Evolving Obstacle Avoidance Behavior in a Robot Arm

David E. Moriarty and Risto Miikkulainen
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
moriarty,risto@cs.utexas.edu

Abstract

Existing approaches for learning to control a robot arm rely on supervised methods where correct behavior is explicitly given. It is difficult to learn to avoid obstacles using such methods, however, because examples of obstacle avoidance behavior are hard to generate. This paper presents an alternative approach that evolves neural network controllers through genetic algorithms. No input/output examples are necessary, since neuro-evolution learns from a single performance measurement over the entire task of grasping an object. The approach is tested in a simulation of the OSCAR-6 robot arm which receives both visual and sensory input. Neural networks evolved to effectively avoid obstacles at various locations to reach random target locations.

1 Introduction

Many industrial tasks such as assembly, packaging, and processing rely heavily on the manipulation and transportation of small components. Robot arms can automate many of these processes and improve the cost efficiency of the operation. To be as effective as their human counterparts, robot arms normally use vision systems based on one or more cameras to identify and locate the target objects (Feddema and Lee 1990; Papanikolopoulos and Khosla 1993; van der Smagt 1995; Weiss et al. 1987; Wijesoma et al. 1993).

Vision-based robot arm control is a very complex task that requires mapping the information of the target and obstacle locations to the joint rotations that position the hand near the target. This task in commonly known as hand-eye coordination. Because it is difficult to specify such a mapping by hand, many researchers have applied machine learning techniques to learn the control strategy. The resulting policies are often much more robust than manually-designed, fixed policies.

One particularly successful approach to control learning is *neuro-control* (Baker and Farrell 1992; Werbos 1992) or using a neural network to learn to implement the control policy. Most approaches to robot arm neuro-control learn hand-eye coordination through supervised training methods such as backpropagation or conjugate gradient descent (Kawato 1990; Miller 1989; van der Smagt 1995; Werbos 1992). Supervised learning, however, requires training examples that demonstrate correct mappings from input to output. Training examples are normally generated by flailing the arm while recording joint movements and final arm positions (Werbos 1992). The supervised approach is sufficient for learning basic hand-eye coordination in domains with unrestricted movement, however, in uncertain or obstacle-filled domains the supervised approach fails. The main problem is that in the supervised training the arm is always moved to the target location in a single joint rotation. In domains with obstacles, this is not always possible because the arm must move around an obstacle. Explicitly generating the necessary intermediate joint positions is extremely difficult and requires significant domain knowledge (Lumelsky 1987). Without such knowledge, it is not possible to learn obstacle avoidance behaviors through supervised methods.

This paper presents an alternative learning control system that *evolves* neural networks through genetic algorithms. Neuro-evolution does not require explicit training examples and learns multiple joint rotations implicitly through evolution. The evolutionary search is guided by a single fitness evaluation over the entire task, which may involve multiple skills such as avoiding obstacles and reaching the target. Neuro-evolution can thus form network controllers that can adapt in uncertain environments.

Evolving neuro-control was tested in a sophisticated robot arm simulation of the OSCAR-6 anthromoporphic robot. The evolution was based on the Hierarchical SANE system (Moriarty and Miikkulainen 1996a, 1996b). Networks were evolved to maneuver the arm to random target locations while avoiding obstacles. Given both camera-based visual and infrared sensory input, the networks learned to effectively combine both target reaching and obstacle avoidance strategies. The current experiments detect obstacle collisions only in the robot's hand; future experiments will extend the control policy

In *From Animals to Animats: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB-96).* Cape Cod, MA.

to avoid obstacles with the full arm.

The body of this paper is organized as follows. The next section summarizes the existing methods for learning robot arm control in neural networks and describes their limitations. Section 3 motivates the use of neuro-evolution, and section 4 describes the implementation of the Hierarchical SANE system to evolve hand-eye coordination and obstacle avoidance in the OSCAR-6 robot. Section 5 describes the experiments conducted in an OSCAR-6 simulator and summarizes the main results. Enhancements to the simulator and applications to real robot arms are outlined in section 6, followed by our conclusions in the final section.

