ELSEVIER

Robotics and Autonomous Systems 19 (1996) 67–83

Robotics and Autonomous Systems

Challenges in evolving controllers for physical robots

Maja Matarić[a,*], Dave Cliff[b,1]

[a] *Volen Center for Complex Systems, Computer Science Department, Brandeis University, Waltham, Ma 02254, USA*
[b] *School of Cognitive and Computing Science, University of Sussex, Brighton BN1 9QH, UK*

Abstract

This paper discusses the feasibility of applying evolutionary methods to automatically generating controllers for physical mobile robots. We overview the state-of-the-art in the field, describe some of the main approaches, discuss the key challenges, unanswered problems, and some promising directions.

Keywords: Evolutionary computation; Robot control; Automated synthesis; Evolving controllers; Evolving hardware; Embodied systems; ffs; Morphology; Physical robots; Genetic algorithms; Genetic programming; Simulation.

1. Introduction

This paper is concerned with the distant goal of automated synthesis of robot controllers. Specifically, we focus on the problems of evolving controllers for physically embodied and embedded systems that deal with all of the noise and uncertainly present in the world. We will also address some systems that evolve both the morphology and the controller of a robot. Within the scope of this paper we define morphology as the physical, embodied characteristics of the robot, such as its mechanics and sensor organization. Given that definition, the only examples of evolving both morphology and control exist in simulation. Evolutionary methods for automated hardware design are an active subarea but are not directly relevant and are not discussed. We overview the main areas of research in applying genetic techniques to developing robot controllers in simulation, on physical systems, and in combination.

Section 2 reviews a selection of work to give an overview of the state-of-the-art in the field. Section 2.1 describes approaches to genetic programming in simulation, Section 2.2 describes approaches to evolving network controllers in simulation, Section 2.3 describes approaches to evolving both the morphology and the controller in simulation, Section 2.4 describes approaches to control in both simulation and the real world through the aid of shaping, Section 2.5 overviews various experiments with evolving in simulation and then testing on physical robots, Section 2.6 describes evolving hardware for a robot controller, Section 2.7 describes work on evolving controllers using real vision, and Section 2.8 overviews the few experiments in evolving fully on-board a physical robot. Section 3 addresses the key issues, challenges, unanswered problems, and promising directions for future research. Section 4 presents our conclusions.

* Corresponding author. Tel.:+1 617 736-2708; fax: +1 617 736-2741; e-mail: maja@cs.brandeis.edu.
[1] Tel.: +44 1 273 678754; fax: +44 1 273 671320; e-mail: davec@cogs.susx.ac.uk.

2. The state-of-the-art

In terms of evolving controllers for physical systems, genetic approaches have been applied to several different subdomains of the problem. Most of the work has used genetic techniques to evolve controllers for simulated robots. A small number of researchers has evolved controllers in simulation and then transferred them onto physical robots. Finally, an even smaller number has evolved directly on physical robotic systems in real time. This section reviews selected research from each of the areas in turn.

2.1. Genetic programming in simulation

Genetic programming (GP), introduced by John Koza [29,30], is one of the more popular approaches to evolving controllers in simulation. Unlike genetic algorithms, which typically operate on bit strings, GP manipulates higher-level primitive constructs such as Lisp programs. This abstraction of representation allows for significantly reducing the search space and can greatly accelerate the evolutionary process. However, the construction of the primitives requires additional craftiness on the part of the programmer, and it has been argued that it hides perhaps the most challenging aspect of controller design. On the other hand, it can also be argued that the use of higher-level primitives is a method for building domain knowledge into the evolutionary system in a form other than through the fitness function.

Genetic programming has been applied to evolving navigation and wall following in a simulated sonar-based robot. The goal of the project was to demonstrate the genetic programming can be used to evolve a subsumption-style [4] robot controller represented with Lisp S-expressions, and compare it to an existing robotic system developed by Matarić [35]. The genetic algorithm utilized many of Matarić's hand-crafted sensing and action primitives, including minimum overall sonar distance, minimum safe distance, edging distance, as well as control functions including move backwards by a fixed amount, turn right, turn left, and move forward [29]. Other primitives, such as stopping, were eliminated in the simulation, and all 12 simulated sonar readings were used, unlike the hand-crafted solution which only relied on the front and lateral readings. The boundary of the simulated robot arena, consisting of straight wall sections and right angles, was divided into squares using the designer-determined edging distance as a "wall-following" threshold. The fitness function was designed as a simple counter of the number of distinct near-wall squares the robot visited during its lifetime. To control complexity, the depth of initial random S-expressions was limited to 4, and that of the crossed-over expressions to 15. Consequently, the evolved expressions were not overly complex, but were nonetheless difficult to parse, although more understandable than the typical evolved network controllers we discuss later. The resulting S-expressions of the most successful individuals encoded a control algorithm that snaked the simulated robot around the walls of the entire arena.

Koza and Rice [32] applied the same GP approach to a box-pushing problem inspired by robotic work done by Mahadevan and Connell [34]. The system used 12 sonars, a bump sensor, a stuck sensor, a set of three action primitives (derived from the original robotics work), and four primitive functions based on the sensory readings. The fitness function was based on the distance between the shortest distance between the box and a wall. The evolved solution was successful in pushing the box toward the nearest wall for different initial box and robot positions. The authors experimented with an added sensory abstraction, "shortest sensor", which facilitated the simulated robot's performance.

The lack of a realistic noise model in the simulation begs the question of how general the results are. Furthermore, the importance of selecting the proper, in this case, expert hand-selected primitives, is difficult to estimate. Genetic programming has subsequently been successfully applied to a variety of domains, but we have not located any literature reporting GP being used on physical robots.

In the simulation domain, Craig Reynolds has applied genetic programming to a series of problems in autonomous agent control. In [43], GP was used to develop controllers which generate coordinated motion strategies in groups of "critters": simulated autonomous agents with idealized and highly simplified sensory and motor characteristics. Reynolds [45] applied GP to evolve controllers for collision-free navigation in twisting corridors. Finally, Reynolds [44] used GP to competitively co-evolve critters which play the children's pursuit-evasion game known as "tag" or "it".

2.2. Evolving network controllers in simulation

Research by Beer and Gallagher [3] demonstrated the use of genetic algorithms to develop continuous-time recurrent neural network controllers for artificial agents. Both the control networks and the artificial agents were simulated. Networks were evolved to produce coordinated sensory-motor behavior for two tasks: chemotaxis and locomotion.

The chemotaxis controller was responsible for guiding an agent to a source of "food" which emitted a chemical signal, the intensity of which diminished as the inverse-square of the distance from the source. The agent had a circular body, with two chemical sensors placed symmetrically either side of the center-line: the sensors produce a signal proportional to the intensity of the chemical signal at their location. The agent also had two effectors on opposite sides of the body which could move it forward in straight lines or in arcs of varying radii: the physics of movement were based on a simple model where velocity was proportional to the applied force. The controller was a six-unit fully interconnected network: two sensory neurons received signals from the chemical sensors, and two motor neurons provided output to the effectors; the remaining two were interneurons. The fitness of the agents was evoluated by monitoring their ability to move toward the source of food and, once at the source, stay there. A variety of strategies evolved: some agents used *tropotaxis* (i.e. moving forward while turning to the side with the stronger input stimulus); others used *klinotaxis* (i.e. oscillating from side to side with a bias toward the side of stronger stimulus) before switching to tropotaxis in the near vicinity of the food source. Once at the source, some agents evolved to come to a halt, while others circled the source, and still others repeatedly crossed the source and then reoriented toward it.

