Neural Processing Letters manuscript No.
(will be inserted by the editor)

Learning Dynamic Obstacle Avoidance for a Robot Arm Using Neuroevolution

Thomas D'Silva · Risto Miikkulainen

Received : 25 March, 2008 / Accepted : 26 January, 2009

Abstract Neural networks can be evolved to control robot manipulators in tasks like target tracking and obstacle avoidance in complex environments. Neurocontrollers are robust to noise and can be adapted to different environments and robot configurations. In this paper, neurocontrollers were evolved to position the end effector of a robot arm close to a target in three different environments: environments without obstacles, environments with stationary obstacles, and environments with moving obstacles. The evolved neurocontrollers perform qualitatively like inverse kinematic controllers in environments with no obstacles and like path-planning controllers based on Rapidly-exploring Random Trees in environments with obstacles. Unlike inverse kinematic controllers and path planners, the approach reliably generalizes to environments with moving obstacles, making it possible to use it in natural environments.

Keywords neural networks, genetic algorithms

1 Introduction

A robot arm consists of a series of mechanical links that are connected by prismatic or rotational joints. An end effector is connected to the end of the arm and used to manipulate objects. Robot arms are used in industrial tasks such as assembling, materials handling and packaging [4].

Traditional approaches in controlling robotic manipulators involve solving the inverse kinematic equations of the arm. The solutions are used to set the joint angles or displacements in order to move the end effector close to a target [5]. One drawback of this approach is that the controller applies only for a specific robot configuration

Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712
E-mail: twdsilva@gmail.com

Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
E-mail: risto@cs.utexas.edu

and environment. If they change, e.g. one of the robot's actuators becomes disabled due to wear and tear, a new controller will have to be developed. For the same reason, inverse kinematic controllers cannot be used to operate a robot manipulator in an environment with obstacles; it cannot take into account the positions of obstacles while controlling the arm. Thus, inverse kinematic controllers are limited to environments that are completely controlled and free of obstacles.

Extensive research has been conducted on designing robot arm controllers for environments with stationary obstacles [3,6,7,16]. In the standard approach, the entire path of the robot arm is planned around obstacles ahead of time [3,5]. However, path planning is computationally expensive because it requires evaluating many paths in order to find one successful one. If the task or the environment changes, for example if the obstacles move, the path planner must compute a new path. It is therefore impractical to use path planning in environments with moving obstacles.

One potential solution is to use neural networks to control the arm. Neural network controllers have been shown to be effective in several challenging tasks such as board games, video games, pole balancing, robot control and vehicle control [1,7,10]. A particularly effective method for constructing neurocontrollers is NEAT, or NeuroEvolution of Augmenting Topologies [10]. In this paper, the NEAT method was used to evolve neural network controllers that learn the dynamics of the robot arm and the environment and learn to avoid both stationary and moving obstacles. Unlike previous approaches, that focused on the end effector only [1,7], the NEAT controllers make sure that no part of the arm touches the obstacles. While doing so, they position the end effector close enough to a target position so that direct control methods can be used to get to the target.

NEAT neurocontrollers were evolved in three different experiments. First, they were evolved in environments without obstacles, demonstrating that neuroevolution can solve the inverse kinematics of a robot arm. Second, controllers were evolved with stationary obstacles to demonstrate that the approach works also in complex environments where inverse kinematic controllers cannot be applied. Third, they were evolved in environments with moving obstacles where path-planning methods are not practical.

The evolved neurocontrollers could position the end effector on average to within 4.03 cm of a target in environments without obstacles, within 12.30 cm of targets with stationary obstacles and within 13.39 cm of a target with moving obstacles. In each case, the end effector is clear of obstacles and close enough to the target so that direct control methods can be used to get to the target.

This result demonstrates that the evolved neurocontrollers effectively solved the robot arm control task. Moreover, they can be used in dynamic environments (with obstacles that move) where inverse kinematic controllers and path planners cannot. They therefore make it possible to extend the domain of robotic manipulators from artificially controlled environments to natural environments.

The rest of this paper is organized as follows. The next section summarizes existing methods used to control a robot arm and their limitations and describes the NEAT neuroevolution method. Section 3 describes the experiments conducted on a robot arm simulator in detail. Experimental results are discussed and future work outlined in Section 4.

