Copyright
by
Thomas William D'Silva
2006

Evolving Robot Arm Controllers Using the NEAT Neuroevolution Method

by

Thomas William D'Silva, B.S.

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2006

Evolving Robot Arm Controllers Using the NEAT Neuroevolution Method

APPROVED BY

SUPERVISING COMMITTEE:

Benjamin Kuipers, Supervisor

Risto Miikkulainen, Supervisor

Evolving Robot Arm Controllers Using the NEAT Neuroevolution Method

Thomas William D'Silva, M.S.E.
The University of Texas at Austin, 2006

Supervisors: Benjamin Kuipers
Risto Miikkulainen

Neuroevolution can be used to evolve neural networks that can control robot manipulators to perform tasks like target tracking and obstacle avoidance in complex environments. Neurocontrollers have been successful in the robot control domain because they are robust to noise, they can be adapted to different environments and manipulator configurations, and they can be used to implement controllers that can perform online learning.

The focus of this report was to evolve neurocontrollers for two environments. First, neurocontrollers are evolved for environments without obstacles and their performance is compared to an inverse kinematic controller and a potential field controller. Second, neurocontrollers are evolved for environments with obstacles and compared with a controller that uses potential fields to implement a path planning algorithm. The neurocontrollers evolved in this report come close to matching the performance of the analytical controllers. The advantage of using neurocontrollers is their robustness to noise and ability to adapt to different environments.

Table of Contents

Abstract **iv**

List of Tables **vi**

List of Figures **vii**

Chapter 1. Introduction **1**

Chapter 2. Background and Related Work **3**

2.1 NEAT genetic algorithm 6

2.2 Robot Arm Kinematics 8

2.2.1 Forward Kinematics 8

2.2.2 Inverse Kinematics 10

2.3 Path Planning algorithm for environments with obstacles 12

Chapter 3. Experiments **14**

3.1 Learning to control the arm in an environment without obstacles . . . 15

3.2 Learning to control the arm in an environment with obstacles 22

Chapter 4. Discussion **29**

Chapter 5. Conclusion **32**

Index **34**

Bibliography **35**

Vita **37**

List of Tables

2.1 Denavit Hartenberg parameters for OSCAR 6 anthropomorphic arm. 11

3.1 Comparison of Neural Network controller and Inverse Kinematic Controller 21

3.2 Comparison of neural network ontroller and potential field controller . 21

3.3 Comparison of neural network ontroller and Path Planning controller. 27

List of Figures

2.1 The three joint, six degree of freedom robot arm which is used to evaluate the neural network controllers. 10

3.1 Three joint six degree of freedom OSCAR-6 robot arm which is simulated using Simderella. 15

3.2 Configuration of the networks evolved in an environment without obstacles. 17

3.3 Fitness of the best network found at each generation in an environment without obstacles averaged over three runs. 18

3.4 Fitness display of best network that was evolved for each point in the training set in an environment without obstacles. 19

3.5 The three joint angles for the analytical and neural network controller for a single target position. 20

3.6 Comparison of the path taken by the end-effector for a neural network controller and a controller that uses a potential field. 22

3.7 Three obstacle sensors are used to navigate around obstacles while moving the end-effector towards the target. 23

3.8 Configuration of the networks evolved in an environment with obstacles. 24

3.9 Fitness of the best network found at each generation in an environment with obstacles averaged over five runs. 26

3.10 Fitness display of best network that was evolved for each point in the training set in an environment with obstacles. 27

3.11 Comparison of the path taken by the end-effector for a neural network controller and a controller that uses path planning. 28

Chapter 1

Introduction

The goal of this report is to present a method that can evolve robust adaptable controllers for robot manipulators that can operate in environments with obstacles. The robot manipulators are used to pick up or track objects while avoiding obstacles. In order to successfully accomplish this task a controller must be able to incorporate information about obstacles into a control algorithm that generates a trajectory that positions the end-effector close to a target.

Traditional approaches in robotic control involve solving the inverse kinematic equations of a robotic manipulator in order to develop controllers that can move the robot arm. One drawback of this approach is that the controllers developed are for a specific robot configuration. If the robot's configuration or environment changes then the controller will have to take into account the new robot kinematics. An inverse kinematic controller cannot be used to operate a robot manipulator in an environment with obstacles. Such a controller must be modified to use a path planning algorithm to be able function in environments with obstacles.

Neural network controllers have been shown to be effective in different control tasks such as pole balancing, robot control and vehicle control [9, 10]. This report uses the Neuroevolution of Augmenting Topologies (NEAT) [9] method to evolve neural network controllers that can be adapted to different environments and robot configurations. The controllers are first evolved in environments without obstacles to demonstrate that neurocontrollers can solve the inverse kinematics of a robot arm. Then, controllers are evolved in environments with obstacles so that they can be used in complex environments to control robot manipulators.

The robot arm has sensors along its joints which are used to sense its distance from obstacles. The arm has a range sensor present at the end-effector which provides the relative distance of the target. The neurocontroller is provided with this sensor input and generates a set of joint angles that will move the end-effector to the target position.

Neural networks were evolved that could position the end-effector to within 5cm of a target in environments without obstacles and within 10cm in environments with obstacles. The performance of the best neural network that was evolved for an environment without obstacles was compared to an inverse kinematic controller and a potential field controller. The performance of the best network that was evolved for environments with obstacles was compared to a controller that uses path planning.

The evolved neurocontrollers come close to matching the performance of the inverse kinematic controller and potential field controller for environments without obstacles. The advantage of using neural network controllers is that the same training process can be used for different robot arm configurations.

For environments with obstacles, the neurocontrollers come close to matching the performance of controllers that use path-planning to avoid obstacles. Path-planning algorithms do not guarantee an optimal path to an object. The evolved neurocontrollers are able to function in environments with different target and obstacle configurations because the target and obstacle sensors are egocentric and not based on global coordinates.

The main contribution of this report is a method that can be used to evolve neurocontrollers for a robot arm that be adapted for different robot and environment configurations. The neurocontrollers can perform complex tasks like controlling the arm in the presence of obstacles and also track moving objects.

Chapter 2

Background and Related Work

Standard approaches in robotics for robot manipulator control involve designing controllers for specific robot arm configurations. Controllers are developed that solve the inverse kinematics of a robot 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. Then the analytical controller is used to calculate the joint state that achieves the desired end-effector position. The controller then sets the joint state which is used to drive the motors which positions the end-effector close to a target [5].

In order to design controllers that can operate in environments with obstacles, path planning algorithms are used to generate a path for the end-effector that navigates around obstacles towards a target position. This sequence of end-effector positions is converted to joint angles by using an analytical controller. The joint angles are used to set to the gains for the actuators in order to move the joints [5].

Inverse kinematic controllers have been used to accomplish tasks such as arm positioning, obstacle avoidance and target tracking. Feddema and Lee [2] used a self-tuning adaptive controller for performing target tracking for a 6 degree of freedom robot arm with a camera attached to the end effector. The model predicted the target position based on past observations and then used this prediction to move the end effector to track the target. A geometric model of the camera was used to determine the linear differential transformation from image features to camera position and orientation. The self-tuning controller is used to adjust for modelling errors and system nonlinearities and for optimal control.

Another system that used an inverse kinematic controller to track objects was implemented by Papanikolopoulos and Khosla [8] who used the sum of squared differences of optic flow vectors to compute a vector of discrete target displacements. This vector was fed into an adaptive controller that created commands for a robot control system. The controller required only partial knowledge of the relative distance of the target with respect to the camera. This approach incorporates the target's dynamics and kinematics in the system model as opposed to decoupling the problems of obtaining information about the target using computer vision methods and then moving the robot to a target position using inverse kinematic methods.