The locomotion controller experiments evolved parameters for a neural network governing leg movements in a simulated hexapod. Each leg was controlled by a small fully interconnected network of five units: three were output units which connected to effectors for raising/lowering the foot, swinging the leg forward, and swinging the leg backward; the other two were interneurons. All units in the network could receive input from a sensory unit giving a signal dependent on the angle made between the leg and the hexapod's body. For reasons of economy, parameters for only a single-leg controller were evolved: the subsequent controller was then replicated appropriately, one controller per leg, with sparse inter-controller connectivity: each unit in a controller made inter-leg connections with only the corresponding units in the controllers on the opposite side of the body, and the controllers of any adjacent legs on the same side of the body.

A single-leg controller network was evolved to produce appropriate stepping patterns: both in the presence of input from the angle-sensor, and with the sensor disabled (in which case the network had to act as a central pattern generator). In both cases, successful controllers evolved. In general, controllers evolved with sensory input showed severely degraded performance when the sensor was removed, but with the sensor attached were capable of fine-tuning their responses on the basis of sensory input – something the central pattern generator networks were not capable of. Results of evolving full-locomotion controllers (i.e., parameters for both the single-leg controller, and for the inter-leg connectivity) were broadly similar: the final evolved controllers exhibited tripod gait patterns akin to those used by fast-walking insects.

Beer and Gallagher compared the approach of evolving the full controllers from scratch with an incremental approach, where a single-leg controller was evolved and then, with the intra-leg parameters held fixed, the inter-leg connectivity parameters were evolved. They found that, although adequate full-locomotion controllers did results from this process, their performance was not as good as with those evolved from scratch. The converse was also true: taking leg controllers evolved from scratch in the full-locomotion experiments and comparing them with the evolved single-leg controllers showed that the full-locomotion controllers had come to be dependent on inter-leg signals to produce appropriate outputs.

2.3. Evolution of morphology and control in simulation

Cliff et al. [8] studied the evolution of visually guided robots, using a simulation model based on a real robot. The robot was controlled by continuous-time recurrent dynamical neural networks. The genotype of the robot coded not only for the connectivity and number of neurons in the controller, but also for the physical positioning of photosensors on the robot's body. The robot was evolved to perform a simple visually guided behavior: to find its way to the center of a circular room. The resulting controller networks were analyzed using a number of techniques: qualitative studies of the functional architecture of the network [9] and the effects of noise [7]; and quantitative studies based on dynamical systems analysis [26].

Sims [47] demonstrated a methodology for jointly evolving morphologies and controllers for embodied three-dimensional creatures. The creatures had realistic mass and inertial properties, and were situated in a physically based dynamical simulation. The physical dynamics of the creatures were by far the most complex and realistic of any evolved in simulation to date. The modeling as well as the evolutionary computation were performed on a CM-5. The morphology of the creatures was represented with a directed graph, a structure particularly well suited for applying various construction operators. Graph nodes could connect to themselves, form cycles, chains, and fractals. The connections between nodes contained information about the position, orientation, scale, reflection, and termination; each of the properties could be mutated independently. The nodes contained information about the dimensions of the part, the joints connecting it with the parent node, the limits on recursing, the connections to other nodes, and the local set of neurons (the controller) equipped with a fixed set of functions. Joint angle sensors, contact sensors, and photosensors could be attached to various nodes. Effectors controlled individual degrees of freedom of the joints between nodes, and received inputs either directly from sensors or via the neurons.

Sims devised a uniform graph representation for the brain and the body of the creature so the two could be evolved together. When a creature is synthesized, individual morphological components and their neural controllers are generated replicated, resulting in similar copies of separate subsystems (e.g., leg segments). The subsystems can be connected locally if they are adjacent in the graph hierarchy. They can also be connected through an independent non-replicated set of neurons which allow for the evolution of global synchronization. Initial genotypes are synthesized randomly, and their fitness is evaluated through a co-evolution process in which individuals compete directly. The genetic material of the winners is combined to create the next creature set. First each genome is mutated through a sequence of steps; internal node parameters are mutated, new nodes are added, the parameters of the connections are mutated, new random connections are added, and unconnected elements are removed. Genotypes with a higher number of components probabilistically undergo more mutation. The individuals are then combined through the application of three mating operators: 40% asexual, 30% crossover of nodes, and 30% grafting by connecting the different parents' nodes. The offspring then undergo another round of competition and the process is repeated.

The method produced a great variety of creature morphologies and behaviors, including creatures that jumped, slid, pushed, toppled, covered, and grasped. Sims also evolved a set of creatures that could follow a moving target. The dynamics of Sims' environment are the most complex used in simulation so far. Perhaps the only unrealistic feature is the simulator's determinism. Due to the computational complexity of the environment, non-determinism would have obviated the possibility of repeating any particular evolutionary run.

2.4. *Evolution by shaping in simulation and on robots*

Work by Colombetti and Dorigo [11,12] demonstrates the use of classifier systems for evolving controllers for a simulated and a real robot. In both cases, incremental learning through shaping was applied in order to accelerate convergence. In the simulated environment the distributed genetic algorithm was applied first to learning basic behaviors (chase, feed, and escape), then to learning their coordination (i.e., behavior selection) under different environmental conditions. Finally, the two types of learning were allowed to run in parallel, after the system had settled into a stable solution.

A simpler experiment was performed in the physical world, where AutonoMouse, a robot capable of sensing the direction of a light source, learned to turn and move toward the light. As in the simulation case, the on-line genetic learning algorithm was applied in the form of shaping; in the first phase of the learning the light target was stationary and the robot was free to move about and learn to approach it. In the second learning phase, the robot was specifically presented with the conditions it failed to learn in the first phase, until satisfactory behavior was observed. Finally, in the last phase the light target was moved and the robot was able to follow it.

The role of shaping is appropriate and useful, as it significantly accelerates the learning/adaptation process. In the described systems, the user could determine, by observation, what parts of the behavior have not been learned and could direct the robot to search in the appropriate area of the behavior space. The effectiveness of the resulting behavior and learning performance, then, is less due to the evolutionary aspect of the techniques than to proper human supervision. The described work is an interesting example of combining shaping and on-line classifier systems. The effectiveness of the approach is dependent on the proper design of the behavior primitives, much like in the case of genetic programming, and on well timed user intervention and guidance.

2.5. Evolution in simulation testing on robots

Several groups have undertaken the job of designing faithful simulations and evaluating their effectiveness by using them to evolve controllers and then test those on a physical robot.

Jakobi [27] and Jakobi et al. [28] developed a simulator for the Khepera robot developed at EPFL in Lausanne, Switzerland. This robot is a two-wheel differential-drive platform with eight active infrared (IR) proximity sensors, where the detector element of each sensor has some sensitivity to visible wavelengths, allowing it to be used in a passive mode to measure ambient light. The simulator system, known as Khepsim, models a single Khepera in a restricted class of environments: the user can specify some arrangement of obstacles (planar walls and upright cyclinders) in a bounded space, and may also position a light-source somewhere within the space. Khepsim is based on idealized mathematical models of the Khepera's kinematics, IR sensors, and ambient light sensors. These models were refined by incorporation of constants derived from empirical studies of the robot. A genetic algorithm was used to evolve neural-network controllers for the simulated Khepera. The evolved controllers could be down-loaded onto the real robot: because of the care taken in validating the simulation, the behavior of the real robot closely matched that shown in simulation. Controllers were evolved to perform obstacle avoidance (exploring the environment without colliding with obstacles) and light-seeking (starting from a random position and orientation in the environment and moving towards a 60 W desk lamp).

A series of experiments reported in [28] explored the effects of the level of noise in the simulation. The conclusion from these experiments was that if the noise levels in the simulator differ significantly from those in the real robot, the behavior of a controller in simulation is much less likely to be transferrable to the real system. When the noise in the simulator is less than that in the real world, it is likely that the evolved controllers will rely on unrealistically accurate actuator and sensor responses. Finally, when the noise in the simulator is set very high, the evolutionary process can exploit the potential of stochastic resonance effects, where noise boosts a weak signal sufficiently to allow it to have some effect on the behavior of the robot: Jakobi et al. [28] describe one controller evolved in a high-noise simulation which, when tested in the real robot (with real noise levels lower than those used in the simulation) failed to produce the behaviors seen in the simulation because the lack of noise failed to raise the response of the sensors above the threshold.