2 Learning to Manipulate a Robot Arm

Several methods have been proposed for learning robot arm control in neural networks (Kawato 1990; Kuperstein 1991; Miller 1989; van der Smagt 1995; Walter et al. 1991). Each of these methods is based on supervised learning from a corpus of input/output examples that demonstrate correct behavior. During training, the network is presented inputs from the database and its output is compared to the desired output. Errors are calculated according to the differences, and modifications are made to the network's weights based on some variant of the backpropagation algorithm. In any supervised learning application, it is crucial that the training corpus contains a good representative sample of the desired behavior.

The most common approach for generating training examples is to flail the arm and record the resulting joint and hand positions (Werbos 1992). For example, if the joints are initially in position $\vec{J}$ and a random rotation $\vec{R}$ results in hand position $\vec{H}$, a training example of the form $Input : \vec{J}, \vec{H}; Output : \vec{R}$ can be constructed. This example reflects the correct rotation to reach target position $\vec{H}$ from joint position $\vec{J}$. Given a sufficient database of such examples, a neural network can learn to approximate the inverse kinematics necessary to translate between the camera-based visual and joint spaces.

A major limitation of generating training examples by "flailing" is that it only applies to situations where the target can be reached in a *single* joint rotation. It cannot demonstrate more general behavior such as reaching while simultaneously avoiding obstacles, where a sequence of rotations are necessary. For example, when an obstacle is placed between the arm and the target, the arm cannot take a direct path, but must instead make several moves around the obstacle. Random arm movement would never produce a sufficient training example for this situation, since there is no single rotation that can reach the target. To produce such behavior using a supervised learning approach, training examples must demonstrate movement to intermediate arm positions (e.g. above the block). It is unclear how such examples could be generated without a path-planning algorithm (Lumelsky 1987), which requires significant domain knowledge of the robot and its environment.

An alternative to supervised learning is to use a *reinforcement learning* method, such as Q-learning (Watkins and Dayan 1992) or genetic algorithms (Goldberg 1989), to form the control policy. In reinforcement learning, no input/output examples are required, and thus no path planning algorithms are necessary to generate intermediate arm positions. Agents learn from signals that provide some measure of performance and which may be delivered after a sequence of decisions have been made. Since reinforcement signals take into account several control decisions at once, appropriate credit can be assigned to the intermediate joint rotations that are necessary to reach the final target position. Reinforcement learning can thus integrate sophisticated behaviors such as obstacle avoidance into a robot arm control policy.

Since artificial evolution of neural networks has been shown competitive and in many cases more efficient than other reinforcement learning methods (Moriarty and Miikkulainen 1996a; Whitley et al. 1993), the approach in this paper is based on neuro-evolution as the reinforcement learning method.

3 Evolving Neuro-Controllers

Recently there has been much interest in evolving neural networks with genetic algorithms in control tasks (Cliff et al. 1993; Moriarty and Miikkulainen 1996a; Nolfi et al. 1994; Whitley et al. 1993; Yamauchi and Beer 1993). Genetic algorithms (Holland 1975; Goldberg 1989) are global search techniques patterned after Darwin's theory of natural evolution. Numerous potential solutions are encoded in strings, called *chromosomes*, and evaluated in a specific task. Substrings, or *genes*, of the best solutions are then combined to form new solutions, which are inserted into the population. Each iteration of the genetic algorithm consists of solution evaluation and recombination and is called a *generation*. The idea is that structures that led to good solutions in previous generations can be combined to form even better solutions in subsequent generations.

Since genetic algorithms do not require explicit credit assignment to individual actions, they belong to the general class of reinforcement learning algorithms. In genetic algorithms, the only feedback that is required is a general measure of proficiency for each potential solution. Credit assignment for each action is made implicitly, since poor solutions generally choose poor individual actions. Thus, which individual actions are most responsible for a good/poor solution is irrelevant to the genetic algorithm, because by selecting against poor solutions, evolution will automatically select against poor actions.