Weiss[4] used image-based visual servo control on simulation studies of two and three degree of freedom robot arms. Vision-based sensors are used to estimate the target position relative to the end-effector of the robot arm. A model reference adaptive controller is used in a stable closed loop dynamical response system. Joint angles are set using inverse kinematics controllers.

One drawback of using analytical controllers is that they are calibrated for specific environments and robot arm configurations. The internal model has to be re-calibrated for different environments. As robot arms become more sophisticated they will have a larger number of degrees of freedom. It is difficult to solve the inverse kinematic equations for robot manipulators with a large number of degrees of freedom.

In order to solve this problem researchers have applied supervised learning methods to train systems that learn to control a robot manipulator effectively [12]. These methods require a training set that demonstrates the correct robot joint state for a different situations. The training set must be carefully chosen so that the controller learns general behavior that allows it to effectively control the arm in situations that are not present in the training set. One way to generate training examples is to randomly move the arm while recording the joint angles and end-effector positions [12]. After the controller has learned to move the arm to targets

in the training set, it is evaluated on a different test set to measure its performance. One limitation of this approach is that it is difficult to generate training examples for complex behaviors like obstacle avoidance. Another drawback is that random joint movements might not generate training examples that demonstrate how to move the arm correctly.

Another approach to learning to control a robot manipulator is to use exploratory methods. These methods provide the robot with a set of exploratory behaviors. Stoytchev[11] developed a system where the robot controller learns affordances during a behavioral babbling stage where the robot randomly chooses different exploratory behaviors, applies them to objects and detects sensor invariants. The exploratory method was used to learn to position the end-effector in order to pick up objects of different shapes like H frames, π frames, sticks and dumbells. A shortcoming of this approach is that there are affordances that cannot be discovered because the robot does not possess the required exploratory behavior.

Researchers have developed systems that use neural network controllers for robot arm control. Neurocontrollers have been successful in the robot arm control task because they are robust to noise and can model non-linear systems. Vision-based robot arm control is a complex task that requires mapping target positions to a set of joint angles. It is difficult to use supervised learning to train controllers for complex vision based control tasks like controlling an arm in situations where the robot configuration and environment changes.

One example of a robot system that used neurocontrollers was developed by Behrman and Di Paolo[1] who used a genetic algorithm for a three degree of freedom robot arm. The system used an overhead camera with two degrees of freedom, an end-effector camera with two degrees of freedom and a two-dimensional array of laser range finders arranged in a rectangular grid at the end-effector. The rays originate at the same focal point in space and the angle between them determines the camera's field of view. Three different neuro-controllers were evolved for the overhead camera,

the end-effector camera and the robot joints. The evolved controllers could position the end-effector to track objects.

Another system was developed by Moriarty[6] who used the SANE genetic algorithm to evolve controllers for a three degree of freedom robot arm in a simulated environment with obstacles. The robot arm had range sensors that provided the target position relative to the end-effector and one obstacle sensor in the end-effector that could be used to avoid obstacles. The controllers evolved could avoid obstacles only at the end-effector. During normal operation the controllers had to avoid obstacles only 11% of the time. The obstacle was always in one of twelve positions which limited the obstacle avoidance behaviors that were learned.

This report evolves neurocontrollers using the NEAT method that can function in environments with obstacles. The NEAT methods has been used to evolve neurocontrollers that have been shown to be effective in robot control tasks such as pole balancing, robot control and vehicle control [9, 10]. Unlike previous approaches [1] the neurocontrollers are able to function in complex environments with obstacles. The neurocontrollers have sensors along the length of the arm which is used to avoid obstacles using the entire arm.

This chapter describes the motivation of using the NEAT method to evolve neurocontrollers. The robot manipulator kinematics for the OSCAR-6 robot arm which was used to evolve neurocontrollers is also discussed. Finally the two analytical controllers that were used as a comparison for the neural network controllers are also described.

2.1 NEAT genetic algorithm

The NeuroEvolution of Augmenting Topologies (NEAT) [9] method evolves increasingly complex neural networks to match the complexity of the problem. NEAT evolves both connection weights and topology simultaneously. It has been shown to

be effective in many applications such as pole balancing, robot control, vehicle control, board games and videogames [9].

NEAT is based on three fundamental principles: (1) employing a principled method of crossover of different topologies, (2) protecting structural innovation through speciation, and (3) incrementally growing networks from a minimal structure. Mating, or the crossing over of genomes of two neural networks of possibly differing structure, is accomplished through innovation numbering. Whenever a new connection between nodes is created through mutation, it is assigned a unique number. Offspring produced with the new connection inherit the innovation number. Whenever networks are crossed over, those genes that have the same innovation number can be safely aligned. Genes of the more fit organism with innovation numbers not found in the other parent are inherited by the offspring as well. Speciation occurs by dividing the population into separate, distinct subpopulations. The structure of each individual is compared dynamically with others and those with similar structure are grouped together. Individuals within a species share the species' overall fitness [3], and compete primarily within that species. Speciation allows new innovations to be optimized without facing competition from individuals with different structures. Networks in NEAT start with minimal structure, consisting only of inputs connected to outputs with no hidden units. Mutation then grows the structures to the complexity needed to solve the problem. Starting this way avoids searching through needlessly complex structures.

The NEAT method was used to evolve neural network controllers that could control the robot arm in environments with obstacles. The ability of the NEAT method to find efficient solutions to complex control problems by evolving networks using speciation was the primary reason it was used to evolve neurocontrollers.

2.2 Robot Arm Kinematics

A robot manipulator consists of a set of links connected by joints. Common joints found in robot arms are revolute (joint described by angle of rotation) and prismatic (joint described by the amount of linear displacement). A robot manipulator with n joints has $n+1$ links. If the joints are numbered 1 to n and the links are numbered 0 to n, then joint i connects link $i-1$ to link i. Joint i is fixed with respect to link $i-1$. When joint i is actuated link i moves. Link 0 (the first link) is fixed and does not move when the joints are actuated.

In order to perform kinematic analysis a coordinate frame $o_i x_i y_i z_i$ is attached rigidly to each link i. The coordinates of each point on link i are constant when expressed in the i^{th} coordinate frame. Coordinate frame 0 is denoted as the inertial or base frame. If A_i is the homogenous transformation matrix that gives the position and orientation of $o_i x_i y_i z_i$ with respect to $o_{i-1} x_{i-1} y_{i-1} z_{i-1}$, then the matrix that expresses the position and orientation of $o_j x_j y_j z_j$ with respect to $o_i x_i y_i z_i$ is called a transformation matrix and is denoted by T_j^i:

$$T_j^i = \begin{cases} A_{i+1} A_{i+2} \ldots\ldots A_{j-1} A_j \quad if \quad i < j \\ I \quad if i = j \\ (T_i^j)^{-1} \quad if \quad i > j \quad . \end{cases} \tag{2.1}$$

2.2.1 Forward Kinematics

The forward kinematics problem is concerned with the relationship between position and orientation of the end-effector given the angles or extensions for the various rotational or revolute joints for the robot. In order to calculate the position of the end-effector with respect to the base (Joint 0), the Homogenous matrix H is used (q_i denotes a single joint variable):

$$H = T_n^0 = A_1 q_1 A_2 q_2 \cdots A_n q_n \quad . \tag{2.2}$$

The Denavit-Hartenberg convention is commonly used for selecting frames of references in robotic applications because it simplifies kinematic analysis. In this convention each homogeneous matrix A_i is presented as a product of four basic transformations:

$$
\begin{aligned}
A_i &= Rot_{z,\theta_i} Trans_{z,d_i} Trans_{x,a_i} Rot_{x,\alpha_i} \qquad (2.3)\\
&= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&\times \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & -c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} c_{\alpha_i} & s_{\theta_i} s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i} c_{\alpha_i} & -c_{\theta_i} s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

The quantities θ_i, a_i, d_i and α_i are parameters associated with link i and joint i which are the joint angle, link length, link offset and link twist. These parameters are obtained by the specific aspects of the geometric relation between two coordinate frames. These parameters are specified in the Denavit-Hartenberg representation of a robot arm manipulator. Figure 2.1 represents the OSCAR 6 robot arm manipulator that was used to evolve neural network controllers. The Denavit-Hartenberg parameters for the OSCAR 6 robot are described in Table 2.2.1.