Although the Khepsim work demonstrates that it is possible to transfer controllers evolved in simulation to real robots, Jakobi et al. emphasize the fact that much care was taken in building the simulation and setting appropriate levels of noise, and that the success of their experiments may be due to the relative simplicity of the Khepera robot; the approach may be infeasible as the complexity of the robots and interaction dynamics increases.

Nolfi et al. [41] also describe work done with the Khepera robot. The authors built a simulator using real sampled position and infra-red data in order to perform experiments in evolving neural network-based controllers for navigation. The topology of the network was simple and fixed, consisting of the individual infra-red sensor inputs, two hidden units, and two outputs for the wheel motors. Synaptic connections and thresholds were coded in the chromosomes.

When the best evolved solutions were transferred to the physical robot and tested in the real world, the performance degraded significantly. The authors countered by allowing the adaptation process to continue on line and reported that within only a few generations the performance was again elevated to the simulation level, hypothesizing that the discrepancy between the simulated and the real world in this domain is quite small.

The authors have also evolved a grasping behavior on the Khepera, using the same simulator [42]. After some trial and error, they chose a 5-input 4-output network with no hidden units. The inputs included two frontal sensors, the average of the two left and two right side sensors, and the gripper sensor. The outputs included the two wheel velocities, and two triggers for specific procedures: pick-up and release. The fitness function was quite complex, and included elements of the robot's distance from the target object, the object's position relative to the robot, the robot's attempts to pick up an object, the presence of an object in the gripper and the release of the object. Although not all of the actions are directly rewarded, enough of them are that the fitness formula can be said to have a shaping effect. The best controllers evolved in the simulation, when transferred to the physical robot, declined in performance, but were able to correctly perform the task two or more times without collisions or inappropriate grasps, and were thus judged to be successful.

The use of shaping and basic primitive behaviors blurs the line between evolution and learning. For instance, work by Nolfi and Parisi [42] employs basic behaviors (rather than motor velocities or even low-level actions) and a shaped fitness function, both of which have been successfuly employed in robot learning [37]. While these approaches utilize the programmer's expertise and thus bias the learning/evolution process in order to simplify and accelerate, they may be necessary to scale up the existing approaches to any realistic robotic behaviors and tasks.

Another example of evolution in simulation and subsequent transfer to the physical robot is described by Miglino et al. [39]. Miglino et al. [39] describe the careful process involved in the design of their Khepera simulator. They offer a thorough statistical analysis of the different types and amounts of sensor noise that affected the resulting evolved behavior in a neural controller for navigation with a LEGO robot. The robot used two optosensors and high-level action including forward and backward by fixed amounts and fixed-degree left and right turns. A rough simulation was used in the evolutionary process in which the fitness function rewarded the number of novel locations the robot visited. The authors found that the addition of more realistic noise into the sensor models reduced the difference between the simulated and real world performance and facilitated the transfer. However, even though the efficiency of the real robot behavior was improved through the introduction of sensor noise in the simulation, a significant difference between the simulated and real navigation trajectories persisted [41].

Work by Grefenstette and Schultz [18] also describes an application of genetic algorithms, within a classifier system SAMUEL, to the problem of learning collision-free navigation in simulation, and transferring those to a mobile robot. In comparison to the Khepera simulations, this work used a more complex Nomad 200 mobile robot, equipped with 20 tactile, 16 sonar, and 16 infra-red sensors. To simplify the learning process, the authors abstracted the 52 sensory inputs and other available state of the robot into the following set: four sonar sensors, four infra-red sensors, time-step, speed, distance to the goal, and angle to the goal. The output being learned was a translation rate and rotation angle for each distinct set of inputs. The task consisted of learning to reach a specific goal region from a fixed start position within predetermined time. To prevent the system from learning an internal map of the environment, the start and goal positions were fixed, but the locations of obstacles were changed at each trial. The initial population consisted of a collection of random rules, some designed by hand and others generated automatically by mutating the hand-coded ones. Instead of mutation and crossover used in other systems we have reviewed, SAMUEL used generalization and specialization, classifier system operators that alter parts of the rule relative to its utility rating.

Both the genome representation and the operators used in SAMUEL are at a higher level than those used in the experiments described so far. Consequently, a relatively small population size (50 rules) was evaluated, using the average performance over 20 trials, and run over 50 generations. Best five rule sets were selected after each five generations, and tested on 50 trials to pick the best rule set. The best performing individuals performed the task successfully 93.5% of the time, and when transferred to the physical robot succeeded 86% of the time.

The success of the transfer between simulation and the real world in this system is at least in part due to the appropriate design of the initial rule set. As a consequence, the learning system was adapting thresholds rather than operating at the level of raw sensory inputs and motor outputs. SAMUEL's design also enabled the user to embed domain knowledge and heuristics into the initial population, thus further accelerating the learning process. The general approach has been successfully applied to other robot domains, as described in [46].

Yamauchi and Beer [49] describe the successful evolution of continuous-time recurrent neural networks which can learn to recognize landmarks from temporal sequences of sonar readings also on a Nomad 200 robot. Significantly, the networks exhibit forms of learning without changes in the connection weights: the learning is solely a result of changes in the networks internal dynamics as a consequence of perturbations from external (sensory) inputs. Also, Gallagher and Beer [17] have reported the transfer of evolved hexapod controllers (described earlier in this paper) to a real hexapod robot.

2.6. *Evolving hardware for control*

The vast majority of the work discussed so far involves using evolution to develop robot control *software*: even where the controller involves parallel distributed processing, the parallelism is frequently simulated on a fast serial von Neumann processor. Recently Thompson [48] reported results from experiments where robot control *hardware*, in the form of physical semiconductor circuits, can be evolved directly, i.e., without the need for a circuit simulator.

Thompson's technique requires the use of reconfigurable hardware: he demonstrates (in simulation) that a field programmable gate array (FPGA) can be used as the substrate for evolving recurrent asynchronous networks of high-speed logic gates, without imposing modularization or clocking constraints. The FPGA demonstration involves the evolution of an oscillator circuit which produces spikes at a frequency of approximately 4 k Hz, from gates which have propagation delays of no more than one nonosecond. [2]

Thompson demonstrates the direct evolution of a control circuit which produces wall-avoidance behavior in differential-drive wheeled robot, using two sonar heads pointing left and right of the robot's direction of travel. The controller is based on Thompson's notion of a dynamic state machine (DSM). A DSM is similar to a finite state machine (FSM) implemented by direct-addressed ROM, where a clocked register holds the current state; this, when combined with input variables, gives the address input for a ROM. The data outputs of that address of the ROM give the next state and the output variables of the FSM. A DSM differs from a direct-addressed-ROM FSM in two important respects. First, RAM is used instead of ROM, to allow the transitions and outputs of the system to be reconfigurable, and hence evolvable. Second, the input, state, and output variables may be either synchronous, in which case their clock rate is genetically specified, or asynchronous. Whether a variable is synchronous or asynchronous is genetically specified. If any of the state variables are asynchronous, then the circuit is not an FSM. DSMs have the potential for rich intrinsic dynamics: reflexive input–output responses can be produced, yet at the same time internal state can be perturbed or maintained over genetically specified time scales.

Because the temporal dynamics of a DSM depend on the physical characteristics of the hardware it is built from, it is not practical to simulate them. Instead, Thompson uses a reconfigurable DSM circuit in the real robot, with direct sensory input from the sonars to the DSM and direct output from the DSM to the motors. Thus, the DSM receives raw echo signals from the sonars, and directly drives the motors. One evolved controller successfully produced wall-avoidance behavior using only 32 bits of RAM and three flip-flops.

