

Neural Network Control of a Free-Flying Space Robot

Edward Wilson* Stephen M. Rock†
Stanford University
Aerospace Robotics Laboratory
Stanford, California 94305
ed,rock@sun-valley.stanford.edu

Abstract

Recent developments in neural network control at the Stanford Aerospace Robotics Laboratory are presented. A “Fully-Connected Architecture” (FCA) is developed for use with backpropagation (BP). This FCA has functionality beyond that of a layered network, and these capabilities are shown to be particularly beneficial for control tasks. A complexity control method is successfully used to manage the extra connections provided, and prevent over-fitting.

Second, a technique that extends BP learning to discontinuous functions is presented and applied to a difficult on-off thruster control problem. This method has many applications, namely any time a gradient-based optimization is used for systems with discontinuous functions. The modification to BP is very small, simply requiring replacement of discontinuities with continuous approximations and injection of noise on the forward sweep.

The viability of both of these neural network developments is demonstrated by applying them to a thruster mapping problem characteristic of space robots. Real-world applicability is shown via an experimental demonstration on a 2-D laboratory model of a free-flying space robot.

1 Introduction

In our research program in neural network control at the Stanford Aerospace Robotics Laboratory, we have developed some widely-applicable neural network methods during our efforts to apply networks to the control of a free-flying space robot.

In Section 2, the experimental equipment (robot) we have used in this research is described, and the particular thruster mapping problem we address is presented. Controlling the on-off thrusters presents a truly challenging problem for the neural network application. In Section 3, we present a “Fully-Connected Architecture” (FCA) for use with backpropagation (BP), that has greater functionality than a standard layered network. Particular benefits of the FCA, some of which are especially useful for control problems, are outlined. In Section 4, we present a method for using BP learning with systems containing discontinuous functions, such as the on-off thrusters on our robot. The method is a simple modification to standard BP, and extends to multiple layers of hard-limiting neurons or the FCA without modification. The experimental viability of these methods was verified on our robot, and is presented in Section 5.

2 Robot Control Application

The control task that motivated the neural network developments was the control of position and attitude of a free-flying space robot using on-off thrusters. Control using on-off thrusters is an important problem for real spacecraft, and the non-linear and adaptive capabilities of neural networks make them attractive for these problems.

Our experimental equipment consists of a mobile robot that operates in a horizontal plane, using air-cushion technology to simulate the drag-free and zero-g characteristics of space. This robot, shown in Figure 1, is a fully self-contained planar laboratory-prototype of a free-flying space robot complete with on-board gas, thrusters, electrical power, multi-processor computer system, camera, wireless Ethernet data/communications link, and two cooperating manipulators. It exhibits nearly frictionless motion as it floats above a granite surface plate on a 50 micron thick cushion of air [1].

*Ph.D. Candidate, Department of Mechanical Engineering. Research partially supported by NASA and AFOSR.

†Associate Professor, Department of Aeronautics and Astronautics.