2 Background

Standard approaches to robot manipulator control involve designing controllers for a specific robot arm configuration and a specific environment [4,5]. Controllers are developed that solve the inverse kinematics of the arm. First, a visual observation of a target object is translated into the desired position of the robot arm's end effector by using computer vision methods. The inverse kinematic controller is then used to calculate the state that achieves the desired end effector position. The controller drives the motors to position the end effector close to that target joint state [5].

Inverse kinematic controllers have been used to accomplish tasks such as arm positioning and target tracking. For example Feddema and Lee [2], used a self-tuning adaptive controller for performing target tracking for a six-degree-of-freedom robot arm with a camera attached to the end effector. Past image observations and past control inputs were used to determine the optimal control input that moved the camera to track an object.

One drawback of using analytical controllers is that they are calibrated for specific environments and robot arm configurations. If the robot configuration changes or the environment changes the controller cannot be used and the inverse kinematics equations have to be solved again.

These limitations can be overcome by using supervised learning methods to train the controller. If the robot configuration changes, the controller has to be retrained but this can be done with little human effort. Supervised learning algorithms require a training set that demonstrates the correct robot joint states for different situations. One way to generate training examples is to move the arm randomly while recording the joint angles and end effector positions [18]. One limitation of this approach is that it is difficult to generate training examples for complex behaviors like obstacle avoidance.

Obstacles can be taken into account through path planning. For example, Spong and Vidyasagar [5], described an algorithm that uses attractive and repulsive potential fields to generate a path that avoids obstacles. Henrich et al.,(1998) developed a path planner based on the A^* search algorithm in an implicitly represented configuration space. Tian and Collins [16], used a genetic algorithm to optimize the parameters of a trajectory for a planar two-link robot arm, Shibata et al. [9], used a similar approach to optimize the path of a robot arm with six degrees of freedom. These implementations show that path-planning in general is a successful approach. However it is computationally expensive, which makes it difficult to use in real time. In particular, it is too slow to be applied continuously when the robot is in an environment where the obstacles are constantly changing, or moving.

For this reason methods have been developed based on neural network controllers. Neural networks are fast, robust to noise, and can generalize even in non-linear environments. For example, Moriarty and Miikkulainen [7] developed a method based on the SANE genetic algorithm to evolve controllers for a three-degree-of-freedom robot arm in a simulated environment with obstacles. This result demonstrated that neurocontrollers could in principle use obstacle and target information from sensors to effectively navigate around obstacles in real time. However, the obstacles were avoided only at the end effector, and there was only a single stationary obstacle in one of twelve positions. Scaling up this result to avoid obstacles with the entire arm, and further to avoid obstacles that are moving, is a challenging task. In this paper, the NeuroEvolution of Augmenting Topologies (NEAT) [10,11] method is used to evolve neurocontrollers that meet this challenge. The NEAT method has previously been shown effective in challenging control tasks such as pole balancing, robot control, vehicle control, board games and video games [10–12, 14, 15]. Many of these domains share features with the obstacle avoidance task, suggesting that NEAT should be successful in this task as well.

In NEAT, both the connection weights and the network topology of increasingly complex neural networks are evolved to match the complexity of the problem. NEAT is based on three fundamental principles: (1) a principled method of crossover of different topologies, (2) protecting structural innovation through speciation, and (3) incrementally growing networks from a minimal structure.

A genome in NEAT consists of *node genes* that specify whether the node is input, output or hidden, and a list of *connection genes* that specify the input node, output node, connection weight, whether the connection is enabled or not, and an *innovation number*. Every time a new connection gene is created, a global innovation number is incremented and assigned to the new connection. NEAT has two mutation operators: *add connection*, where a single new connection gene is added to two previously unconnected nodes, and *add node*, where an existing connection between two nodes is disabled and replaced with two new connections (with the two old nodes connected to the new node). Innovation numbers allow networks of different topologies to be combined during crossover. During crossover the genomes of both parents are compared and genes with the same innovation number are inherited by the offspring genome. Genes that do not match are inherited from the fitter parent, or if the parents are of equal fitness, from both parents randomly. Innovation numbers allow NEAT to perform crossover without expensive topological analysis.