Although the controller was evolved directly in hardware, there was some use of simulation: the robot was put on jacks so that its wheels span in the air, while sensors monitored their rotation, and used the data to update the position of a simulated robot in a simulated environment. The simulator then synthesized the sonar signals that would result from the

[2] The best randomly connected circuit in the initial population produced spikes at a frequency of around 18 M Hz.

robot's orientation and location in its environment, and supplied these to the physical DSM. Thus the robot could be physically stationary while the DSM is subjected to a "virtual reality" (VR) evaluation. The authenticity of the VR results can be checked by taking the robot off the jacks and testing it in a real environment. Thompson demonstrates a close correspondence between the virtual and real behaviors of the wall-avoidance controllers evolved in this manner.

While DSMs are a simpler form of reconfigurable hardware than FPGAs, either approach to developing asynchronous control circuits requires that, for the full potential to be available, the real hardware has to be evaluated in real-time. The intrinsic dynamics of the physical circuit would be very difficult to model in simulation, and any sufficiently accurate simulation is likely to run much slower than real-time on a serial machine. In principle, fully synchronous control circuits could be evaluated at rates faster than real-time, by increasing the clock speed, but this would require the sensory-motor interactions with the environment to be simulated at an appropriately accelerated rate in evaluation, and we are not aware of anyone who has done this.

2.7. *Evolving with real vision*

Producing simulations of visual sensing, with bandwidth of anything more than a few pixels, that are sufficiently accurate to be worth using in evolutionary robotics can take prohibitively long. Using a real video input system is a much more attractive option.

Harvey et al. [24] developed a cartesian gantry-robot which allowed for 50 Hz frame rates from a 64×64 monochrome CCD camera with an umbilical video-feed cable to off-board computers. The gantry allowed for three-dimensional physical translation of the camera, and a mirror mounted on a stepper-motor allowed for the direction of view of the camera to be rotated through 360° in the horizontal ("yaw") plane without any twisting of the umbilical. The camera mounting (or "head") included eight binary touch-sensors for detecting collisions with obstacles.

Building on experience from their earlier simulation studies, discussed above, Harvey et al. evolved neural network controllers to produce a variety of visually guided behaviors. Rather than allow the neural network controllers direct access to the motors governing the x, y, z, and yaw of the camera head, code was written to provide a "virtual" differential-drive wheeled robot: the outputs of the network controllers were nominally to the left and right motors: these signals were then transformed to appropriate changes in the yaw-angle and horizontal (x, y) position of the camera head – the vertical altitude of the head was kept fixed, thereby simulating a wheeled robot on a planar surface. The mapping from left and right motor values to x, y, and yaw outputs included terms to model the momentum of the robot. Thus, while the motor side of the robot was, at least in part, a simulation, it was possible to evolve neural network controllers for real-time visual control with real video input. The gantry offered a further advantage: the head could be positioned and re-positioned very accurately, so near-identical initial conditions could be used for successive trials of the same or differing genotypes.

The visual input to the neural networks was given by a genetically specified sampling pattern: the genotype could specify a number of circular visual receptive fields: the radius and position of the center of the receptive field (in image coordinates) were read from the genome for each visual sensory input unit in the network. During an evaluation, the instantaneous input to the sensory unit was an estimate of the instantaneous average image intensity of the pixels within the unit's receptive field.

The gantry was used to demonstrate the principle of *incremental evolution* [23]. Rather than starting with a population of random genotypes when attempting to evolve controllers to achieve some challenging task, it is better to evolve from a population which has already been selected for a similar but less-challenging task. In [24], a sequence of evaluation tasks was used on a population of size 30 to evolve controllers capable of an elementary visual discrimination task. The sequence started with 12 generations where selection was for behavior which took the robot towards a large white rectangular target in an otherwise dark visual environment. Following this, the population was selected for ability to approach a much smaller white rectangular target: after six generations, an individual emerged which could approach the smaller target and also "pursue" it if the target was moved at a reasonable speed. The population was then subjected to 15 generations of selection for the ability to head towards a small white triangle, and avoid moving towards a white rectangle of size similar to the triangle. After 15 generations, fit individuals emerged, giving a total of 33 generations in the incremental sequence. As a comparison, the same initial random population was evolved for 15 generations of selection for approaching the small white target. At the end of 15 generations, there were high-scoring individuals in the population but further analysis indicated that these had less robust controllers than the best of the population emerged incrementally for the same task.

One of the final controllers evolved to approach the triangular target was analyzed to see how it worked: the active part of the network relied on only two receptive fields, set at an angle which allowed for discrimination between an oriented edge of the triangular target and the vertical edges of the rectangular target.

2.8. *Evolution entirely on robots*

Work by Floreano and Mondada is one of the very first attempts of evolving controllers entirely on a physical robot in real-time without any human intervention. In [14,15] the authors describe successful results of evolving navigation and obstacle avoidance behaviors on a Khepera robot. This robot has proven to be the most successful platform for physical evaluation of genetic approaches to controller evolution, due to its small size and protability, as well as the ability to be tethered for external computational and battery power. The described experiments used the physical sensors and effectors of the robot, processed by an onboard microcontroller, to act and evaluate in the real world, but performed the computation on an off-board workstation that provided a significant improvement in computational and memory capacity. The robot was also equipped with special designed hardware for interacting with an external laser positioning device that allowed for recording the robot's exact movement over time for subsequent analysis.

The controller is represented in the form of a neural network whose weights and thresholds are coded as floating point values in the genetic string. The topology of the network, a multi-layer perceptron of sigmoid units with a set of recurrent connections at the hidden layer, was static, while the weights were modified by the evolutionary process. The inputs to the network consisted of the robot's eight infra-red sensory values, and the outputs fed velocity commands directly to the two motors on the wheels (i.e., no high-level actions were used). The fitness function penalized collisions and lack of movement.

In the first set of experiments, the authors demonstrated a robust collision-free navigation behavior, but had to add a bending penalty to the fitness function in order to prevent a particular efficient solution that kept the robot spinning in a small circle within an obstacle-free area.

In [40] the authors describe the approach applied to evolving a homing behavior. A decaying battery was simulated and the robot evolved a "recharging behavior" in which it learned an internal representation of the environment and moved toward the light (i.e., the "recharging station") which boosted its simulated battery level. The evolved behavior was also tested in an altered environment, when the light source was removed. The robot executed a searching behavior within the appropriate region until its battery was exhausted. The evolved behavior kept the robot wandering around its world until its internal battery level reached a low level, then taking a short path toward the light.

The authors also describe the evolution of a simple grasping behavior [40] by adding graspable balls to the environment, a simple gripper to the robot, and introducing gripping to the action set. The fitness function was based purely on the number of grasped objects. In order to evolve all aspects of gripping, including approaching an object, an incremental approach was employed. The system was first run on a high density of balls in the environment, then run on half-density to further refine the search and approach behaviors, much like the method used in [13]. The learned gripping behavior is more complex than most others because it involves the use of sensors with different ranges: the gripper sensors are only relevant after the other infrared get the robot close enough to the ball to be grasped. This experiment was successful, but required several days of continuous evolution, even after the gripping action was reduced to a fixed action pattern to minimize complexity.

3. Issues and challenges

One of the main goals of the work on evolutionary robotics is to provide a methodology for automatically synthesizing more complex behaviors than those that can be designed by hand. However, a survey of the results in the field to date does not show any demonstrations that have reached that goal. Much like the state-of-the-art in learning on robots, none of the evolved or learned behaviors have been particularly difficult to implement by hand. To develop evolutionary techniques to a level where they can seriously be considered for use in designing robots, there are many challenges to be overcome, critical questions to be addressed, and some promising directions to be explored. This section addresses them in turn.

3.1. Evolving on physical robots

Real time on real hardware. Evolution on physical systems takes prohibitively long. As demonstrated by the successful example of evolving collision-free navigation on a Khepera [14] at approximately 39 min per generation and a hundred generations, 65 h were required to evolve the desired behavior. While the authors present impressive experimental techniques and repeatable data, it is clear that this approach will not scale up to more complex behavior evolution that will require both more time per individual trial as well as more generations.