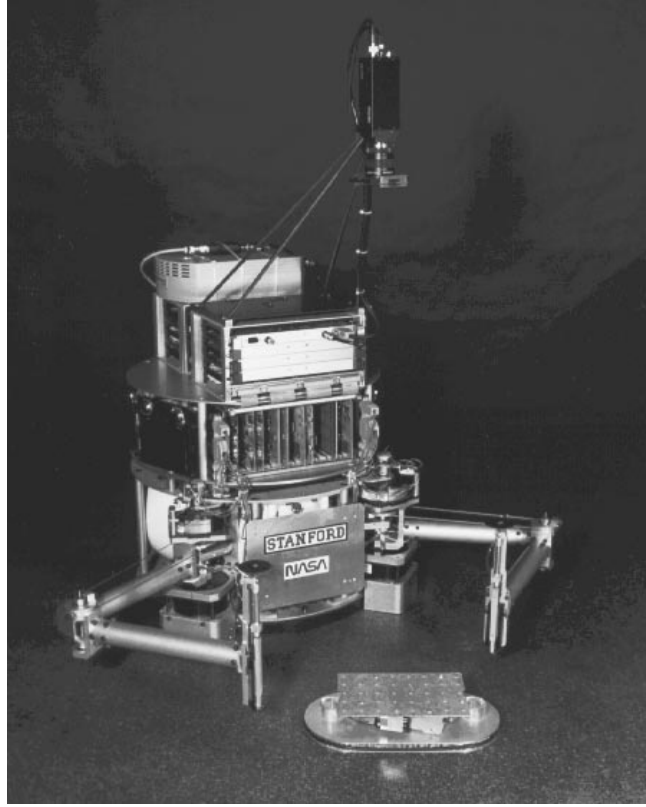


Figure 1: **Stanford Free-Flying Space Robot**

The three degrees of freedom (x, y, θ) of the base are controlled using eight thrusters positioned around its perimeter, as shown in Figure 2. The thruster mapping task to be performed during each sample period is to take an input vector of continuous-valued desired forces, $[F_{xdes}, F_{ydes}, \tau_{\theta des}]$, and find the output vector of discrete-valued (off, on) thruster values, $[T_1, T_2, \dots, T_8]$, that minimizes a specified cost function. The on-off nature of the thrusters substantially complicates the control design, due to their discontinuous nature and the fact that each thruster simultaneously produces both a net force and torque. The current base control strategy is shown in Figure 2.

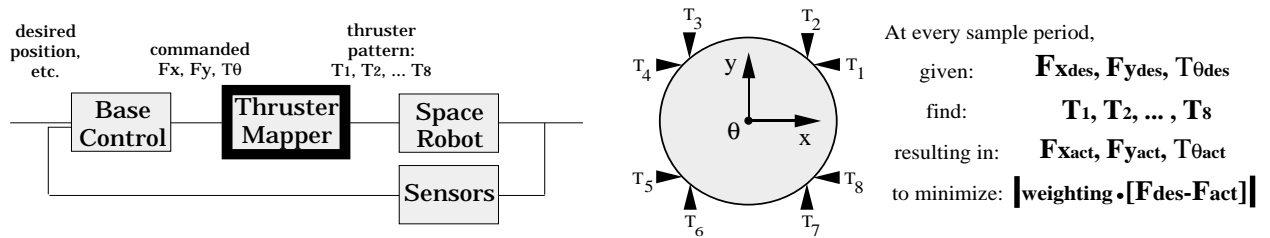


Figure 2: **Base Control Strategy, Thruster Mapping Problem Definition**

Left: the control strategy treats the thrusters as linear actuators. Right: the on-off thrusters and coupling between forces and torque make this problem difficult. The thruster mapper must find the thruster pattern producing a force closest to that requested by the base control module.

In this paper, we focus on developing neural networks for the “Thruster Mapper” component. A subsequent step, made possible by the developments in Section 4, is to merge the base controller and thruster mapper design into a single component, improving performance.

Three different techniques used to solve the thruster mapping problem have been applied as research progressed. They are summarized in Figure 3. The first implementation used an exhaustive search at each sample period to find the thruster pattern minimizing the force error vector [1]. Symmetries are used to

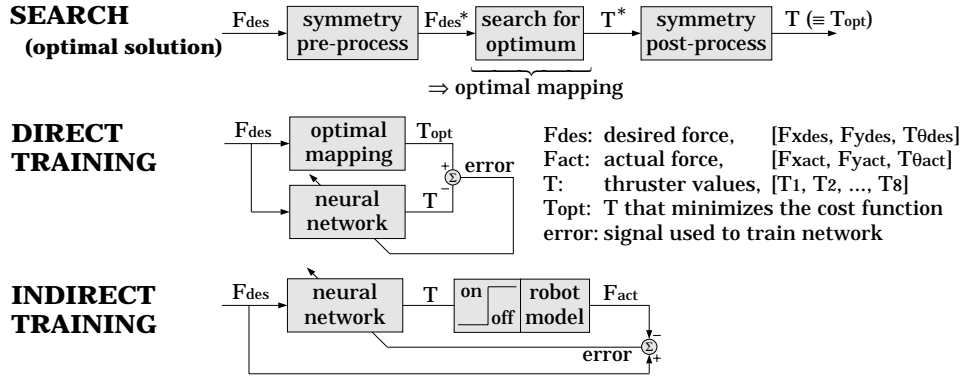


Figure 3: Thruster Mapping Methods

reduce the search space, but this method relies on testing every possible thruster pattern to find the one with minimum error. In the second method, a neural network was trained to emulate the optimal mapping produced by the exhaustive search [2]. This work led to the Fully Connected Architecture, presented in Section 3. The third method used a neural network trained to find the optimal solution when presented with a model of the plant, but no optimal teacher. This required back-propagation of error through the discontinuous thrusters, which motivated development of the noise injection method presented in Section 4.