Adding nodes and connections initially decreases the fitness of the network, so speciation is used to protect newly augmented networks. NEAT clusters individual genomes into species so that individuals compete within their own species rather than against the entire population. The distance measure of two genomes is computed as a weighted sum of the average weight differences of matching genes and the number of genes that do not match. Genomes are compared one at a time with a randomly chosen member of the species from the previous generation and are placed into the species if the distance measure is less than a threshold. Speciation protects structural innovation by allowing individuals to compete within their own niches.

NEAT begins evolution with an initial population of networks with no hidden nodes and weights that are randomly initialized. New structures are introduced as evolution proceeds, and only networks with high fitness are used to generate new individuals for the next generation. Mutation grows the structures to the complexity needed to solve the problem. Such a process of starting minimally and complexifying as necessary makes it possible to find solutions faster, and find more complex solutions [11, 14]. NEAT should therefore be able to solve more complex versions of the robot arm control task than has been possible before.

3 Experiments

Neural network controllers were evolved that could place the end effector of the simulated OSCAR-6 robot arm near a target object in an environment that could contain stationary or moving obstacles. Figure 1 shows the arm as seen in the Simderella 3.0 simulator [17].

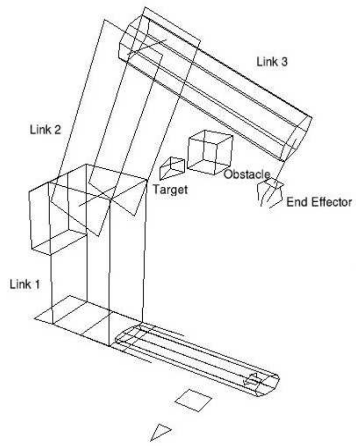

Fig. 1 *A three-joint OSCAR-6 robot arm with six degrees of freedom, simulated using Simderella [17].* Links 1, 2 and 3 are 46cm, 51cm and 50cm long. The joints between links 1 and 2, links 2 and 3, and link 3 and the end effector each have one degree of freedom, i.e. the angle between them. Additionally, link 1, link 3 and the end effector can rotate around their main axis for a total of six degrees of freedom. The neurocontrollers use range sensors along the arm and the end effector to sense the obstacles and the target. The neurocontrollers' outputs are used to control the rotation of link 1 and the angles between links 1 and 2 and links 2 and 3; the other degrees of freedom are not used. The target is placed in the reachable space of the robot arm, and the obstacle is centered around the midpoint of the line joining the end effector and the target so that the robot has to avoid obstacles for every training example.

The first and second joints have one degree of freedom each while the third joint and the end effector have two degrees of freedom. The neural networks were trained to control three of these, i.e. the angle of rotation of the first link, and the angles between links 1 and 2 and links 2 and 3. These three degrees of freedom were sufficient to allow the controller to reach target positions in the space directly in front of it. Reducing the degrees of freedom allowed neuroevolution to learn to solve the inverse kinematics of the arm in a fewer number of generations.

The objective was to evolve neurocontrollers that could position the end effector within 15 cm of a target, i.e. close enough so that the arm could thereafter be moved directly to the target using a direct controller (e.g. one based on inverse kinematics). This goal was achieved in three steps. First, neurocontrollers were trained to reach the target in environments without any obstacles. Second, they were trained to do the same in environments with stationary obstacles. Third, they were trained to do the same with moving obstacles.

The target set was chosen so that all targets were within the robot's reach space. The obstacles were placed directly on the path between the end effector and the target so that the neurocontroller would have to learn to move the arm around them. Performance of the best neurocontrollers were compared with an inverse kinematic controller for environments without obstacles and with a path-planning controller based on Rapidly-Exploring Random trees (RRT; LaValle,1998).

3.1 Learning to control the arm in an environment without obstacles

The objective of the first experiment was to evolve neurocontrollers that could solve the inverse kinematics of the robot arm. The neural network is provided with sensor inputs from the environment, which are then used to determine how to move the arm. The inputs consist of:

- three joint angles that represent the current joint state;
- x, y and z positions of the target in absolute coordinates; and
- x, y and z positions of the target relative to the end effector.

The networks have four output neurons, three of which determine how much each rotation or joint angle changes at each timestep. The angles are thresholded between $[-5, +5]$ degrees, which forces the neurocontroller to make several small joint rotations towards the target. In preliminary work, this approach was found to be more effective than specifying the final angle directly. The networks also have an output neuron that controls whether or not to stop moving the robot. This representation was found to be more effective than having to set all three joint angle rotations to zero degrees.