In neuro-evolution, the solutions take the form of neural networks. Properties such as weights and connections are encoded in chromosomes, which are evolved by the ge-

netic algorithm. Neuro-evolution offers many important advantages to robotic arm control over supervised methods. First, it operates using only the overall performance of the neural network controllers as a guide. If avoiding obstacles is a necessary component of good performance, the genetic algorithm will select for networks that can avoid obstacles. No input/output examples are necessary, and thus neuro-evolution is not constrained by the inability to generate training examples. Second, neuro-evolution can be applied with very little *a priori* information. Knowledge of the robot arm dynamics, the arm's environment, or the components of the visual system is not necessary as they are in path-planning algorithms. Neuro-evolution evolves this knowledge through experience and tailors it's control policy to meet the specific demands of the domain.

4 A Neuro-Evolution Implementation for Robot Arm Control

This section describes a neuro-evolution algorithm and architecture for controlling the OSCAR-6 robot arm. Figure 1 shows a picture of an OSCAR robot. The current implementation is designed for evolution in a simulation model and then application to the real robot arm. Future work will study evolution directly with the OSCAR robot.

The network controller receives both camera-based visual and infrared sensory input representing the location of the target and the distance from obstacles, and must resolve a series of joint rotations to position the hand at a target location. The hardware specifications and algorithms used to generate the input are independent of the learning system and are considered given. For descriptions of camera-based vision and infrared sensors in robot manipulators, see (Lumelsky 1987; Papanikolopoulos and Khosla 1993; Sanderson and Weiss 1983; van der Smagt 1995; Wijesoma et al. 1993). The focus of this paper is how to automatically integrate the sensory information into an effective control policy.

4.1 Primary and Secondary Control Networks

The task of reaching a target can be seen as a composite of two basic movements. First, the robot arm must make several large joint rotations to get within a certain proximity of the target object. Such rotations often involve detecting and avoiding obstacles in the arm's path. Second, the robot arm must make smaller, more precise movements to position the end effector within grasping distance of the target object. This observation leads to an efficient design of a neuro-evolution system, where control is divided between two networks. The first, called the *primary network*, positions the arm near the target, while the *secondary network* makes the smaller movements to reach the target object.

Figure 1: The OSCAR-6 robot arm. OSCAR is designed for pick-and-place tasks and has been applied in several industrial settings. The arm contains 6 total joints and a camera mounted in the end effector.

When a task is initiated, the primary network moves the arm until it specifies that the arm should be stopped or it exceeds a prespecified maximum number of joint rotations. There is no fixed proximity boundary for the primary network; its goal is to "get as close as it can". It is possible to evolve primary networks to complete the entire task, however, because of the diminishing returns of late evolution, it is often more advantageous to accept a certain level of proficiency and begin a new search for secondary networks. Since the secondary networks start relatively close to the target, there is normally little need for an obstacle avoidance strategy. Thus, any of the existing supervised approaches can generate effective secondary networks. Secondary networks are evolved in this paper to demonstrate that neuro-evolution can solve the entire task. Evolving the secondary networks also generates obstacle avoidance behavior for situations where the primary network leaves the arm very close to an obstacle.

4.2 Network Architectures

Each neural network controller (primary and secondary) contains 9 input, 16 hidden, and 7 output units (figure 2). The input units correspond to the x, y, and z relative distances of the hand to the target and six directional proximity sensors located on the hand that sense obstacles in the negative and positive x, y, and z directions. In other words, they sense obstacles in back of, in front of, to the left of, to the right of, above, and below the end effector. They have a 10 cm range and return the absolute distance to the obstacle. If no obstacle is currently within the sensor range, the activation is 10.0.