In order to solve the forward kinematics of the arm, the matrix transformations are used to calculate the position of the end-effector with respect to the base coordinate frame. Here only three of the six degrees of freedom of the robot are used to control the arm. l_1, l_2 and l_3 are d_1, a_2 and d_4 in Table 2.2.1. The variables x, y

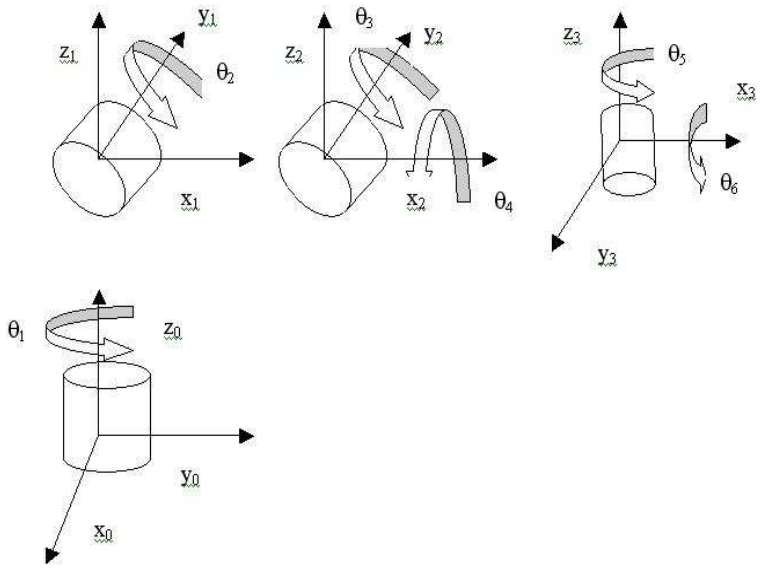

Figure 2.1: *The three joint, six degree of freedom robot arm which is used to evaluate the neural network controllers.* This figure represents the four coordinate frames that are used to solve the kinematics of the OSCAR-6 arm. Joints 1 and 2 have one degree of freedom each, while joint 3 and the end-effector each have two degrees of freedom. While evolving neurocontrollers only the joint angles θ_1, θ_2 and θ_3 are used to control the robot while the remaining three joint angles are held at a constant value.

and z denote the position of the robot's end-effector with respect to the base. The solution to the forward kinematics then becomes:

$$x = (l_2 cos\theta_2 + l_3 sin\theta_3)cos\theta_1 \tag{2.4}$$

$$y = (l_2 cos\theta_2 + l_3 sin\theta_3)sin\theta_1 \tag{2.5}$$

$$z = l_1 cos\theta_2 - l_2 sin\theta_2 - l_3 cos\theta_3 \quad . \tag{2.6}$$

2.2.2 Inverse Kinematics

The inverse kinematics problem deals with finding the joint variables for a given end-effector position and is usually more difficult to solve than the forward

θ	d	a	α
θ_1	46cm	0cm	-90^o
θ_2	0cm	51cm	0^o
θ_3	0cm	0cm	-90^o
θ_4	50cm	0cm	-90^o
θ_5	0cm	0cm	90^o
θ_6	21cm	0cm	0^o

Table 2.1: Denavit Hartenberg parameters for OSCAR 6 anthropomorphic arm.

kinematics problem. The position of the end-effector enables the calculation of the homogeneous matrix H. In order to solve the inverse kinematics a solution has to be found for:

$$T_n^0(q_1, \cdots, q_n) = H \quad , \tag{2.7}$$

where

$$H = A_1 q_1 A_2 q_2 \cdots A_n q_n \quad . \tag{2.8}$$

Equation 2.7 represents 12 non-linear equations in n unknown variables because A is a 4×4 matrix and the bottom row of both T_n^0 and H are (0,0,0,1). This can be written as

$$T_{ij}^0(q_1, \cdots, q_n) = h_{ij}, \quad i = 1, \cdots, 4 \quad . \tag{2.9}$$

It is difficult to solve 12 non-linear equations in n variables. In situations where the robot arm configuration changes, the analytical solution has to be recomputed each time. For complex robot manipulators that have a large number of degrees of freedom n is large and finding an analytical solution is extremely difficult.

The analytical solution to the inverse kinematics of the OSCAR-6 robot arm is

$$\theta_1 = arctan2(p_x, p_y) \tag{2.10}$$

$$\theta_2 = -arctan(\frac{|p_y|}{|p_x|}) - arccos(\frac{x^2 + y^2 + z^2 + l_2^2 - l_3^2}{2l_2\sqrt[2]{x^2 + y^2 + z^2}}) \tag{2.11}$$

$$\theta_3 = -arctan(\frac{|p_y|}{|p_x|}) - arccos(\frac{x^2 + y^2 + z^2 + l_2^2 - l_3^2}{2l_2\sqrt[2]{x^2 + y^2 + z^2}}) \quad . \\ + \frac{3\pi}{2} - arccos(\frac{l_2^2 - l_3^2 - x^2 - y^2 - z^2}{2l_2 l_3}) \tag{2.12}$$

The performance of the best neurocontroller was compared against a controller that implemented an analytical solution of these inverse kinematic equations. The results are discussed in the Section 3.

2.3 Path Planning algorithm for environments with obstacles

A controller that uses path planning moves the arm according to the sum of attractive and repulsive forces that act on the arm. The attractive field grows as the distance of the target from the end-effector increases and is zero when the end-effector is at the target position. Each obstacle generates a repulsive field that grows as the end-effector moves close to an obstacle. If the obstacle is more than ten units away from the end-effector the repulsive force is zero. The net force that acts on the arm is

$$U_{att}(q) = \alpha * T(q) \quad , \tag{2.13}$$

where

$$U_{rep}(q) = \beta * 1/O(q) \tag{2.14}$$

$$U(q) = U_{att}(q) + U_{rep}(q) \quad , \tag{2.15}$$

q represents the joint configuration of the robot at any instant, $U_{att}(q)$ the attractive force due to the target, $U_{rep}(q)$ the repulsive force due to the obstacle, $T(q)$ the distance of the end effector from the target for a given joint configuration, $O(q)$ represents the distance of the obstacle from the target for a given joint configuration, α and β are constants.

In order to generate a path that moves the robot around obstacles towards a target position, the controller calculates the net force at each time step for the current joint configuration and moves the arm based on the direction and magnitude of the net force. This process allows the controller to generate a sequence of joint angles that will move the arm towards the target while navigating around obstacles. One problem with this approach is that the arm often gets stuck in a local minimum where the net force is zero but the end effector is not near the target. If the end effector get stuck in a local minimum a random movement away from the obstacle is taken and the process is repeated. This process continues for a fixed number of time steps or until the robot arm hits an obstacle. The average path taken and the average final distance of the end-effector from the target for the training set is compared with results from neural network controllers.