Each neural network evaluation starts by resetting the robot arm to a standard initial configuration, as depicted in Figure 1. Each network is evaluated over a fixed training set of 168 target positions uniformly randomly distributed within the robot's reach space. During each evaluation, the network is allowed to move the arm until the network stops the arm by activating the stop neuron, or the number of timesteps reaches 20.

The fitness function is computed as the percentage of distance the arm moved from the initial starting point towards the target position. Percentage is used instead of actual distance so that both near and far targets need to be learned.

Figure 2 depicts the fitness of the best network found at each generation, averaged over ten runs. The fitness reaches 0.88, corresponding to an average distance of 3.79 cm from the target over the 168 position training set. Over a fixed uniformly randomly distributed set of 168 test positions, the networks achieved an average distance of 4.03 cm from the target over the ten runs. In comparison, the inverse kinematic controller is on average able to move the end effector to within 0.12 cm of the target over the test set. However, the target distance of 4.03 cm is close enough so that the robot arm can be directly controlled to move the end-effector to the target the rest of the way. In other words, the neurocontroller effectively solved the inverse kinematics of the robot arm.

3.2 Learning to control the arm in an environment with stationary obstacles

In the second experiment, the neural network controllers were trained to move the end effector close to a target position while avoiding stationary obstacles.

The networks receive input from sensors along the robot arm, providing information about the target and the obstacle. These sensors allow the neurocontroller to avoid hitting the obstacle with the entire arm and not just its end effector. The inputs consists of

- three joint angles that represent the current joint state;
- x, y and z positions of the target relative to the end effector;

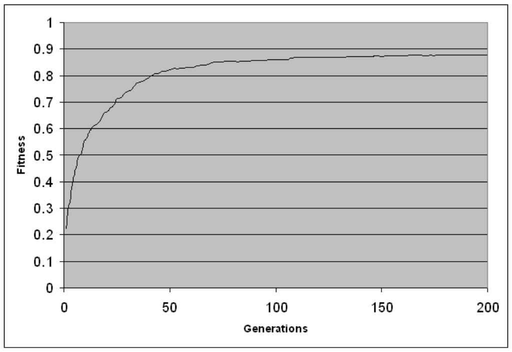

Fig. 2 *Fitness of the best network found at each generation in an environment without obstacles averaged over ten runs.* The fitness of the starting network is low because it consists of inputs connected directly to outputs with random weights. The fitness eventually reaches a maximum of 0.88, meaning that the controller can position the end effector to within 3.79 cm of a target on average over the training set, and 4.03 cm over the test set.

- x, y and z positions of the obstacle relative to the end effector;
- x, y and z positions of the obstacle relative to the midpoint of the second link; and
- x, y and z positions of the obstacle relative to the midpoint of the third link.

Each neural network evaluation starts from the same standard joint configuration. Each network is evaluated over a fixed training set that consists of 96 target positions uniformly randomly distributed in the robot's reach space. The target is placed in one of those positions and an obstacle is placed midway between the initial position of the end effector and the target position. To get to the target, it is therefore necessary for the controller to navigate around the obstacle.

During each evaluation the network is allowed to move the arm until the network stops the arm, or the number of timesteps reaches 60, or the robot hits the obstacle.

The fitness function is the relative distance travelled towards the target, as in the previous experiment. If any part of the arm hits the obstacle the network is assigned a fitness of zero. This fitness function rewards networks that are able to navigate around obstacles and move the end effector close to a target using an efficient path.

Figure 3 depicts the fitness of the best network found at each generation, averaged over ten runs. The final fitness is 0.68 on average, which corresponds to a final target distance of 11.12 cm over the training set, and 12.30 cm over the 96-position test set (if the neurocontroller hit an obstacle, the initial distance is used in the average calculation). This distance is again close enough to the target for a direct control method to take care of the rest. On average the final controllers successfully navigate around obstacles in 87.1 of 96 test cases. In other words, the neurocontroller is able to avoid obstacles and still position the end effector close to a target.