Each joint's rotation is determined by two unique output units. The first unit is linear and the sign of its total

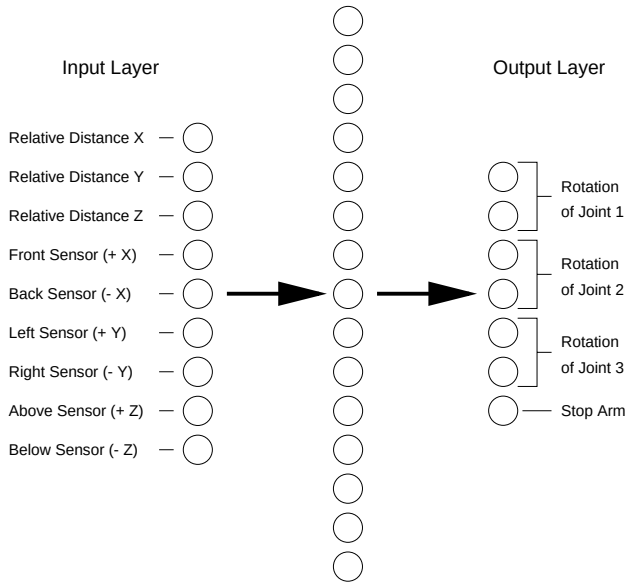

Figure 2: The architecture of the neural network controller for the OSCAR-6 robot. The dark arrows indicate activation propagation from the input to hidden layer and hidden layer to output layer. The connections and weights between the hidden layer and the input and output layers are evolved by the genetic algorithm.

activation specifies the direction of rotation. The second output unit is sigmoidal and specifies the magnitude of rotation. Dividing the output function this way makes it easier for hidden units to control a specific function, such as the direction of rotation of a particular joint. In the primary network, the magnitude output units are normalized between 0.0 and 5.0, limiting each joint rotation to [-5,+5] degrees. This forces the primary network to make several small joint rotations to reach the target, which allows it to more effectively sense and avoid obstacles in the arm's path. In the secondary network, the rotation is normalized between 0.0 and 1.0 to allow for fine movements near the target.

When the joint rotations are small, it is not necessary to take into account a large magnitude of distance from the target object. Such information can only interfere with the next local movement. Thus, it is very useful to "cap" the camera-based visual input units at 10.0 cm, such that if the target object is further away than 10.0 cm in any direction, the corresponding input unit receives an activation of only +/- 10.0.

A final threshold output unit is included as an override unit that can prevent movement regardless of the activations of the other output units. If the activation of the override unit is positive, the arm is not moved. If it is negative, the joint rotations are made based on the other output units. Without the override unit, stopping the arm would require setting the activation of the three sigmoidal units to exactly 0.0. Since genetic algorithms do not make systematic, small weight changes, it is very difficult to evolve neurons to compute such exact values. The override unit allows networks to easily stop the arm when it is sufficiently close to the target.

4.3 The Genetic Algorithm

The Hierarchical SANE (Symbiotic, Adaptive Neuro-Evolution) system was used to form the hidden layer connections in the neuro-control networks. SANE[1] was designed as a fast, efficient genetic algorithm for building neural networks in domains where it is not possible to generate training data for normal supervised learning. Symbiotic evolution has been evaluated in several tasks including the standard pole-balancing benchmark where it outperformed existing neuro-evolution and reinforcement learning approaches (Moriarty and Miikkulainen 1996a).

In contrast to standard neuro-evolution algorithms that evolve a population of neural networks, in SANE two separate populations are evolved: a *population of neurons* and a *population of network blueprints*. The neuron population provides efficient evaluation of the genetic building blocks, while the population of blueprints learns effective combinations of these building blocks.

Each individual in the blueprint population consists of a set of pointers to individuals in the neuron population. During each generation, networks are constructed by combining the hidden neurons specified in the blueprints. Each blueprint receives a fitness according to how well the corresponding network performs in the task. Each neuron receives a fitness according to how well the top five networks in which it participates perform in the task. A very aggressive genetic selection and recombination strategy is used to quickly build new structures in both the neuron and blueprint populations (see (Moriarty and Miikkulainen 1996b) for details).