Battery lifetime. The unavoidable need to recharge robot batteries further slows down the experimental procedure. In most of the Khepera-based experiments described, the robot was tethered thus eliminating both the on-board power and computation problem, but tethering is not possible on all platforms and in all domains, nor does it scale up to multi-robot co-evolution experiments.

Robot lifetime. Aside from the prohibitive time overhead, physical hardware of a robotic system cannot survive the necessary continuous testing without constant maintenance and repairs. As the complexity of behaviors scales up, the need to offload much of the experimentation and evaluation to a simulation will increase.

3.2. Evolving in simulation

Noise and error models. The difficulty of accurately simulating physical systems is well known in robotics [5]. Since it is impossible to simulate all details of a physical system, any abstraction made in a simulation may be exploited by the genetic algorithm and result in behavior that is maladaptive in the real world. As has been empirically demonstrated, too little, too much, or too inaccurate noise in a simulation creates nontransferrable systems. Consequently, it has been necessary to obtain careful measurements from the robot in order to construct the simulator. The Khepera is particularly convenient for modeling due to its clean and simple design, but most robost do not share that property, and have already been found difficult to simulate for the purposes of testing hand-crafted controllers in simulation. It would appear that the application of evolutionary methods to controller design only makes it more difficult to work in simulated domains and highlights this unsolved robotics problem.

Generality v. usefulness of simulations. As described above, the most successful simulations have been based on accurate physical measurements incorporated into the sensor and effector models, as well as into the fitness function. This not only makes the job of writing a simulation for a nontrivial robot very challenging, it also produces an extremely specialized tool that does not generalize to any other system. The investment of time required to construct very faithful simulations has proven to be largely prohibitive in robotics in general, where most researchers have employed quite coarse simulations but allow for some algorithmic testing, and are then coupled with subsequent evaluation of a physical system. A similar, incremental approach has been applied to testing evolved controllers that required subsequent refinement on the real system. As the complexity of robotic systems grows and the gap between the simulation and the real system widens, the question of the value of investing in a specialized simulation will become increasingly important.

Brooks [6] suggests the application of interleaved off-line (simulated) evolution and on-line (physical) evaluation as a means of "connecting to reality". Most of the approaches we reviewed that transitioned from simulation to the real world and completed the evolutionary process there, stopped at that level rather than continue interleaving by returning the system to simulation for evolving still higher-level behaviors. In an ideal scenario the interleaving could be performed automatically without the need for human intervention. However, even in a simplified human-controlled form, such a continuous on-line off-line development and evaluation process may be necessary for scaling up to more complex behaviors. Related approaches have been used by Floreano [13] and Harvey et al. [24], both on physical robots, to incrementally evolve, evaluate, and "freeze" behaviors before moving to the next more complex level.

3.3. Evaluation

Experimental repeatability. Difficulties with experimental repeatability, both at the level of evaluating individual genotypes and replicating entire experiments, are inherent in robotics work, due to the noise in the sensors, effectors, and the environment resulting in a large variance across trials. Any stochastic components in the algorithms compound the problem. Simulation designers have the luxury of eliminating non-determinism in order to create repeatable trials [47], but the results are not necessarily relevant in the physical, nondeterministic environment.

The problems of variation and noise in evaluation are well known in the evolutionary computation community. This issue is particularly important in evolving architectures for autonomous agents. A recent paper by Aizawa and Wah [1] describes techniques for scheduling GAs in noisy environments: one technique allocates a given number of samples (i.e. fitness evaluations) among the members of the population in an adaptive fashion. In this method, different individuals in the population may get different numbers of evaluations: an individual may be evaluated repeatedly in order to form an accurate estimate of its underlying performance distribution. In cases where sampling fitness is costly (as is generally the case in evolving robot systems), this method can save time that would otherwise be wasted if too many samples are taken. The method also helps prevent too few samples from being taken, which could lead to inaccurate estimations of fitness and hence incorrect application of the differential reproduction.

Behavior convergence. Determining when the desired behavior has been achieved on a physical robot is notoriously difficult. Consequently, much of the analysis is qualitative and based on human judgement. While human observers can apply reasonable phenomenological performance descriptions, relevant quantitative analysis is rare [11]. Average performance is difficult to establish as trials vary significantly, and due to the overhead of physical experiments as well as their variability, typically insufficient data are available for statistically significant analysis. Since this problem is endemic in physical system evaluation [38] it must also be addressed by the evolutionary robotics community.

3.4. Fitness function design

Complexity of design. The process of designing an evaluation function for behavior evaluation on a robot is delicate and laborious [40]. In the majority of cases, the necessary insights are gained through incremental augmentation over many trials in the environment. Unfortunately, most of this insight is lost since typically only the best fitness function is reported but not the complex process that generated it.

Fitness function complexity. As with any learning and optimization technique, a genetic algorithm will exploit all of the available fitness features, some of which may be opaque to the designer. The work on evolving effective grasping behavior [42] has already demonstrated that, as more complex behaviors are evolved that involve the interaction of multiple goals and subgoals, the more complex the fitness function becomes. To date, the designers have resorted to indirectly embedding all of the subgoals into the fitness function. While it may be effective, this approach eliminates the autonomy of evolution and both fails to reduce the job of the designer and strongly biases the possible solutions.

Measurability of fitness parameters. Nolfi et al. [41] point out that the design of a fitness function in simulation, however complex, is easier than the same in the real world where not all of the sensory information is readily available. For example, much of the navigation and homing work depends on the robot's position within the environment, which can be directly obtained in simulation, but not necessarily outside of one. Experiments have added special sensors for this purpose. While such solutions can be very elegant (e.g., see [41] for a clever example used in evolving object size discrimination), they are not general and do not scale to more complex robot behaviors.

3.5. Co-evolution

Co-evolution has been shown to be a powerful method for searching fitness landscapes [25,33]. It has been particularly successful in evolving robust co-operative and competitive behaviors in societies [33], but has not yet been applied to evolving controllers for physical robots; the co-evolutionary simulations discussed above [44,47] are not intended to result in the controllers for real robots. However, as a method for accelerating evolution, co-evolution could be applied both in simulation and on a physical group of robots. A variant of the latter idea has been applied by Matarić [36] to achieve collective learning in a group of robots through a direct exchange of received reinforcement and learned information. The introduction of crossover and mutation would turn this approach into a readily testable method for on-line co-evolution, assuming the availability of the robots and inter-robot communication.

However, co-evolutionary systems can exhibit dynamics which run counter to those required of an engineering-design evolutionary system, where something approximating "continuous progress" is the desired dynamic. In co-evolutionary systems, cycles through genotype space are possible, and limited "genetic memory" can lead to an overall reduction in performance levels of the evolved controllers (see [10] for further details). Nevertheless, it may be possible to automatically detect such situations and take appropriate actions [10], in which case co-evolution promises to be an approach with considerable potential.

3.6. Genetic encodings

Using evolutionary techniques to generate robot control systems requires that parameters determining the nature of the controller are encoded in a manner suitable for use in a genetic algorithm. In most cases, this encoding is a string of characters: the "genotype" of the robot. In much evolutionary robotics research, the encodings are application-specific, and sometimes ad hoc; researchers develop an encoding scheme with features necessary for the task at hand. This is not always easy: in some cases, the refinement of an encoding scheme can represent a significant amount of work. Increasingly, the need for more general purpose encoding schemes, applicable across a range of applications, is being recognized. Here we comment on some of the properties that may be required of such encoding schemes.

A primary requirement is that the encoding is robust with respect to the genetic operators employed: mutation of a fit individual should, on the average, yield an individual of roughly the same fitness. Similarly, crossover between two parents of similar fitness should, on the average, produce off-spring with similar fitness.