An inverse kinematic controller cannot be used in environments with stationary obstacles because it does not take into account the position of the obstacle. The final evolved neurocontrollers were therefore compared with a path-planning controller. In order to make it feasible for the path planner to avoid obstacles with the entire arm, it was based on Rapidly-exploring Random Trees (RRT; LaVelle 1998). An RRT is a data structure that allows searching nonconvex high-dimensional spaces efficiently.

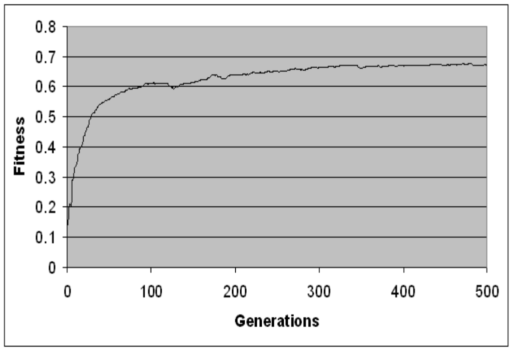

Fig. 3 *Fitness of the best network found at each generation in an environment with stationary obstacles, averaged over ten runs.* The final network had a fitness of 0.68, corresponding to a final distance of 11.12 cm over the training set and 12.30 cm over the test set. This result demonstrates that neurocontrollers can be used to effectively control the arm in environments with stationary obstacles.

With the RRT implementation in the Motion Strategy Library [19] the path planners avoided stationary obstacles along the entire length of the arm while positioning the arm close to a target.

An example case is shown in Figure 4. Both the neurocontroller and the RRT controller are able to avoid the obstacle and position the arm close to the target. On average the RRT controller gets the end effector to within 0.29 cm of a target over the 96-position test set. The average final distance of the neurocontroller is somewhat larger, although still practical. This approach is especially useful because the same approach, unlike the path-planning approach, can be used in more complex environments with moving obstacles, as described in the next section.

3.3 Learning to control the arm in an environment with moving obstacles

In the third experiment, the neural network controllers were trained to move the end effector close to a target position while avoiding a moving obstacle. The final neurocontrollers in the previous experiment were used as a starting point for evolution.

The experimental setup is otherwise the same as before, except the obstacle moves back and forth across the midpoint between the end effector and the for a distance of 80 cm at a speed of 2.67 cm per timestep. The neurocontroller must learn to move towards the target while the obstacle is not in the way.

Figure 5 depicts the average fitness of the best network found at each generation. The final networks in the ten runs had an average fitness of 0.71, which represents a final target distance of 12.29 cm over the training set and 13.39 cm over the 96-position test set. The final neurocontrollers were able to avoid 82.9 of the 96 obstacles in the test set on average.