3 Fully-Connected Architecture

3.1 Introduction

We develop a general architecture for feed-forward neural networks that can be trained using backpropagation [3] [4]. This “Fully-Connected Architecture” refers to the structure shown in Figure 4, which was first presented by Werbos [4] [5]. The network’s neurons are considered to be ordered, beginning with the first input, ending with the last output, and having hidden units in between, perhaps interspersed among input or output units. Note that there is no longer a concept of layers. Backpropagation restricts information flow to one direction only, so to get maximum interconnections, each neuron takes inputs from all lower-numbered neurons and sends outputs to all higher-numbered neurons.

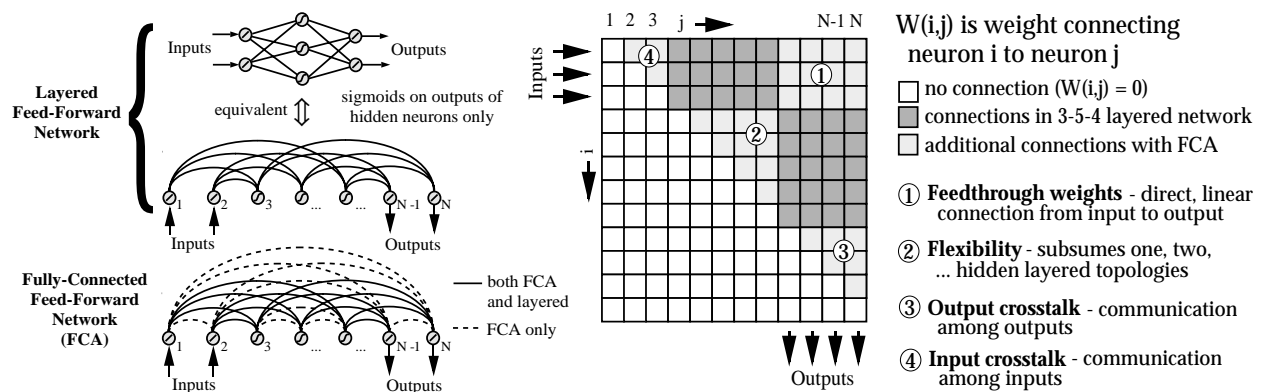


Figure 4: Extra connections available with FCA

This general feed-forward architecture subsumes more-familiar single or double-hidden-layer architectures. Here we show that it has all the connections of a single-hidden-layer network, and some extras as well.

3.2 Comparison with layered network

The right side of Figure 4 highlights the benefits of the extra connections that are unused in a single-layered network. The question is whether the enhanced functionality outweighs the increased computational load and susceptibility to over-fitting. This must be decided for each application.

Advantages of the FCA:

- The Feed-through segment is a matrix that implements a direct, linear connection from inputs to outputs (provided sigmoids are used only on hidden units). This provides fast initial learning and allows *direct pre-programming* of a linear solution calculated by some other method. This is particularly important for control applications, where there is a large body of linear control knowledge that can be drawn upon to provide a good starting point. The FCA provides for seamless integration of linear and non-linear components
- Flexibility: since the FCA subsumes any number of hidden layers, when combined with a systematic weight pruning procedure, the network topology (defined by the remaining connections) is set in a systematic manner based on gradient descent.
- Cross-talk among inputs and outputs may be valuable, i.e. one output may excite or inhibit another output, a feature unavailable with layered networks.

Disadvantages:

- Increased complexity: number of weights increases quadratically with the number of hidden units, versus linearly for a layered architecture. The extra weights increase susceptibility to over-fitting.
- Slower hardware implementation: updating must be one neuron at a time, versus one layer at a time for layered networks.

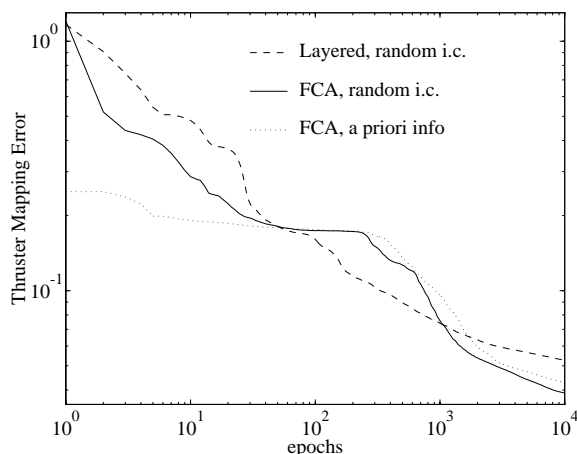


Figure 5: Training History, FCA versus Layered Networks

Figure 5 compares learning histories (thruster mapping error on the training set) for the thruster mapping problem outlined in the previous section. The networks, each with 5 hidden neurons, were trained to emulate the optimal mapping (minimizing force error). The figure represents the average performance for ten different sets of initial weights.