In cases where a genetic algorithm (GA) is being used to tune or optimize a set of parameters influencing a pre-specified controller, this robustness requirement is typically the sole concern; and in many cases it does not present a serious difficulty. Yet in such cases a significant proportion of the design effort can be expended in creating the initial controller which is then configured by evolution. For example, if a GA is used to configure the weights and thresholds in a pre-specified neural network controller, then the designer needs to make a priori commitments concerning the number of units, and their potential connectivity. It is possible that insufficiently many units are in the initial design, so no satisfactory setting of the network parameters can be found. Some effort (either analytic or empirical trial-and-error) is required to determine the minimum network architecture. This issue could possibly be sidestepped by using a manifestly over-specified initial network architecture, i.e., have many more units than are likely to be necessary, in the hope that the GA finds a solution involving a large number of zero weights, effectively deleting some of the units from the network. Yet such an approach could easily waste much time evaluating overly complex designs, and it would probably also be necessary to include terms in the evaluation function to encourage the use of fewer active units. Arriving at the correct balance between rewards for behavior and rewards for economy in controller design may not be easy.

A more appealing prospect is to have the controller, and possibly also the morphology, be (as far as is possible) entirely the product of an automatic evolutionary process, thereby minimizing the pre-commitment on important architectural issue. As was noted above, a number of researchers have studied the evolution of co-adapted morphology and control (as advocated by Brooks [6]), but only in simulation. The advantages of such co-adaptation indicate that any truly general encoding scheme should smoothly integrate specifications for both controller and morphological features. The encoding used by Sims [47] is a promising start, but has two aspects which require attention: first, the simulations were not intended as models of real robots – building a working physical version of one of Sims' evolved agents is probably beyond the limits of current robot technology; second, Sims' description of the mutation and recombination processes involved in breeding new agents makes it clear that these processes are significantly more complex than the flipping of bits or splicing and concatenation of bit-strings that are more commonly used in GAs. This increase in procedural complexity of the genetic operators is not a welcome prospect, but it may be unavoidable.

The encoding scheme should allow for a wide variety of possible architectures. The widest possible range is given by allowing the length of the genotypes to be variable, as in Koza's genetic programming. Harvey [20–23] has developed the *species adaptation genetic algorithm* (SAGA) explicitly for dealing with variable length genotypes. The attraction of SAGA is that (assuming that the length of the genotype is in reasonable correspondence with the complexity of the resultant phenotype), if no upper limit is put on genotype length, then in principle it should be possible to start with short, simple genotypes which produce elementary behaviors sufficient for evolution to operate on, and then the length can increase until it encodes for a sufficiently complex design. In this sense, to use fixed-length genotypes is to presuppose the dimensionality of the space of possible solutions explored by the genetic search, and this presupposition might be wrong. As with mutation and crossover, variations in length should be achievable without, on the average, significant impact on fitness. To this end, Harvey [21] developed a specialized crossover procedure for variable-length genotypes.

In may autonomous agent applications, it is highly desirable to have some degree of symmetry, both in morphology and in the responses of the controller [6]. Using n-fold symmetry can, in principle, reduce the length of the genotype by a factor of n: for instance, with twofold (e.g., bilateral) symmetry, the left-hand side of the controller/morphology can be specified on the genome, while the right-hand side can be generated by reflection in the appropriate axial plane. Nevertheless, some mechanism for sommetry breaking is also likely to be required, as the responses of symmetric controllers can have weaknesses such as singularities on the plane of reflection.

In order that complex controller architectures can be encoded in compact genotypes, it is also highly desirable that the encoding scheme allows for repeated substructures: this need is particularly acute in evolving parallel distributed processing architectures for visually guided agents, where a particular arrangement of processing units may need to be repeated over a raster of visual input units (e.g., to achieve the same effect as convolving the input image with a difference-of-gaussians mask). Gruau [19] has developed a modular encoding scheme for neural networks, inspired by Koza's work on gene-splicing for automatically defined functions (ADFs) [31]. In Gruau's scheme, the genotype specifies a sequence of graph-rewrite operations which are applied to an initial graph consisting of a single neuron: many of the rewrite operators generate one or more new neurons via a process of "cell-division". The rewrite sequences have a binary-tree structure, with the two subtrees from a branch-node specifying sequences of rewrite rules to be applied to the two daughter networks resulting from application of a division operator. Using Koza's gene-splicing technique, the genotype is divided into a number of separate trees ordered in a hierarchy: terminals in a tree can be "subroutine" calls to trees lower in the hierarchy, allowing for modularity and repeated structures. Gruau developed this scheme to satisfy the following criteria which he argues should be met by any modular neural network encoding scheme:

- *Completeness*: any network can be encoded.
- *Compactness*: encodings should be of minimal size.
- *Closure*: any genotype should produce a meaningful network.
- *Modularity*: the code for the network should be formed from code for subnetworks, with the possibility of module re-use, or recursion.
- *Scalability*: the length of the encoding should only weakly reflect the complexity of the network: a fixed-length encoding should allow for a variety of networks.
- *Expressive power*: the encoding should be capable of expressing the network connectivity, the weights, the "learning" method, and so on.

Gruau's encoding scheme exhibits these properties, and has been demonstrated to work on a number of problems, including the control of a simulated hexapod walking robot. Further empirical experience with this and other encoding schemes is required before their generality can be fully determined.

3.7. Combinatorics of evaluation

To evaluate the fitness of an individual genotype, it is common for the individual to be subjected to a number of trials, with the fitness being calculated from a summary statistic such as the mean score. There are two primary motivations for basing fitness on the results of multiple tests: the need to counter the effects of nondeterminism (discussed previously in this paper) and the need to control for the effects of *free parameters* in the evaluation process. The free parameters may be, for example, the arrangement of obstacles in an environment, or the ambient lighting conditions, or the initial conditions (position and orientation) of the robot, and any other agents it may need to interact with. In many applications, the setting of these free parameters may have a significant effect on the subsequent observed behavior. If a small fixed set of parameter settings is employed, there is the danger that the evolutionary process opportunistically exploits this regularity, over-fitting to the test set and failing to produce desired behaviors in other conditions. Attempting to avoid this problem by assigning random values to the free parameters on each trial introduces a new source of stochastic variation: two identical controllers may record differing fitnesses because of chance variations in the initial conditions which they were subject to during evaluation; worse still, a good genotype may be incorrectly rated as less fit than a poorer one.

An alternative is to have sufficiently many trials with different settings of the free parameters to ensure that the final evolved system is robust with respect to variation in these parameters. But there may be no method of determining a priori how many trials is sufficient, so a period of trial-and-error experimentation may be required. For any single free parameter p, some number n_p of trials should be conducted with different values of p. We will ignore here the potentially problematic issue of deciding how the n_p trials should be distributed through the range of possible values of p.

Now assume that there are f free parameters, and the possibility of interactions between the settings of the parameters cannot be ruled out, so each free parameter is varied in turn while keeping the others constant. For simplicity's sake, assume that the same number n_p of different value-settings is used for each parameter. Then there will be N_p^f trials spread throughout the f-dimensional space of possible parameter values. This number can grow prohibitively quickly.

For example, suppose there are five free parameters ($f = 5$), each of which is tested with four different values ($n_p = 4$), and suppose that each trial takes 15 s; use a reasonable population size of 100, and evolve for 100 generations. While all these parameters are reasonable, the result is not: there will be $4^5 = 1024$ trials to control for the effects of variations in initial conditions. With each trial lasting 15 s, evaluating a single genotype will take about 4.25 h. With a population of size 100 a single generation would require about 2.5 weeks to evaluate, and the 100th generation will finish in roughly five years. This is true regardless of the nature of the trial: it could be 15 s of monitoring a noisy physical robot, or 15 s of deterministic simulation.