Chapter 3

Experiments

Experiments were conducted to evolve neural network controllers that could place the end effector of the robot arm near a target object in an environment that may contain obstacles. Figure 3.1 shows the OSCAR-6 robot arm as seen in the Simderella 3.0 robot arm simulator that was used to train the neural network controllers. The first and second joints have one degree of freedom each while the third joint and the end-effector have two degrees of freedom. In the experiments the neural networks were allowed to control three of the six degrees of freedom. Three degrees of freedom was sufficient to allow the controller to move the arm to cover most of the positions in the space directly in front of the robot. This configuration also allowed the neural networks to evolve a controller that could solve the inverse kinematics of the arm in a fewer number of generations.

Neural networks were evolved to learn to control the arm in two different experiments. First, the neurocontrollers were trained to control the arm in environments without obstacles. The performance of the best neurocontrollers was compared with an inverse kinematic controller and a potential field controller. Second, the neurocontrollers were trained to control the arm in environments with obstacles. The performance of the best neurocontrollers was compared with a controller that used path planning to move the arm around obstacles.

The neurocontrollers were trained to position the end-effector close to targets in the space directly in front of the arm. The target set was chosen so that the positions were within the robot's reach space. The obstacles were also placed in this

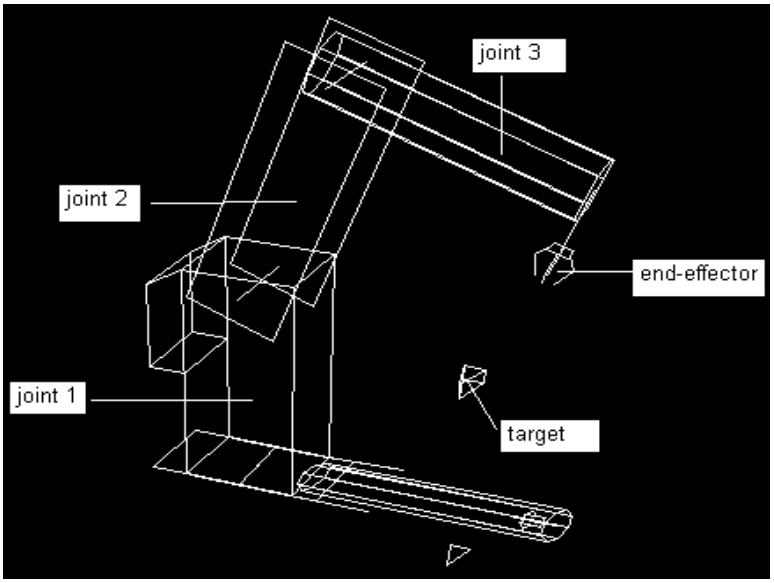

Figure 3.1: *Three joint six degree of freedom OSCAR-6 robot arm which is simulated using Simderella.* The inputs to the neurocontroller are the current joint state (joint angle 1, 2 and 3) and the relative position of the target with respect to the end effector. The neurocontrollers outputs three joint angles which are used to control joints 1, 2 and 3. At each time step the sensor input is presented to the neurocontroller which moves the robot to the target position. Joint 1, 2 and 3 are of lengths 46cm, 51cm and 50cm respectively.

reach space in a manner that made it necessary for the controller to learn to move around obstacles to reach a target.

3.1 Learning to control the arm in an environment without obstacles

The objective of this experiment was to evolve a neural network that could output the joint angles required to position the end-effector close to target objects. In order to evolve neurocontrollers that learn to effectively control the arm, the neural network is provided with sensor input from the environment, which is then used to determine how to move the robot arm. The inputs to the network controllers are the

the current joint angles, and the x, y and z positions of the target relative to the end-effector.

The network controller has three outputs that determine how much each joint angle changes at every timestep. The joint angles are thresholded between [-5,+5] degrees which forces the neurocontroller to make several small joint rotations towards the target which allows it to more effectively control the arm. The networks have an output neuron that controls whether or not to stop moving the robot. This configuration allows the network to stop the robot more easily that having to set all three joint rotation angles to 0^o. Figure 3.2 shows the configuration of the networks evolved to control the arm in an environment without obstacles. The inputs to the network are the current joint angles and the position of the target relative to the end-effector. The outputs are the three joint rotations and a stop arm neuron.

Each neural network evaluation starts by resetting the robot arm to a legal initial configuration. The targets are placed within a 180^o rotation of the robot's first joint. Each network is evaluated over a training set with 168 target positions which are uniformly distributed within the robot's reach space. During each evaluation the network is allowed to move the arm until

1. The network stops the arm by activating the stop neuron, or
2. The number of timesteps exceeds 30.

In order to evaluate a network, the fitness function takes into account the final distance from a target and the path length taken by the end-effector.The fitness function consists of two components:

1. Percentage of distance travelled towards target (TargetDistanceRatio), and
2. Ratio of path length to ideal path length (PathLengthRatio).

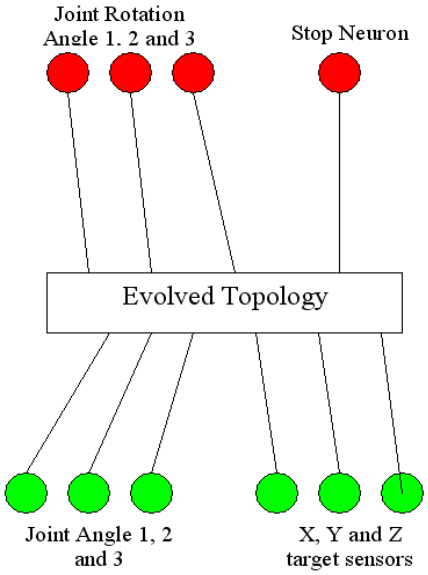

Figure 3.2: *Configuration of the networks evolved in an environment without obstacles.* The inputs to the neurocontroller are the current joint state and position of the target relative to the end effector. The outputs are the joint rotations and the stop signal. The initial starting network consists of inputs connected directly to the outputs with zero weights. As the networks evolve more nodes are added and the weights of the connections are modified so that the network can effectively control the arm.

TargetDistanceRatio is computed as the percentage of distance the arm moved from the initial starting point towards the target position. For example, if the arm started 50 units from the target position, and at the end of the trial, the arm was 25 units from the target position then TargetDistanceRatio is $(50 - 25)/50 = 0.5$. Compared to the final distance from the target, the percentage distance is a more accurate comparison between a network that receives a close target and a network that has to move towards a far-away target.

The ideal path length is the length of the straight line between the initial end-effector position and the target position. PathLengthRatio is computed as the ratio of the ideal length to the path length taken by the end-effector. This component is designed to reward networks that can position the end-effector close to the target

position by taking the shortest path to the target. It also penalizes network controllers that oscillate when they are close to a target while rewarding network controllers that stop when the robot's end-effector is close to the target.

The final fitness function is a weighted sum of the two sub-components:

$$a * TargetDistanceRatio + b * PathLengthRatio \quad . \tag{3.1}$$

In the stationary target experiments $a = 0.6$ and $b = 0.4$.

Figure 3.3 depicts the average fitness of the best network found at each generation with $a = 0.6$ and $b = 0.4$ averaged over three runs. The best network that was found over the 400 generations has a fitness of 0.78. The best neurocontroller could on avereage move to the within 4.43 cm of a target.