Looking at the initial learning performance, the FCA network performs better, due to the weight gradient being instantly available via the direct connection of inputs to outputs. As expected, the FCA network with the *a priori* linear solution built in provides the best early performance.

In the middle region, between 100 and 1000 epochs, the layered performance surpasses that of the FCA, due to the reduced number of parameters, and simplified search space. However, after 1000 epochs, the greater functionality of the FCA network comes into play and performance surpasses that of the layered network.

3.3 Complexity control

The example of Section 3.2 has shown the potential value of the extra connections associated with a fully-connected neural network. However, the high number of parameters, while increasing functionality, makes the network susceptible to over-fitting. Often during training, performance on test and training sets will improve until a certain point, and then test performance will worsen as the network stops generalizing, and begins to fit the particular data set, as seen in Figure 6. Use of a “sufficiently-large” training set can reduce over-fitting problems, but this may not be practical due to a lack of data, or an adaptation speed requirement that requires a faster solution than this brute-force approach.

Many systematic network pruning techniques have been proposed. One we have used successfully involves the addition of a complexity cost term to the total cost function, as first proposed by Weigend and Rumelhart in [6]. Each weight contributes $\lambda \cdot (w_s^2 / (w_s^2 + 1))$ to the total cost function, where $w_s = w/w_0$ is a scaled value of the weight. The scale factor, w_0 , effectively sets the cutoff point for weights, and λ selects the relative importance of complexity cost versus performance cost.

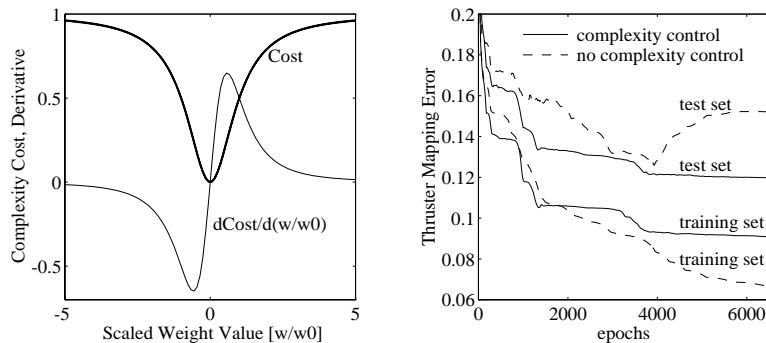


Figure 6: **Complexity Control Function, Effect on Network Performance**

The complexity control function and training histories for a fully-connected network with 5 hidden neurons are plotted in Figure 6. Without complexity control, over-fitting becomes clear at around the 4000th epoch, as the performance on the test set worsens, while performance on the training set improves. With the addition of the complexity term, over-fitting is controlled, as performance histories on test and training sets no longer diverge.

4 Backpropagation for Discontinuous Functions

4.1 Introduction

Optimization methods that use gradient information often converge much faster than those that do not. Use of the backpropagation algorithm to get this gradient information for training neural networks has made them useful in many applications; however, BP’s requirement of continuous differentiability, not only for the network itself, but for anything that the error is backpropagated through (e.g. the plant model in a control problem), limits its applicability.

This is a significant limitation since there are many applications where discrete-valued states arise. For example: on-off thrusters commonly used in spacecraft; other systems with discrete-valued inputs and outputs; and NNs built with signums (also known as hard-limiters or Heaviside step functions) rather than sigmoids. Signum networks may be preferred to sigmoidal ones due to hardware considerations.

In cases like these, one choice is to use an alternative method not restricted to continuously differentiable functions, such as unsupervised learning, simulated annealing, or a genetic algorithm, but these are usually significantly slower to train, because they do not use gradient information.

Another option is to approximate the discrete-valued functions with linear functions or smooth sigmoids during the learning phase, and switch to the true discontinuous functions at run-time, as with the original ADALINE [7]. This method works in many cases where the behavior of the system with sigmoids is close

enough to that of the real system. However, this assumption is unreliable, and the thruster control problem presented here offers a clear example where this method fails.

In related research aimed at using gradient-based learning for multi-layer signum networks, Bartlett and Downs [8] use weights that are random variables, and develop a training algorithm based on the fact that the resulting probability distribution is continuously differentiable. The algorithm is limited to one hidden layer, requires all inputs to be 1 or -1, and needs extra computation to estimate the gradient.

In this section we present a technique for BP learning for systems with discontinuous functions, and apply it to the on-off thruster control problem described in Section 2.