Of course, this is an exaggerated example: there are established statistical techniques for experiment design which offer principled methods for reducing the number of evaluations required, and often problem-specific heuristics can be introduced to cut short an individual trial, or to terminate the evaluation of a particular genotype. But there is a further issue in evolving autonomous mobile robots which seems to create an inescapable difficulty: there are, we presume, many classes of desirable behavior where it is not possible to evaluate an individual in a few seconds. Furthermore, some behaviors may, by their nature, not be evaluable until the end of the trial, thus preventing premature termination as a time-saving measure. A population of size 100, evolving for 100 generations, at the bare minimum of one trial per individual, will last 10000 times the duration of a single trial. If the trial lasts 1 min, the experiment takes a week, and so on. In the limit, evolving controllers to produce behaviors of the type: "do this behavior and keep doing it for as long as possible" is likely to be fraught with difficulties: as the controllers get better, the duration of evaluating an individual will increase, and the rate of evolutionary search will slow down. While some kind of macro-parallel architecture could be employed to factor the population-size out of the equations (e.g., if working in simulation, use one simulation per individual, spread across as many workstations as there are individuals in the population), evolving for temporally extended tasks is an open problem.

3.8. *Evolving from higher-level primitives*

Koza's extension of genetic programming to use gene-splicing and automatically defined functions (ADFs), where the building-blocks of evolution at one level are themselves evolving at a lower level, is certainly an appealing and apparently powerful development. But we are cautious about the use of Lisp S-expressions for robot control. The expressive granularity of LISP could introduce further problems if due care is not exercised. Evolving with a restricted set of primitives may limit the design space, possibly to such an extent that the evolution of appropriate controllers is a theoretical impossibility (i.e., there is no possible configuration of the available primitives that produces an appropriate controller). On the other hand, evolving with larger sets of primitives introduces the danger that the design space is so large that successful evolution becomes a practical impossibility (i.e., an appropriate configuration of primitives is extremely unlikely to be found). The latter problem is more likely if there is considerable epistatic interaction between the primitives, which would seem to be an increased possibility as the primitives become more complex.

Neural networks are widely acknowledged to exhibit tolerance in the presence of noise, and graceful degradation with respect to component failure: these features indicate that the fitness landscapes for neural network controllers are likely to be smooth (i.e., show relatively little epistasis). Furthermore, as Beer [2] notes, a particular class of continuous-time recurrent neural networks (CTRNNs) has the added attraction of being universal dynamics approximators, i.e., the trajectory of any smooth dynamical system can be approximated by such networks [16]. Beer's analysis of the dynamics of small CTRNN circuits [2] indicates that particular simple circuits, of one or two neurons, could constitute very useful building blocks for evolving larger CTRNNs with rich intrinsic dynamics.

Nevertheless, Koza's ADF concept can still be employed, in the manner demonstrated by Gruau. If Gruau's techniques, or some similar approach involving the evolution of a hierarchical modular network structure, can be developed to work successfully using the small CTRNN building blocks indetified by Beer, then evolving genuinely complex and challenging controllers for real robots may be a possibility.

4. Summary

The work we have reviewed indicates that evolutionary techniques have some promise for the automatic synthesis of both robot control systems and the physical morphology for robots. Such work demonstrates, by providing existence proofs, that evolutionary techniques can in principle reduce the human effort required to configure or design robot systems.

But for a real reduction in human effort, the effort expended in designing or configuring the evolutionary system should be less than that required to manually design or configure the robot controllers that it produces. In general, so far this has not been the case; the behaviors produced by current evolved controllers are, on the whole, relatively simple, and could have been designed by hand with the same or lesser amount of effort. At this early stage, this is not necessarily cause for alarm: the matter of interest is not what the controllers make the robots do, but how the controllers came to be. Yet to develop evolutionary techniques to a level where they can seriously be considered for use in designing robots, there are several challenges to be overcome, and some critical questions to be addressed. If the challenges can be successfully addressed, the use of evolutionary techniques may become a viable alternative to manual design.

References

[1] A.N. Aizawa and B.W. Wah, Scheduling of genetic algorithms in a noisy environment, *Evolutionary Computation* 2 (2) (1994) 97–122.

[2] R.D. Beer, On the dynamics of small continuous-time recurrent neural networks, *Adaptive Behavior* 3 (4) (1995) 471–511.

[3] R.D. Beer and J.C. Gallagher, Evolving dynamical neural networks for adaptive behaviour, *Adaptive Behaviour* 1 (1) (1991) 91–122.

[4] R.A. Brooks, A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* RA-2 (1986) 14–23.

[5] R.A. Brooks, Intelligence without reason, in: *Proc. IJCAI-91* (1991).

[6] R.A. Brooks, Artificial life and real robots, in: *Proc. Toward a Practice of Autonomous Systems, 1st European Conf. on Artificial Life* (MIT Press, Cambridge, MA, 1992) 3–10.

[7] D. Cliff, I. Harvey and P. Husbands, Evolved recurrent dynamical networks use noise, in: S. Gielen and B. Kappen, eds., *Proc. Int. Conf. on Artificial Neural Networks* (Springer, Berlin, 1993) 285–288.

[8] D. Cliff, I. Harvey and P. Husbands, Explorations in evolutionary robotics, *Adaptive Behavior* 2 (1) (1993) 71–108.

[9] D. Cliff, P. Husbands and I. Harvey, Analysis of evolved sensory motor controllers, Technical Report CSRP 264, School of Cognitive and Computing Sciences, Sussex University. Presented at: 2nd *European Conf. on Artificla Life* (1992) unpublished proceedings.

[10] D. Cliff and G.F. Miller, Tracking the red queen: Measurements of adaptive progress in co-evolutionary simulations, in: F. Morán, A. Moreno, J.J. Merelo and P. Chac0́n, eds. *Advances in Artificial Life: Proc. 3rd Int. Conf. on Artifical Life* (Springer, Berlin, 1995) 200–218.

[11] M. Colombetti and M. Dorigo, Learning to control an autonomous robot by distributed genetic algorithms, in: J.-A. Meyer, H.L. Roitblat and S.W. Wilson, eds., *Proc. Simulation of Adaptive Behavior* (MIT Press, Cambridge, MA, 1993) 305–311.

[12] M. Colombetti and M. Dorigo, Training agents to perform sequential behavior, *Adaptive Behavior* 2 (3) (1994) 247–276.

[13] D. Floreano, Patterns of interactions in shared environments, in: *Toward a practice of autonomous systems: Proc. 1st European Conf. on Artificial Life* (1993) 347–366.

[14] D. Floreano and F. Mondada, Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot, in: *Simulation of Adaptive Behavior* (1994) 421–430.

[15] D. Floreano and F. Mondada, Evolution of homing navigation in a real mobile robot, *IEEE Transactions on Systems, Man, and Cybernetics* (1996).

[16] K. Funahashi and Y. Nakamura, Approximation of dynamical systems by continuous time recurrent neural networks, *Neural Networks* 6 (1993) 801–806.

[17] J.C. Gallagher and R.D. Beer, Application of evolved locomotion controllers to a hexapod robot, Technical Report CES-94-7, Case Western Reserve University Department of Computer Engineering and Science, 1994.

[18] J. Grenfenstette and A. Schultz, An evolutionary approach to learning in robots, in: *Proc. Machine Learning Workshop on Robot Learning*, New Brunswick, NJ (1994).

[19] F. Gruau, Automatic definition of modular neural networks, *Adaptive Behavior* 3 (2) (1994) 151–183.

[20] I. Harvey, The artificial evolution of behaviour, in: J.-A. Meyer and S.W. Wilson, eds., *From Animals to Animats: Proc. 1st Int. Conf. on the Simulation of Adaptive Behavior* (MIT Press, Cambridge, MA, 1990).

[21] I. Harvey, The SAGA cross: the mechanics of crossover for variable-length genetic algorithms, in: R. Männer and B. Manderick, eds., *Parallel Problem Solving from Nature 2* (North-Holland, Amsterdam, 1992) 269–278. Also available as University of Sussex School of Cognitive and Computing Sciences Technical Report CSRP223.

