. The evolution of strategies for multi-agent environments. *Adaptive Behavior*, 1(1):65–90, 1992.

[16] J. Grefenstette, C. Ramsey, and A. Schultz. Learning sequential decision rules using simulation models and competition. *Machine Learning*, 5(4):355–381, 1990.

[17] J. Grefenstette and A. Schultz. An evolutionary approach to learning in robots. Naval Research Laboratory Technical Report AIC-94-014, 1994.

[18] I. Harvey. Evolutionary robotics and SAGA: The case for hill crawling and tournament selection. In C. Langton, editor, *Artificial Life III*, pages 299–326. Addison Wesley, Reading, MA, 1993.

[19] I. Harvey, P. Husbands, and D. Cliff. Issues in evolutionary robotics. In J-A. Meyer, H. Roitblat, and S. Wilson, editors, *From Animals to Animats: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 364–373, Cambridge, MA, 1993. MIT Press.

[20] H. Hexmoor, D. KortenKamp, and I. Horswill. Software architectures for hardware agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9:147–156, 1997.

[21] H. Hexmoor and L. Meeden. Learning in autonomous robots: A summary of the 1996 robolearn workshop. *Knowledge Engineering Review*, 11(1), 1997.

[22] J. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI, 1975.

[23] P. Husbands, I. Harvey, D. Cliff, and G. Miller. Artificial evolution: A new path for artificial intelligence? *Brain and Cognition*, 34:130–159, 1997.

[24] N. Jakobi, P. Husbands, and I. Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In F. Moran, A. Moreno, J. Merelo, and P. Chacon, editors, *Advances in Artificial Life: Proceedings of the Thrid European Conference on Artificial Life*, pages 704–720. Springer-Verlag, 1995.

[25] T. Jochem and D. Pomerleau. Life in the fast lane: The evolution of an adaptive vehicle control system. *AI Magazine*, 17(2):11–50, 1996.

[26] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, 1992.

[27] Brady M. and H. Huosheng. Software and hardware architectures of advanced mobile robots for manufacturing. *Journal of Experimental and Theoretical Artificial Intelligence*, 9:257–276, 1997.

[28] M. Marjanovic, B. Scassellati, and M. Williamson. Self-taught visually guided pointing for a humanoid robot. In *From Animals to Animats: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, 1996.

[29] L. Meeden. An incremental approach to developing intelligent neural network controllers for robots. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 26(3):474–485, 1996.

[30] L. Meeden. Using robotics as an introduction to computer science. In J. Stewman, editor, *Proceedings of the Ninth Florida Artificial Intelligence Research Symposium*, pages 473–477, 1996.

[31] L. Meeden, G. McGraw, and D. Blank. Emergent control and planning in an autonomous vehicle. In *Proceedings of the Fifteenth Annual Meeting of the Cognitive Science Society*, pages 735–740, Hillsdale, NJ, 1993. Lawrence Erlbaum Associates.

[32] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1996.

[33] R. Mondada, E. Franzi, and P. Ienne. Mobile robot miniturization: A tool for investigation in control algorithms. In *Proceedings of the Thrid International Symposium on Experimental Robots*, Kytoto, Japan, 1993.

[34] H. Morvac. The universal robot. In C. Pickover, editor, *Visions of the Future: Art, Technology and Computing in the Twenty-First Century*, pages 65–73. St. Martin's Press, New York, NY, 1992.

[35] S. Nolfi. Using emergent modularity to develop control systems for mobile robots. *Adaptive Behavior*, 5(3–4):343–363, 1997.

[36] S. Nolfi, D. Floreano, O. Miglino, and F. Mondada. How to evolve autonomous robots: Different approaches in evolutionary robotics. In R. Brooks and P. Maes, editors, *Artificial Live IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 190–197, Cambridge, MA, 1994. MIT Press.

[37] P. Nordin and W. Banzhaf. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behavior*, 5(2):107–140, 1997.

[38] D. Pomerleau. *Neural Network Perception for Mobile Robot Guidance*. Kluwer, Norwell, MA, 1993.

[39] A. Ram, G. Boone, R. Arkin, and M. Pearce. Using genetic algorithms to learn reactive control parameters for autonomous robotic navigation. *Adaptive Behavior*, 2(3):277–305, 1994.

[40] C. Reynolds. Evolution of corridor following behavior in a noisy world. In D. Cliff, P. Husbands, J-A. Meyer, and S. Wilson, editors, *From Animals to Animats: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 402–410, Cambridge, MA, 1994. MIT Press.

[41] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. In J. McClelland and D. Rumelhart, editors, *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, Cambridge, MA, 1986.

[42] T. Smithers. On why better robots make it harder. In J-A. Meyer, H. Roitblat, , and S. Wilson, editors, *From Animals to Animats: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 64–72, Cambridge, MA, 1993. MIT Press.

[43] W. Van de Velde. Toward learning robots. *Robotics and Autonomous Systems*, 8(1–2):1–6, 1991.