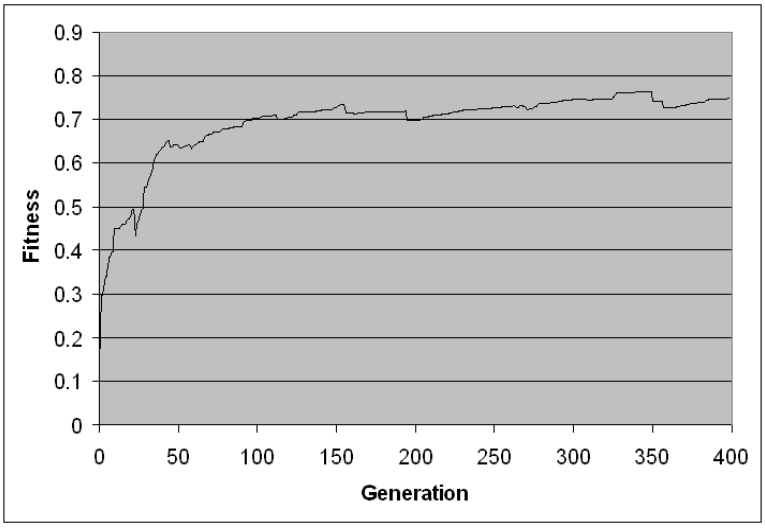

Figure 3.3: *Fitness of the best network found at each generation in an environment without obstacles averaged over five runs.* The best network that was found had a fitness of 0.78. The fitness of the starting network is low because it consists of inputs connected directly to outputs with random weights. As evolution proceeds, the fitness of the best network found at each generation increases.

Figure 3.4 shows how the best network evolved performs on each target position in the training set. The target positions are color coded with yellow having a

fitness greater than 0.8 and red having a fitness less than 0.2. The neural network is able to position the robot arm close to nearly all targets except for some outlying target position that are very high or low (with respect to the z axis).

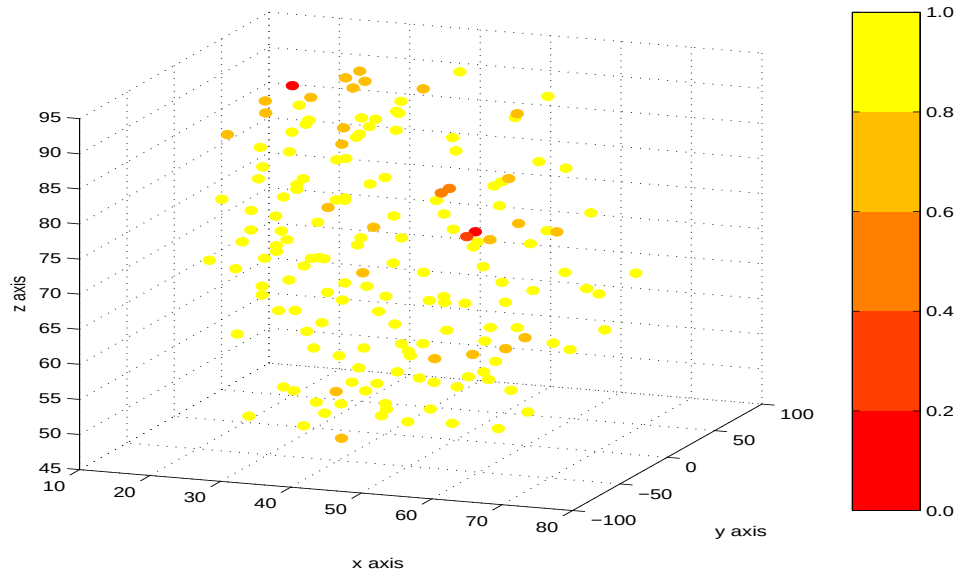

Figure 3.4: *Fitness display of best network that was evolved for each point in the training set in an environment without obstacles.* The points are color coded according to the fitness with yellow being the highest fitness and red the lowest fitness. The fitness of most points in the training set is at least 0.6.

The best evolved network controller was then compared with a controller that solved the inverse kinematics equations to control the arm. The inverse-kinematic controller can compute the required joint angles to position the robot's end-effector close to a target position. Figure 3.5 shows the three joint angles at each time step for the analytical controller and the neural network controller for one target position during an evaluation. The analytical controller is able to position the arm near the target in fewer timesteps compared to the evolved neural network controller. Once the neural network positions the arm close to a target it activates the stop neuron at timestep 17. The final distance of the neurocontroller's end effector from the target is quite close to the inverse kinematics controller.

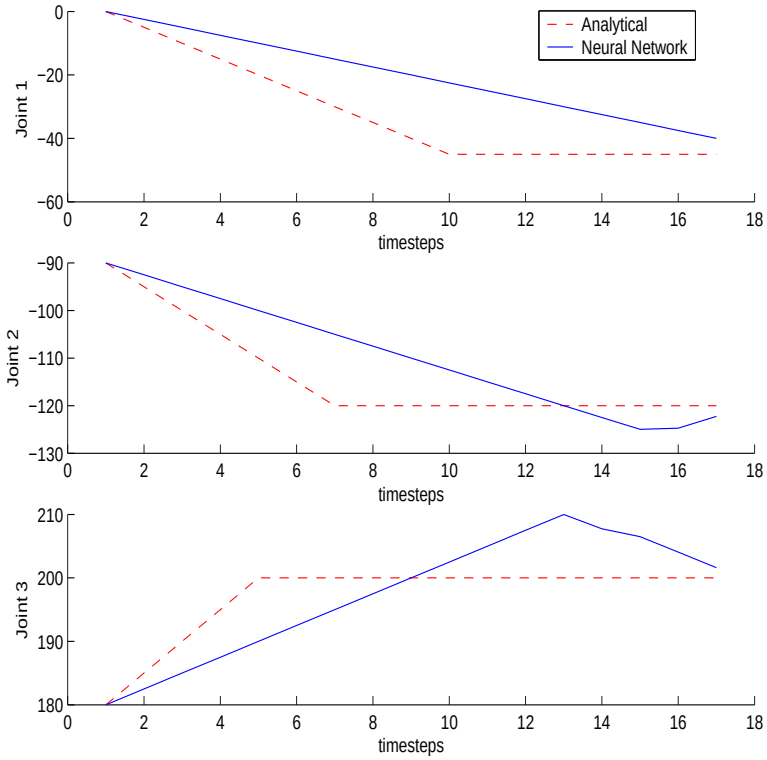

Figure 3.5: *The three joint angles for the analytical and neural network controller for a single target position.* The analytical controller is able to move to the target in 10 timesteps. The neurocontroller also moves to the target but takes 17 timesteps to do so. The neurocontroller is able to move to within 5cm of the target which is close to the inverse kinematic controller's performance.

Table 3.1 shows the average L_1,L_2 and $L\infty$ distances of the joint angles of the best neural network controller and the analytical controller for one target position. For most target positions the neural controller moves moves the robot to the same configuration as the analytical controller, however it takes a larger number of timesteps to reach the correct joint configuration which explains the difference in the average L_1 and L_2 distances of the joint angles.

The best evolved neurocontroller is also compared with a controller that used a potential field algorithm to control the robot arm. In most cases, the potential field

	$Joint1$	$Joint2$	$Joint3$
L_1	6.97	5.19	4.90
L_2	8.49	6.47	6.14
L_∞	15.49	12.20	11.75

Table 3.1: *Comparison of Neural Network controller and Inverse Kinematic Controller* This table shows the L_1,L_2 and $L\infty$ between the best evolved neurocontroller and inverse kinematic controller of joint angles 1, 2, and 3 averaged over the training set. The neurocontroller comes close to matching the inverse kinematic controller in terms of average final distance of the end-effector from the target. The neurocontroller however takes a larger number of timesteps which is why the L_1 and L_2 distances are not zero.

controller takes a shorter, more energy efficient path to the target than the inverse kinematic controller. Table 3.1 shows a comparison of the performance of the best neurocontroller with a potential field controller. The neurocontroller comes close to matching the performance of the potential field controller. Figure 3.6 shows the path taken by the best neurocontroller and the potential field controller for a single target position. The neurocontroller takes a less optimal path, but comes close to matching the final target distance of the potential field controller.