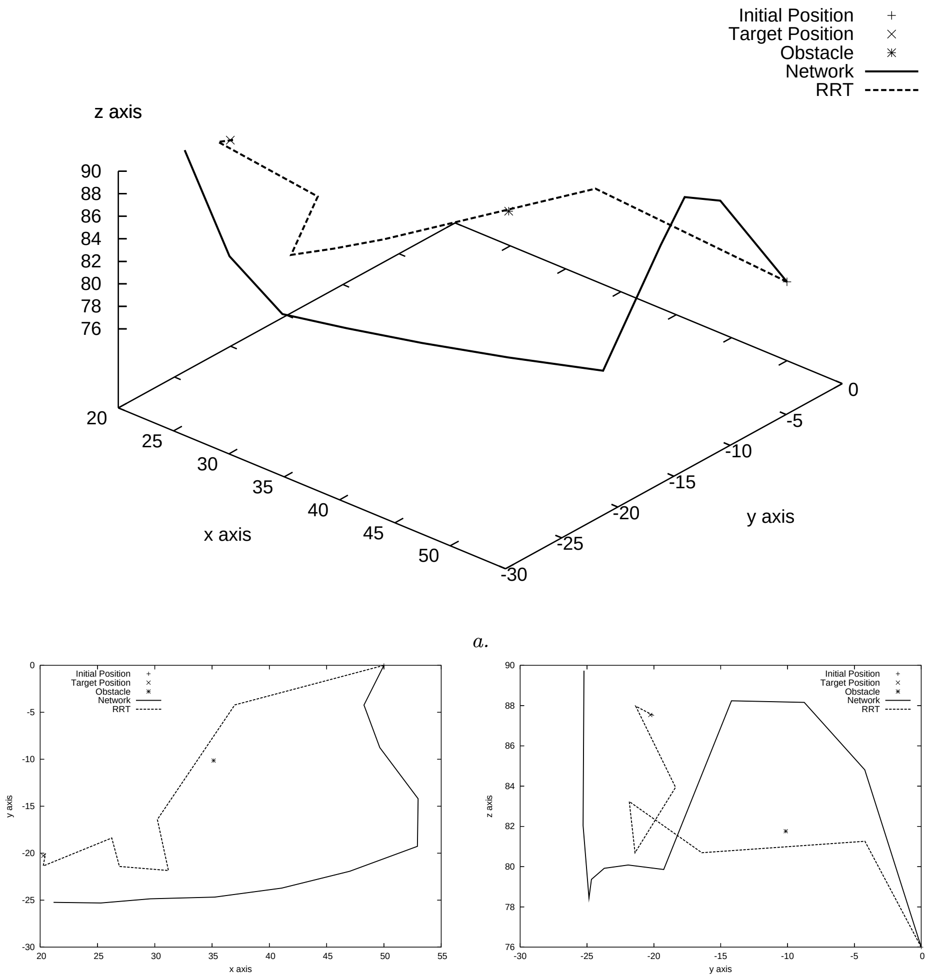

Fig. 4 *An illustration of the path taken by the end effector in an environment with stationary obstacles for a neurocontroller and a RRT controller.* The obstacle is placed at the midpoint of the line joining the starting position and the target position. (a) A 3D visualization of the path. (b) Projection of the path in the xy plane, and (c) projection in the yz plane. The neurocontroller was trained on target positions that were uniformly randomly distributed in the robot arm's reach space; it is able to generalize and position the effector close to the target in this new case. In contrast, the RRT controller randomly explores the search space to find a path that avoids the obstacle. The two controllers therefore take different paths to the target while avoiding the obstacle.

In summary, the controller constructed through neuroevolution can avoid moving obstacles as well as stationary ones, which is a great advantage over traditional methods. Neurocontrollers trained this way could potentially be used to control manipulators in complex natural environments, which is currently not practical in any other way.

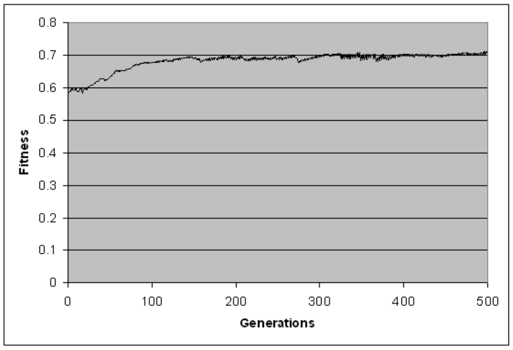

Fig. 5 *Fitness of the best network found at each generation in an environment with moving obstacles, averaged over ten runs.* The final network that was evolved had a fitness of 0.71 on on average, positioning the end effector to less than 12.29 cm from a target in training and 13.39 cm from a target in testing while avoiding moving obstacles. This result demonstrates that neurocontrollers can be used to effectively control the arm in complex natural environments.

4 Discussion and Future Work

The evolved neurocontrollers achieved qualitatively similar performance as the inverse kinematic controller in environments without obstacles, and the path-planning controller in environments with stationary obstacles. This result demonstrates that the neurocontrollers learned the inverse kinematics of the robot arm and learned to effectively use sensor information to get around obstacles.

The neurocontrollers can move the end effector to within 4.03 cm of the target in an environment without obstacles, 12.30 cm with stationary obstacles, and 13.39 cm with moving obstacles, without touching the obstacle with the end effector or the arm thereof. In each case, the end position is close enough for direct controllers to take over and do the rest; the networks have already performed the hard part of the task. In particular, while path planners can be used to control the arm in environments with stationary obstacles where there are no time constraints, to date there is no other practical method to control it in an environment with moving obstacles. This result demonstrates that the neurocontrollers could potentially be used to perform complex tasks in the natural world in the future.

The neurocontrollers described in this paper do not change once they have been evolved. It would be desirable to automatically detect changes in the robot arm configuration or environment and adapt to it dynamically while the system is performing. For example, the robot arm configuration may change if one of its actuators become defective due to wear and tear, or the environment can change if new obstacles are introduced.