SANE offers two important advantages over other neuro-evolution approaches. First, it maintains diverse populations. Because several different types of neurons are necessary to build an effective neural network, there is inherent evolutionary pressure to form neurons that perform different functions. Diversity allows recombination operators (crossover) to continue to generate new neural structures even in prolonged evolution, which ensures that the solution space will be explored efficiently. Second, SANE decomposes the search for solutions into a search for partial solutions. Instead of searching for complete networks all at once, solutions to smaller problems (good neurons) are evolved, which can be combined to form an effective full solution (a network).

[1]For the remainder of the paper, the name SANE will refer to the hierarchical version described in (Moriarty and Miikkulainen 1996b)

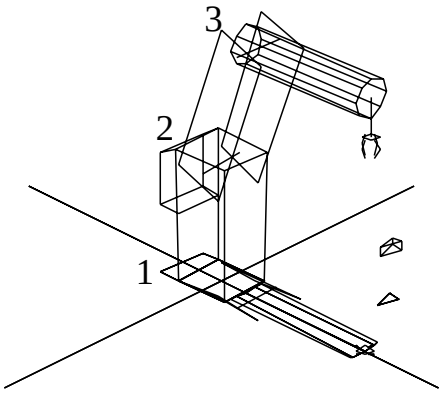

Figure 3: The Simderella simulation of the OSCAR robot. The numbers indicate the joints which are to be controlled.

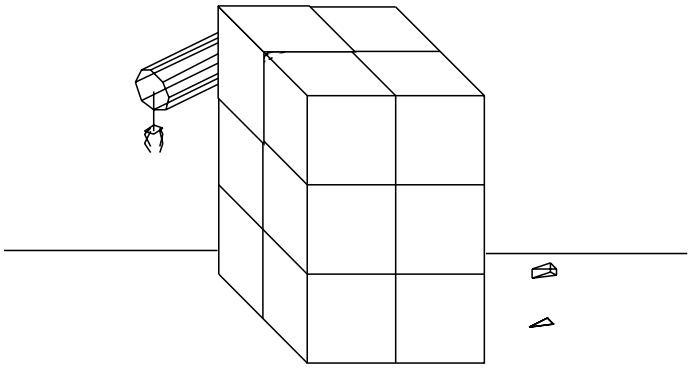

Figure 4: The 12 obstacle placements in the robot simulator. Each box is $30 \times 30 \times 30$ centimeters. During any trial, one box is filled with an obstacle.

5 Evaluation

5.1 Experimental Setup

The Simderella 2.0 package written by van der Smagt (1994) was used as the robot arm simulator in these experiments. Simderella is a simulation of the OSCAR-6 anthromoporphic arm, and van der Smagt (1995) has reported that controllers that perform well in Simderella exhibit very similar performance when applied to OSCAR. Figure 3 illustrates the Simderella robot. Our eventual goal is to demonstrate evolved neuro-controllers in a real robot arm, however, currently simulation models are sufficient to test the viability of our approach.

Obstacles were introduced in the Simderella environment in the form of "boxes". Figure 4 shows the twelve different obstacle placements used in the simulations. During each trial, which consists of a sequence of moves to reach a target, one of the boxes is occupied by an obstacle. If the end effector moves into an occupied box, the trial ends, and the last position before the collision is used as the final position for fitness evaluation. This obstacle scheme is fairly primitive, since obstacles always have the same size and there is no check if the rest of the arm (besides the hand) violates an occupied box. However, the task is still quite difficult and, to our knowledge, no existing supervised learning approach can learn the intermediate joint rotations without significant *a priori* information about the size, shape, and location of the obstacles.

Each neural network evaluation begins with random, but legal, joint positions and a random target position. The target and hand are never started within an obstacle. The same reach space as van der Smagt (1995) is employed, where targets are placed within a 180 degree rotation of the first joint. A total of 450 target positions were created and separated into a 400 position training set and a 50 position test set. During evolution, a target position is randomly selected from the training set before each network evaluation. During each trial, a network is allowed to move the arm until one of the following conditions occur:

1. The network stops the arm.
2. The network places the arm in an illegal position (e.g. hits the floor).
3. The network hits an obstacle.
4. The number of moves exceeds 40.