[22] I. Harvey, Spices adaptation genetic algorithms: A basis for a continuing SAGA, in: F. Varela and P. Bourgine, eds.,*Towards a Practice of Autonomous Systems: Proc. 1st European Conf. on Artificial Life* (MIT Press, Cambridge, MA, 1992) 346–354. Also available as University of Sussex School of Cognitive and Computing Sciences Technical Report CSRP221.

[23] I. Harvey, Evolutionary robotics and SAGA: the case for hill crawling and tournament selection, in: C. Langton, ed., *Artificial Life 3 Proc.*, Santa Fe Institute Studies in the Sciences of Complexity, Proc. Vol. XVI (Addison-Wesley, Reading, MA, 1993) also available as University of Sussex School of Cognitive and Computing Sciences Technical Report CSRP222, 1992.

[24] I. Harvey, P. Husbands and D. Cliff, Seeing the light: Artificial evolution; Real vision, in: D. Cliff, P. Husbands, J.-A. Meyer and S.W. Wilson, eds., *From Animals to Animats 3: Proc. 3rd Int. Conf. on Simulation of Adaptive Behavior*, (MIT Press, Cambridge, MA, 1994) 392–401.

[25] W.D. Hillis, Co-evolving parasites improve simulated evolution as an optimization procedure, *Physica D* 42 (1990) 228–234.

[26] P. Husbands, I Harvey and D. Cliff, Circle in the round: state space attractors for evolved sighted robots, *Robotics and Autonomous Systems* 15 (1995) 83–106.

[27] N. Jakobi, Evolving sensorimotor control architectures in simulation for a real robot, Master's thesis, University of Sussex School of Cognitive and Computing Sciences, 1994 (unpublished).

[28] N. Jakobi, P. Husbands and I Harvey, Noise and the reality gap: the use of simulation in evolutionary robotics, in: F. Morán, A. Moreno, J.J. Merelo and P. Chacón, eds., *Advances in Artificial Life: Proc. 3rd Int. Conf. on Artificial Life* (Springer, berlin, 1995) 704–720.

[29] J.R. Koza, Evolution of subsumption using genetic programming, in: F.J. Varela and P. Bourgine, eds., *Proc. 1st European Conf. on Artificial Life* (MIT Presss, Cambridge, MA, 1990) 110–119.

[30] J.R. Koza, *Genetic Programming* (MIT Press, Cambridge, MA, 1992).

[31] J.R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs* (MIT Press, Cambridge, MA, 1994).

[32] J.R. Koza and J.P. Rice, Automatic programming of robots using genetic programming, in: *Proc. AAAI-92* (MIT Press, Cambridge, MA, 1992) 194–201.

[33] H.H. Lund, Specialization under social conditions in shared environments, in: *Proc. Advances in Artificial Life, 3rd European Conf. on Artificial Life* (1995) 477–489.

[34] S. Mahadevan and J. Connell, Automatic programming of behavior-based robots using reinforcement learning, in: *Proc. AAAI-91*, Pittsburgh, PA (1991) 8–14.

[35] M.J. Matarić, Integration of representation into goal-driven behavior-based robots, *IEEE Transactions on Robotics and Automation* 8 (3) (1992) 304–312.

[36] M.J. Matarić, Learning to behave socially, in: D. Cliff, P. Husbands, J.-A. Meyer and S. Wilson, eds., *From Animals to Animats: Int. Conf. on Simulation of Adaptive Behavior* (1994) 453–462.

[37] M.J. Matarić, Reward functions for accelerated learning, in: W.W. Cohen and H. Hirsh, eds., *Proc. 11th Int. Conf. on Machine Learning* (Morgan Kauffman, New Brunswick, NJ, 1994) 181–189.

[38] M.J. Matarić, Evaluation of learning performance of situated emboided agents, in: *Proc. Advances in Artificial Life, 3rd European Conf. on Artificial Life* (1995) 579–589.

[39] O. Miglino, H.H. Lund and S. Nolfi, Evolving mobile robots in simulated and real environments, Technical Report 95–04, Institute of Psychology, C.N.R., Rome, 1995.

[40] F. Mondada and D. Floreano, Evolution of neural control structures: some experiments on mobile robots, *Robotics and Autonomous Systems* (1996).

[41] S. Nolfi, D. Floreano, O. Miglino, and F. Mondada, How to evolve autonomous robots: different approaches in evolutionary robotics, in: *Proc. Artificial Life IV* (1994) 190–197.

[42] S. Nolfi and D. Parisi, Evolving non-trivial behaviors on real robots: an autonomous robot that picks up objects, in: *Proc. 4th congress of the Italian Association for Artificial Intelligence* (Springer, Berlin, 1995)

[43] C. Reynolds, An evolved, vision-based behavioral model of coordinated group motion, in: J.-A. Meyer, H. Roitblat and S. Wilson, eds., *Proc. 2nd Int. Conf. on Simulation of Adaptive Behaviour (SAB92)* (MIT Press, Cambridge, MA, 1993) 384–393.

[44] C. Reynolds, Competition, coevolution, and the game of tag, in: R. Brooks and P. Maes, eds., *Artificial Life IV* (MIT Press, Cambridge, MA, 1994) 59–69.

[45] C. Reynolds, Evolution of corridor following behavior in a noisy world, in: D. Cliff, P. Husbands, J.-A. Meyer and S.W. Wilson, eds., *From Animals to Animats 3: Proc. 3rd Int. Conf. on Simulation of Adaptive Behavior* (MIT Press, Cambridge, MA, 1994) 402–410.

[46] A.C. Schultz, Using a genetic algorithm to learn strategies for collision avoidance and local navigation, in: *Proc. 7th Int. Symp. on Unmanned Untethered, Submersible Technology*, Durham, NH (1991) 213–225.

[47] K. Sims, Evolving 3D morphology and behavior by competition, in: *Proc. Alife IV* (MIT Press, Cambridge, MA, 1994) 28–39.

[48] A. Thompson, Evolving electronic robot controllers that exploit hardware resources, in: F. Morán, A. Moreno, J.J. Merelo and P. Chacón, eds., *Advances in Artificial Life: Proc. 3rd Int. Conf. on Artificial Life* (Springer, Berlin, 1995) 640–656.

[49] B.M. Yamauchi and R.D. Beer, Integrating reactive, sequential, and learning behavior using dynamical neural networks, in: D. Cliff, P. Husbands, J.-A. Meyer and S.W. Wilson, eds., *From Animals to Animats 3: Proc. 3rd Int. Conf. on Simulation of Adaptive Behavior* (MIT Press, Cambridge, MA, 1994) 382–391.

Maja J Mataric is an assistant professor in the Computer Science Department and the Volen Center for Complex Systems at Brandies university. She received a Ph.D. in Computer Science and Artificial Intelligence from MIT in 1994. She has worked at NASA's Jet Propulsion Lab, the Free University of Brussels AI Lab, LEGO Cambridge Research Labs. GTE Research Labs, and the Swedish Institute of Computer Science. Her Interaction Laboratory conducts research on the dynamics of interaction in complex adaptive system, multi-agent systems, control and learning in intelligence agents, and cognitive neuroscience modeling of visuo-motor skill learning.

Dave Cliff was born in 1996. He has a B.Sc. in Computer Science from the University of Leeds, and MA and D.Phil. degrees in Cognitive Science from the University of Sussex. He was a founder member (with I. Harvey and P. Husbands) of the Sussex Evolutionary Robotics Research Group. His research interests are primarily in parallel distributed processing for visual control of action in autonomous agents: both animals and artificial creatures. Dr. Cliff is currently a Lecturer in Computer Science and Artificial Intelligence at the University of Sussex, and an associate faculty member of the Sussex Center for Neuroscience.