One possible approach would be to use a self-teaching architecture similar to that of Nolfi and Parisi [8]. In this architecture, the training input is provided by a separate neural network that has the same sensor input as the neurocontrollers and is subject to the same evolutionary process as the neurocontroller, but with a fitness function that is designed to train the network to detect environmental changes. In the robot arm application, the teaching network could recognize the configuration of the environment and robot and modify its outputs in order to train the neurocontroller to adapt to the new environment. Evolving such adaptable neurocontrollers is an important direction for future work.

5 Conclusion

Neurocontrollers were evolved for environments without obstacles to demonstrate that the NEAT genetic algorithm can be used to learn the inverse kinematics of the robot arm. In environments with stationary obstacles, the neurocontrollers' performance was qualitatively similar to that of path planning controllers. Furthermore, in environments with moving obstacles, where path-planning methods are not practical, evolution was able to find effective solutions. A robot arm controlled in this manner should therefore be able to operate robustly in a natural environment with moving obstacles, which is currently not otherwise possible.

Acknowledgements The authors would like to thank Kenneth Stanley for making the NEAT code available, Patrick van der Smagt for making the Simderella code available and Steve LaValle for making the Motion Strategy Library available. This research was supported in part by NSF under grant EIA-0303609.

References

1. T. Buehrmann, E. D. Paolo (2004) Closing the loop: Evolving a model-free visually-guided robot arm. Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems, 9:63–68
2. J. T. Feddema, G. C. S. Lee (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):1172–1183
3. D. Henrich, C. Wurll, H. Worn (1998) On-line path planning by heuristic hierarchical search. The 24th Annual Conference of the IEEE Industrial Electronics Society, 4:2239–2244
4. F. L. Lewis, D. M. Dawson, C. T. Abdallah (2003) Robot Manipulator Control. CRC, New York
5. S. H. M. Spong, M. Vidyasagar (2006) Robot Modelling and Control. Wiley, Hoboken
6. Z. Mao, T. C. Hsia (1997) Obstacle avoidance inverse kinematics solution of redundant robots by neural networks. Robotica, 15:3–10
7. D. E. Moriarty, R. Miikkulainen (1996) Evolving obstacle avoidance behavior in a robot arm. From Animals to Animats: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, 4:468–475
8. S. Nolfi, D. Parisi (1997) Learning to adapt to changing environments in evolving neural networks. Adaptive Behavior, 5:75–98
9. T. Shibata, T. Abe, K. Tanie, M. Nose (1997) Motion planning by genetic algorithm for a redundant manipulator using a model of criteria of skilled operators. Information Sciences, 102:171–186
10. K. O. Stanley (2004) Efficient Evolution of Neural Networks through Complexification. PhD thesis, Department of Computer Science, University of Texas at Austin
11. K. O. Stanley R. Miikkulainen (2002) Evolving neural networks through augmenting topologies. Evolutionary Computation, 10(2):99–127
12. K. O. Stanley, B. D. Bryant, R. Miikkulainen (2005) Real-Time Neuroevolution in the NERO Video Game. IEEE Transactions on Evolutionary Computation, 9(6):653–668
13. K. O. Stanley, R. Miikkulainen (2004) Evolving a Roving Eye for Go. Proceedings of the Genetic and Evolutionary Computation Conference. 3103:1226–1238
14. K. O. Stanley, R. Miikkulainen (2004) Competitive Coevolution Through Evolutionary Complexification. Journal of Artificial Intelligence Research, 21:63–100

15. N. Kohl, K. O. Stanley, R. Miikkulainen, M. Samples, R. Sherony (2006) Evolving a Real-World Vehicle Warning System. Proceedings of the Genetic and Evolutionary Computation Conference, 1681–1688
16. L. Tian C. Collins (2004) An effective robot trajectory planning method using a genetic algorithm. Mechatronics, 14:455–470
17. P. van der Smagt (1994) Simderella: A robotic simulator for neuro-controller design. Neurocomputing, 6(2)
18. P. J. Werbos (1992) Neurocontrol and supervised learning: An overview and evaluation, Van Nostrand, New York
19. S. LaValle (1998) Rapidly-exploring random trees: A new tool for path planning, TR 98-11, Computer Science Department, Iowa State University
20. S. LaValle (2009) Motion Strategy Library, http://msl.cs.uiuc.edu/msl