The score for each trial is computed as the percentage of distance that the arm covered from its initial starting point to the target position. For example, if the arm started 120 cm from the target and its final position is 20 cm from the target, the network receives a score of $(120 - 20)/120 = 0.83$. The percentage of distance covered, instead of the absolute final distance, provides a fairer comparison between a network that receives a close target and a network that receives a distant target. Each network is evaluated over a single randomly selected target.

A population of 1600 neurons and 200 network blueprints are evolved by SANE. The first stage of evolution consists of only primary networks. The population is evolved for 200 generations, and the best network of each generation is tested over the 50 target test set. The overall best network is then fixed as the primary network, and the secondary network evolution begins from a random population.

5.2 Results

Figure 5 shows the performance of the primary networks per generation averaged over 10 simulations. The graph plots the average final distance of the best neural network found at or before each generation. On average, a network capable of moving the arm within an average of 10 cm was found within the first 100 generations.

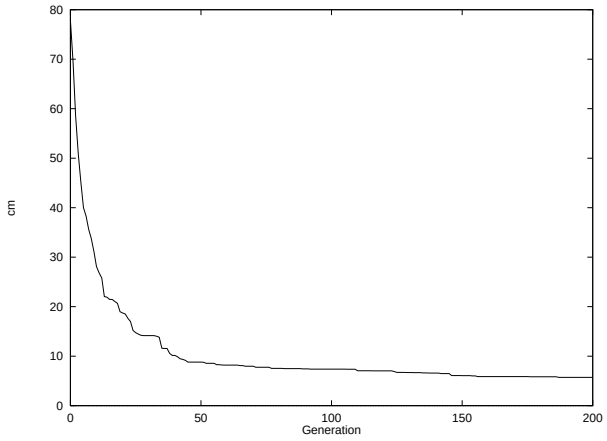

Figure 5: Evolution of primary networks. The average distance from the targets is plotted for the best network found so far at each generation. The curve is an average over 10 simulations; in each simulation, the distances were averaged over 50 randomly placed targets.

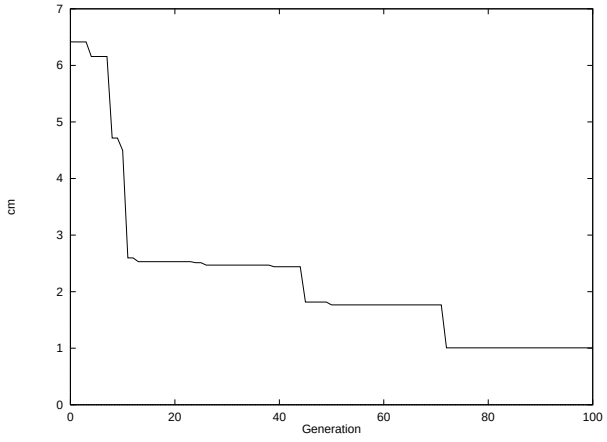

Figure 6: Evolution of secondary networks. The distance per generation is plotted, averaged over 50 trials and 10 simulations. In each simulation, a different primary network was used.

Figure 6 shows the performance of the secondary networks per generation. Again, the graph presents an average of 10 simulations. In each simulation, the primary network was taken from a different primary evolution. Within 80 generations, the secondary networks were able to position the arm with an error of only 1 cm, which is considered acceptable for most industrial applications (van der Smagt 1995). Thus, in this task, the combination of the primary and secondary networks can effectively control the robot arm to within industry standards.