	$AveragePathLength$	$AverageTargetDist$
$NeuralNetworkController$	50.81cm	4.45cm
$PotientalFieldController$	31.04cm	0.84cm

Table 3.2: *Comparison of neural network ontroller and potential field controller* This table shows the average path length and average final distance of the end effector from the target for the best evolved neurocontroller and a potential field controller. The potential field controller takes a shorter path to the target, and moves closer to the target but the neurocontroller comes close to matching this performance.

The best evolved neurocontroller was on average able to position the end-effector to within 4.45cm of a target position which is close to the final target distance using the potential field controller. The evolved neurocontrollers take a relatively short path to a target position. This experiment demonstrates that it is possible to

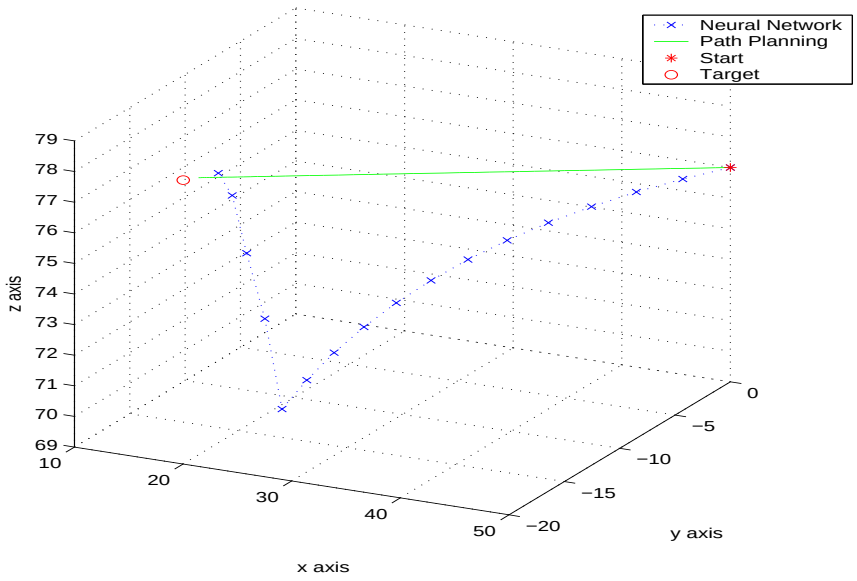

Figure 3.6: *Comparison of the path taken by the end-effector for a neural network controller and a controller that uses a potential field.* The potential field controller takes a shorter path to the target while the neural network controller takes a less optimal path to the target. The average final target distance of the potential field controller is 0.84cm while the neural network controll comes close to this performance with an average distance of 4.45cm.

evolve neurocontrollers that can effectively control the arm in environments without obstacles.

3.2 Learning to control the arm in an environment with obstacles

In the second experiment, the neural network controllers were trained to move the object close to a target position while avoiding obstacles. The inputs to the neural network include the current joint angles, the position of the target relative to the end effector, and three obstacle sensors present in the second joint, third joint and the end-effector as seen in Figure 3.7. These sensors provide the x,y and z relative distances of the closest obstacle. The sensors have a 10cm range and return 1 if there

is no obstacle is within range. If there is an obstacle within range, they return the distance of the nearest obstacle scaled between 0 and 1. The outputs of the controller are three joint thresholds and a stop flag. The three outputs determine how much each joint angle changes at every timestep. The joint angles are thresholded between [-5,+5] degrees. This forces the neurocontroller to make several small joint rotations towards the target which allows it to more effectively sense and avoid obstacles along the arm's path. Figure 3.8 shows the configuration of neural network controllers evolved for obstacle avoidance.

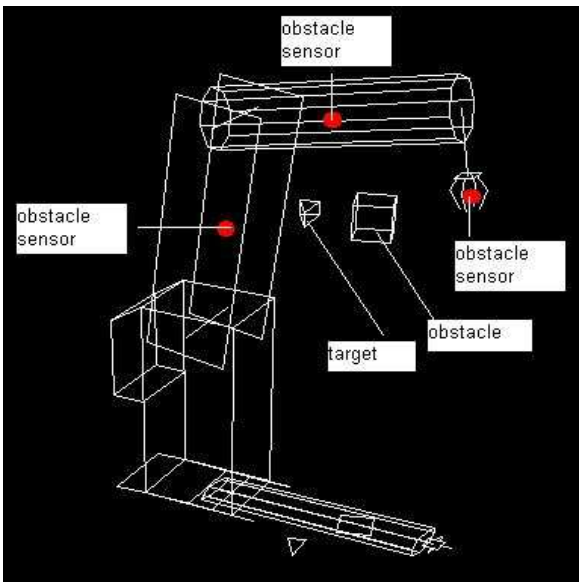

Figure 3.7: *Three obstacle sensors are used to navigate around obstacles while moving the end-effector towards the target.* The inputs are the obstacle sensor data, target sensor data, current joint state and the outputs are the thresholded joint angles and a flag that indicates whether or not to stop moving the arm. The obstacle sensors allow the neurocontroller to sense obstacles along the length of the arm and avoid obstacles along the entire arm. The obstacle is always initially placed between the end effector and target so that the neurocontroller has to perform obstacle avoidance for every position in the training set to get a high fitness. Joint 1, 2 and 3 are of lengths 46cm, 51cm and 50cm respectively.

Each neural network evaluation starts by resetting the robot arm to the same

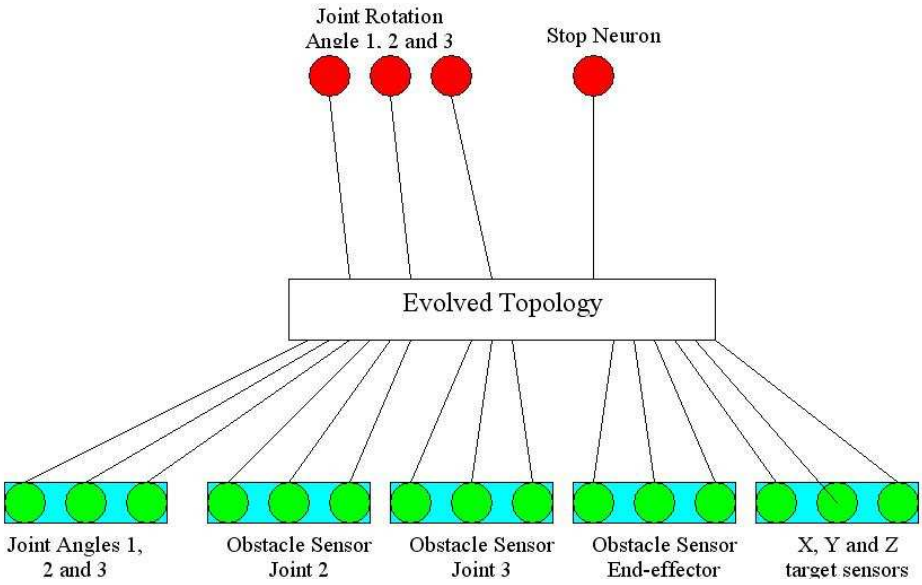

Figure 3.8: *Configuration of the networks evolved in an environment with obstacles.* The inputs to the neurocontroller are the current joint state, position of the target relative to the end effector and obstacle range sensor data. The robot has three obstacle sensors at joint 2, joint 3 and the end effector which returns the thresholded distance of the nearest obstacle. The outputs are the joint rotations and the stop signal. As evolution progress NEAT adds more hidden nodes and modifies the weights so that the network's fitness increases and the neurocontroller can effectively control the arm while avoiding obstacles.

joint configuration. The target is placed within a 180^o rotation of the robot's first joint, and an obstacle is placed at the midpoint of the line between the initial position of the end-effector and the final target position. Placing the obstacle in such a manner makes it necessary for the controller to navigate around an obstacle to reach the target for every position in the training set. The training set consists of 96 target positions that are uniformly distributed in the robot's reach space. During each evaluation the network is allowed to move the arm until

1. The network stops the arm,
2. The number of timesteps exceeds 30, or

3. The robot hits an obstacle.

If during an evaluation the robot arm hits an obstacle at any point along the arm, the trial is stopped and the fitness is set to zero. The fitness function is a weighted sum of the the relative distance travelled towards the target and the path length, similar to the previous experiment. This fitness function rewards networks that are able to navigate around an obstacle and move the end-effector close to a target while taking the shortest path possible.