It is difficult to measure how efficiently a network avoids obstacles as a function of each generation, since early networks do not hit many obstacles simply because they do not move the arm very far. Thus, counting the number of hits is a poor measure of obstacle avoidance. A better metric is the percentage of trials in which the

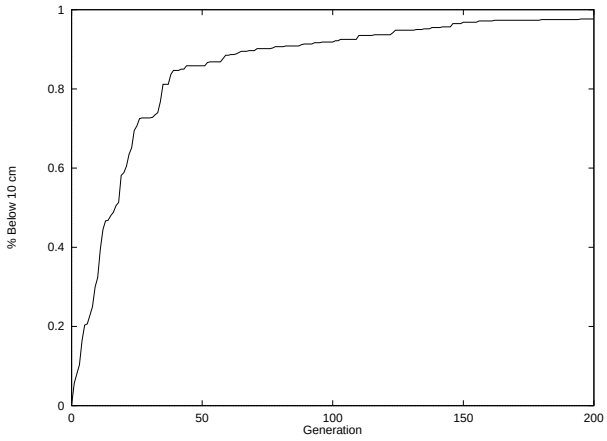

Figure 7: Evolution of obstacle avoidance behavior. The percentage of trials within 10 cm per generation is plotted for the primary network evolution.

primary network positions the arm within 10 cm of the target object. To achieve a high percentage, the network must contain a strong avoidance strategy coupled with an effective target reaching ability. Figure 7 plots this percentage for the best primary networks at each generation. On average, the best primary networks moved within 10 cm of the target objects 98% of the time. When collisions did occur, it was often not due to poor control decisions. With only 6 proximity sensors, blind spots are inevitable, and occasionally a nearby obstacle can not be detected. Thus given more sensors, the primary networks should encounter even fewer obstacles.

In order to discover how often it was necessary to avoid obstacles, a manually-designed inverse-kinematics controller[2] was tested on the 50 position test set with different box configurations. On average, the fixed controller, which takes the most direct path, hit obstacles in 11% of the trials. Since the best networks found in these simulations hit obstacles in only 2% of the trials, we can conclude that significant avoidance strategies have been evolved.

6 Future Work

The simple task described in this paper is already an important advancement in robot control, since it demonstrates that it is possible to learn a general obstacle avoidance strategy in a neuro-controller. However, the obstacle task needs to be scaled up before neuro-evolution can be claimed effective in real-world robot arm control. The approach will be scaled up in two stages: first, the obstacle task in the Simderella simulator will be extended to less regular obstacles and infrared sensors will be placed along the robot arm to prevent collisions with the entire arm. Second, the approach will be tested with a

[2]The controller is included as part of Simderella simulator

real robot arm. Simulation models are useful for evaluating new control algorithms, but important issues such as sensor noise and real-time control are often difficult to model. Thus, to truly validate our approach experiments are needed using real hardware.

Experiments will examine the feasibility of evolving neural network controllers directly with a real arm. Since domain models are expensive to generate and require significant *a priori* knowledge, it is important to study neuro-evolution as a model-free control system. While SANE does contain a very fast genetic search engine, the number of network evaluations currently required may expend too much time in real hardware. Improvements to the network architecture and fitness calculation may significantly reduce the number of evaluations. For example, dividing each joint movement between two output units (direction and magnitude) instead of one, resulted in about half as many evaluations to reach the same level of proficiency. In addition, much of our work will continue to focus on methods for improving the general efficiency of a neuro-evolution search.

The neural controller described in this paper is a fixed adaptive controller (Werbos 1992); once the controller is evolved it does not change. However, it would be desirable to build a controller that can adapt online to take advantage of domain specific information. Nolfi and Parisi (1995) have developed a method where evolved neural networks compute training signals for the the controller after every activation. Such networks could be used as online learning controllers by evolving the ability to adjust connection weights in response to the specific environment. Future work will study if evolved local learning can create more robust neuro-controllers and help networks evolved in simulation adapt to real hardware.

7 Conclusion

In many industrial settings, it is crucial for a robot arm to detect and avoid obstacles in the its path. Existing methods for learning robot arm control, however, cannot learn the intermediate joint rotations necessary to move around an obstacle. By evolving neuro-controllers with genetic algorithms, such rotations can be learned since performance is evaluated over multiple control steps. Experiments in a sophisticated simulation of the OSCAR-6 robot arm showed that neuro-evolution can effectively integrate both target reaching and obstacle avoidance into a single control policy. Future experiments will examine the application and scale-up potential of this approach to real robot arms.

Acknowledgments

The authors would like to thank Patrick van der Smagt for making his Simderella simulator available. The Simderella software was developed by Patrick van der Smagt and is supported by the Dutch Foundation for Neural Networks. The OSCAR-6 robot is owned by the Autonomous Systems Group at The University of Amsterdam. The picture of the OSCAR-6 robot is reprinted with permission. This research was supported in part by the National Science Foundation under grant #IRI-9504317.

References

Baker, W. L., and Farrell, J. A. (1992). An introduction to connectionist learning control systems. In *Handbook of Intelligent Control*, 35–63. New York: Van Nostrand Reinhold.

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

Feddema, J. T., and Lee, G. C. S. (1990). Adaptive image feature prediction and control for visual tracking with a hand-eye coordinated camera. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(5).

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Ann Arbor, MI: University of Michigan Press.

Kawato, M. (1990). Computational schemes and neural network models for formation and control of multijoint arm trajectory. In *Neural Networks for Control*. Cambridge, MA: MIT Press.

Kuperstein, M. (1991). INFANT neural controller for adaptive sensory-motor coordination. *Neural Networks*, 4(2).

Lumelsky, V. J. (1987). Algorithmic and complexity issues of robot motion in an uncertain environment. *Journal of Complexity*, 3:146–182.

Miller, W. T. (1989). Real-time application of neural networks for sensor-based control of robots with vision. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(4):825–831.

Moriarty, D. E., and Miikkulainen, R. (1996a). Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–32.

Moriarty, D. E., and Miikkulainen, R. (1996b). Hierarchical evolution of neural networks. Technical Report AI96-242, Department of Computer Science, The University of Texas at Austin.

Nolfi, S., Floreano, D., Miglino, O., and Mondada, F. (1994). How to evolve autonomous robots: Differ-

ent approaches in evolutionary robotics. In *Artificial Life IV*. Cambridge, MA.

Nolfi, S., and Parisi, D. (1995). Learning to adapt to changing environments in evolving neural networks. Technical Report 95-15, Department of Neural Systems and Artificial Life, Institute of Psychology, CNR - Rome.

Papanikolopoulos, N. P., and Khosla, P. K. (1993). Adaptive robotic visual tracking: Theory and experiments. *IEEE Transactions on Automatic Control*, 38(3):429–444.

Sanderson, A. C., and Weiss, L. E. (1983). Adaptive visual servo control of robots. In Pugh, A., editor, *Robot Vision*, 107–116. New York: Springer-Verlag.

van der Smagt, P. (1994). Simderella: A robot simulator for neuro-controller design. *Neurocomputing*, 6(2).

van der Smagt, P. (1995). *Visual Robot Arm Guidance using Neural Networks*. PhD thesis, The University of Amsterdam, Amsterdam, The Netherlands.

Walter, J. A., Martinez, T. M., and Schulten, K. J. (1991). Industrial robot learns visuo-motor coordination by means of neural-gas network. In Kohonen, T., editor, *Artfificial Neural Networks*, vol. 1. Amsterdam.

Watkins, C. J. C. H., and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

Weiss, L. E., Sanderson, A. C., and Neumann, C. P. (1987). Dynamic sensor-based control of robots with visual feedback. *Journal of Robotics and Automation*, RA-3.

Werbos, P. J. (1992). Neurocontrol and supervised learning: An overview and evaluation. In *Handbook of Intelligent Control*, 65–89. New York: Van Nostrand Reinhold.

Whitley, D., Dominic, S., Das, R., and Anderson, C. W. (1993). Genetic reinforcement learning for neurocontrol problems. *Machine Learning*, 13:259–284.

Wijesoma, S. W., Wolfe, D. F. H., and Richards, R. J. (1993). Eye-to-hand coordination for vision-guided robot control applications. *The International Journal of Robotics Research*, 12(1):65–78.

Yamauchi, B. M., and Beer, R. D. (1993). Sequential behavior and learning in evolved dynamical neural networks. *Adaptive Behavior*, 2:219–246.