Figure 3.9 depicts the average fitness of the best network found at each generation. The best network that was found over 400 generations has a fitness of 0.7. For the training set that was used this represents a final target distance that is 10cm, which is reasonably close to the target. The neurocontroller is able to incorporate obstacle avoidance while the controlling the arm while still being able to position the end effector close to targets.

Figure 3.10 depicts how the best network evolved performs on each target position within the training set. As before, the points are color coded with yellow being the highest fitness and red being the lowest fitness. The neural network is able to position the robot arm close to nearly all targets.

The best neural network controller was then compared with a controller that uses a path planning algorithm to move the arm as described in Section 2. The results are shown in Figure 3.11 for a single position in the training set. The path planning controller first moves directly to the target but then hits a local minimum because of the obstacle's repulsive force. It then makes a random move away from the obstacle and then moves to the target. The neurocontroller also avoids the obstacle while moving to the target. This result is shown quantitatively in Table 3.2. The average final distance of the neural network controller is less around 10cm from the target position, and the average path length is comparable to that generated by the path planning algorithm.

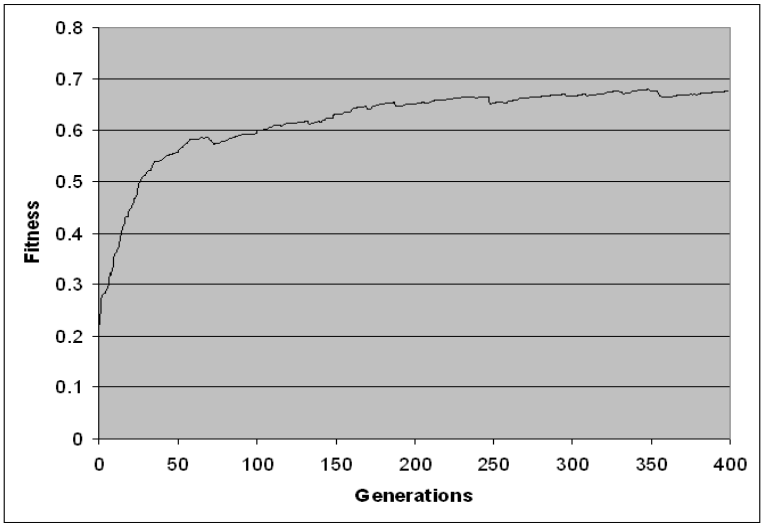

Figure 3.9: *Fitness of the best network found at each generation in an environment with obstacles averaged over five runs.* The best network that was found had a fitness of 0.7. This neurocontroller could position the end-effector to around 10cm of a target while avoiding obstacles. This demonstrates that neurocontrollers can be used to avoid obstacles while still controlling the arm effectively.

The neurocontrollers were trained to effectively control the arm while also avoiding obstacles. The best evolved neurocontroller was on average able to position the end-effector to within 10.43cm of a target position. The evolved neurocontrollers on average takes a shorter path around obstacles towards a target position compared to the path planning algorithm. The average final distance of the target from the end effector for the neurocontroller is close to the controller that uses path planning. This experiment demonstrates that it is possible to evolve neurocontrollers that can effectively control the arm in environments with obstacles. The best evolved neurocontroller takes a short path to the target and positions the arm to within 10cm of the target.

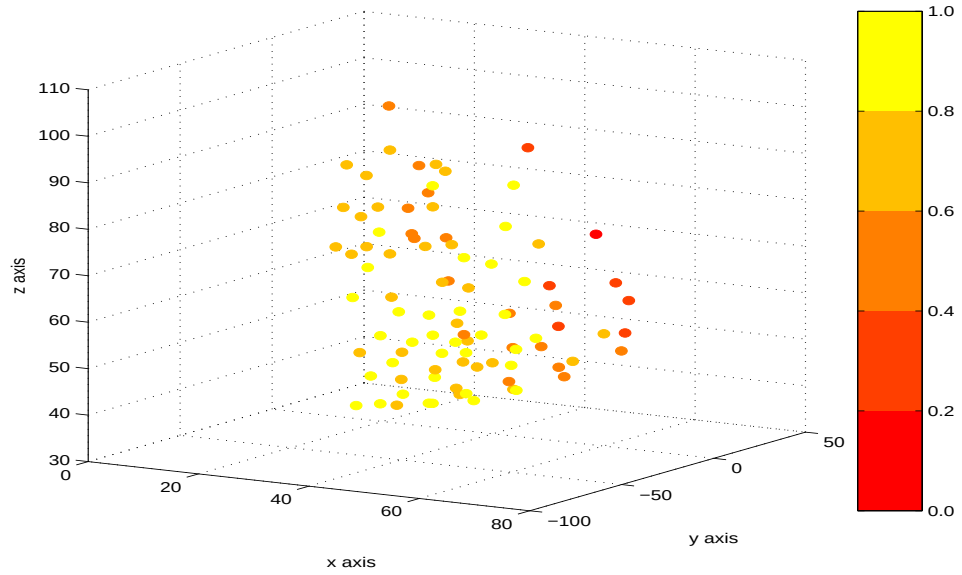

Figure 3.10: *Fitness display of best network that was evolved for each point in the training set in an environment with obstacles.* The graph shows the performance of the best evolved neurocontroller on every point in the training set. The points are color coded according to the fitness with yellow being the highest fitness and red the lowest fitness. The fitness of the neurocontroller for most positions in the training set is at least 0.6. The neurocontroller avoids obstacles 59 out of the 96 target positions in the training set.

	$AveragePathLength$	$AverageTargetDist$
$NeuralNetwork$	48.49	10.43
$PathPlanning$	69.45	2.83

Table 3.3: *Comparison of neural network controller and path planning controller.* This table shows the average path length and average final distance of the end effector from the target for the best evolved neurocontroller and a controller that uses path planning. The neurocontroller takes a shorter path to the target, but the analytical controller is able to move closer to the target.

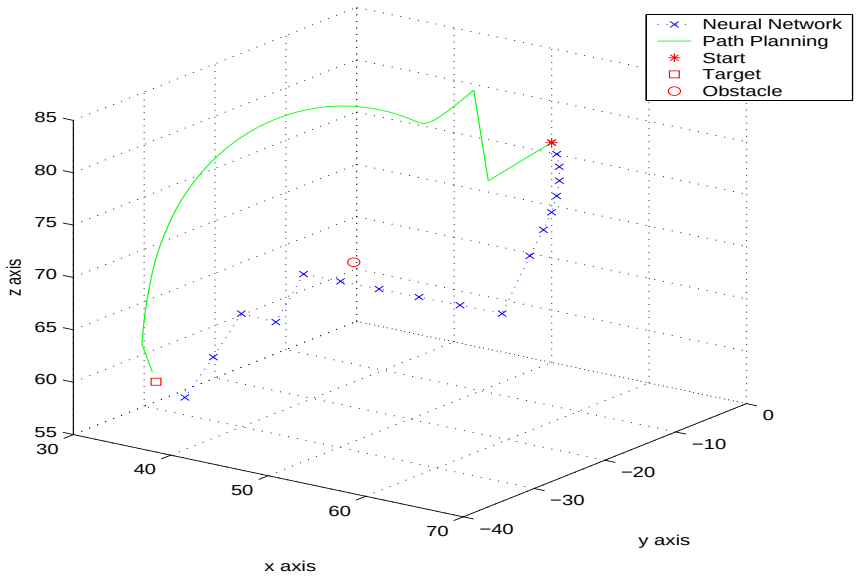

Figure 3.11: *Comparison of the path taken by the end-effector for a neural network controller and a controller that uses path planning.* The obstacle is placed at the midpoint of the line joining the starting position and the target position. The path planning controller first moves directly to the target, but hits a local minimum because of the obstacle. It then makes a random move away from the obstacle and then moves to the target. The neurocontrollers moves to takes a path that avoids the obstacle towards the target. The final distance of both controllers from the target is 2cm.

Chapter 4

Discussion

This report presented a method for evolving neurocontrollers that can be used to operate robot manipulators in environments with and without obstacles. The neurocontrollers use range sensors that are fitted along the robots joints to sense obstacles and the distance of the target relative to the end-effector.

The neurocontrollers that were evolved were compared with analytical controllers that used the solutions to the inverse kinematic equations and path planning to control the robot manipulator. The neurocontrollers evolved in for the environment without obstacles can move the end effector to within 4.43 cm of the target, which is almost as good as an analytical controller. This results demonstrates that neurocontrollers can be used to control manipulators in environments without obstacles.

The neurocontrollers evolved for environments without obstacles are also able to move the arm to 10cm of the target. The path taken by the evolved neurocontrollers is relatively short. These results demonstrates that the neurocontrollers can be used to perform complex tasks like obstacle avoidance. The performance of the evolved neurocontrollers comes close to the analytical controller with respect to final distance from the target.

The neurocontrollers evolved are able to cover most of the distance from the starting configuration to the target and on average are able to position close to within 5cm for environments without obstacles and 10cm for environments with obstacles. One possible explanation for this is that robot control consists of two types of move-

ment. First, the neurocontroller must make several large joint rotations to move close to the target. These movements can include detecting and avoiding obstacles in the robot's path. Second, the neurocontroller must make smaller, more precise movement to get within grasping range of the target. It is possible to evolve neurocontrollers that do both, however because of the diminishing returns of late evolution, it takes a larger number of generations to evolve controllers that can move the very close (less than 1cm) of the target. One way of overcoming this is to evolve a secondary neural networks that are activated only when the target distance is less than 10cm. These networks are evolved specifically to control the arm when the target distance is less than 10cm. Future work will use secondary neurocontrollers to control the arm after the primary neurocontroller positions the end-effector within 10cm of the target.

The advantage of using neurocontrollers is the ability to adapt to different environments and manipulator configurations. Neuroevolution can be used to retrain neurocontrollers that can function in the new environment. The neurocontrollers described in this report do not change once they have been evolved. It would be desirable to build a neurocontroller that could detect and adapt to changes in the robot arm configuration or environment. Nolfi and Parisi [7] developed a self-teaching architecture for training online neurocontrollers. In this architecture, the training input for the neurocontroller is provided by a separate neural network that has the same sensor input as the neurocontrollers and is subjected to the same evolutionary process as the neurocontroller, but with a fitness function that is designed to train the network to detect environmental changes. The teaching network can recognize the configuration of the environment and robot and modifies its outputs in order to train the neurocontroller to adapt to the new environment.

The self-teaching architecture could be used to create online learning controllers by evolving the ability to adapt the neurocontroller in response to changes in the robot or environment configuration. Future work will focus on training robust adaptable neurocontrollers in simulation and using these neurocontrollers to control

real manipulators.

Chapter 5

Conclusion

The goal of this was report was to evolve neurocontrollers that could control robot manipulators in environments that could position the robot manipulator's end effector close to target positions while avoiding obstacles.

First, neurocontrollers were evolved for environments without obstacles to demonstrate that the NEAT genetic algorithm could be used to evolve neural networks suited to the robot arm control task. The performance of the best neurocontroller that was evolved in this experiment was compared with an inverse kinematics controller and a potential field controller. The neurocontroller could move the end effector to within 5cm of a target position, which is close to the inverse kinematics and potential field controller results.

In the second experiment, neurocontrollers were used to evolve controllers that could function in environments with obstacles. The performance of the best neurocontroller that was evolved in this experiment was compared with a controller that used a path planning algorithms to navigate around obstacles towards a target. The best neurocontroller is able to position the end effector to within 10cm of a target. This result indicates neuroevolution was able to integrate obstacle avoidance behavior into the neurocontroller. The best neurocontroller evolved had a shorter average path length, and came close to matching the performance of the path planning controller.

The advantage of using neurocontrollers for robot arm control is that neurocontrollers can be adapted to different robot and environment configurations. Future research will focus of evolving robust controllers that learn online which evolve to

adapt to changes in the robot's environment.

Index

Abstract, iv

Background and Related Work, 3
Bibliography, 35

Conclusion, 32

Discussion, 29

Experiments, 14

Introduction, 1

Learning to control the arm in an environment with obstacles, 22
Learning to control the arm in an environment without obstacles, 15

Robot Arm Kinematics, 8

Bibliography

[1] Thomas Buehrmann and Ezequiel Di Paolo. Evolving obstacle avoidance behaviot in a robot arm. In *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems*, 2004.

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

[3] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 148–154, San Francisco, 1987. Kaufmann.

[4] A. C. Sanderson L. E. Weiss and P. C. Neumann. Dynamic sensor-based control of robots with visual feedback. *Journal of Robotics and Automation*, RA-3(5), 1987.

[5] S. Hutchinson M. Spong and M. Vidyasagar. *Robot Modelling and Control*. Wiley, 2006.

[6] David E. Moriarty and Risto Miikulainen. Evolving obstacle avoidance behaviot in a robot arm. In *From Animals to Animats: Proceedings of the Fourth International Convference on Simulation of Adaptive Behavior*, Cape Cod, MA, 1996.

[7] S. Nolfi and D. Parisi. Learning to adapt to changing environments in evolving neural networks, 1995.

[8] N. P. Papanikolopoulos and P. K. Khosla. Adaptive robotic visual tracking: theory and experiments. *IEEE Transactions on Automatic Control*, 38(3), 1993.

[9] K. O. Stanley. *Efficient Evolution of Neural Networks through Complexification.* PhD thesis, Department of Computer Science, University of Texas at Austin, 2004.

[10] K. O. Stanley and R. Miikkulainen. Evolving a roving eye for go. In *Proceedings of the Genetic and Evolutionary Computation Conference*, New York, NY, 2004. GECCO, Springer-Verlag.

[11] A. Stoytchev. Toward learning the binding affordances of objects: A behavior-grounded approach. In *Proceedings of AAAI Symposium on Developmental Robotics*, Stanford, CA, March 2005. AAAI.

[12] P. J. Werbos. *Neurocontrol and supervised learning: An overview and evaluation.* Van Nostrand, New York, 1992.

Vita

Thomas D'Silva was born in Cochin, India on January 12, 1984 . He attended high school in Dubai, U.A.E. He obtained his Bachelor of Science in Electrical and Computer Engineering from the Illinois Institute of Technology in Chicago. He joined the University of Texas in the Fall of 2004.

Permanent address: 1071 Clayton Ln., Apt 421
Austin, Texas 78723

This report was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.