4.2 Noisy sigmoid training algorithm

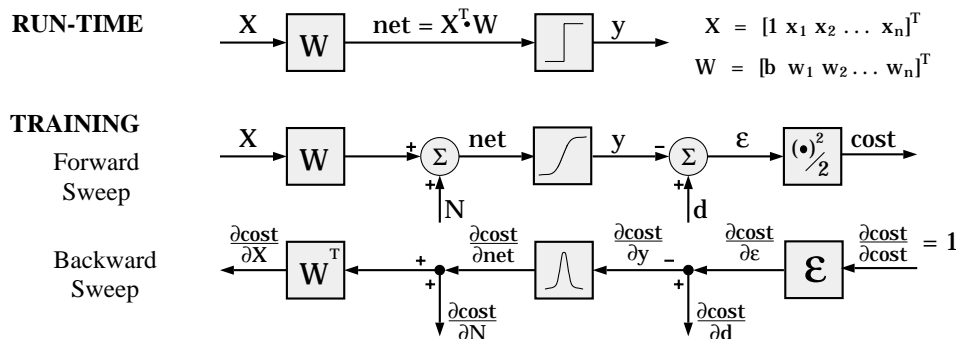


Figure 7: **Training Algorithm**

During training, replace discontinuous signums with sigmoids, and inject noise before the sigmoid on the forward sweep. The backward sweep calculation is the same as standard backpropagation.

We introduce the method of noise injection by applying it to the training of a single hard-limiting neuron, as shown in Figure 7. The first block diagram shows the neuron as it appears at run-time: a dot-product and hard-limiter. The next two diagrams show the neuron during training, where the hard-limit from the top diagram has been replaced by a smooth sigmoid function.

This is almost the same as training a standard neuron with backpropagation – the only difference involves the injection of zero-mean noise, N , immediately before the sigmoid. The noise injection enters just as the desired signal does, and does not corrupt the calculation of $\partial \text{cost} / \partial W$. Using an unmodified backward sweep is not only the simplest thing to do, it does precisely the right calculations for estimating the weight gradient. To summarize, the training algorithm is:

- Replace the hard-limiters with sigmoids during training
- Inject noise immediately before the sigmoids on the forward sweep
- Use the exact same backward sweep as with standard backpropagation

4.3 Intuitive Explanation

Without addition of noise, the network may train using values in the sigmoid transition region (roughly -0.8 to 0.8) that will be unavailable at run-time. Simply rounding off at run-time may introduce significant errors. For example, in a hypothetical cost surface, a value of 0.4 may be optimal, but if forced to choose between -1 and 1, a value of -1 may be better. The goal of noise injection is to move neuron activations away from the transition region, so round-off error will be small when the discontinuous functions are replaced.

An intuitive reason for adding the noise is to throw the neuron off its transition region, and effectively force it to hard-limit at the high or low value. For this reason, the standard deviation of the noise is chosen to be higher than the width of the transition region of the sigmoid. Figure 8 shows how the neuron output distribution changes as the noise level increases. With no noise, only a single output can result, but as noise increases to cover most of the transition region, the output distribution approaches that of a hard-limiting function. Differentiability is maintained, however, so gradient information will be available to speed up

learning. Since the noise has pushed the distribution to approximate a hard-limiting non-linearity, when the hard-limiter is re-introduced at run-time the performance degradation will be small.

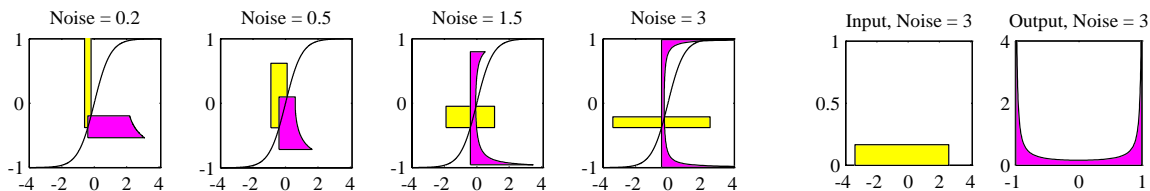


Figure 8: **Effect of Input Noise Level on Sigmoid Output Distribution**

Lightly-shaded region represents the sigmoid input probability distribution (in this case, $-0.3 + \text{noise}$). Darkly-shaded region is the sigmoid output distribution (from -1 to 1), plotted horizontally to correspond to the sigmoid plot. Each distribution has an area of 1. As noise level increases, and the input distribution spreads out, the sigmoid output approaches that of a hard-limiter, while remaining differentiable. At right, input and output distributions are plotted separately.

4.4 Extensions, application considerations

This method has been successfully applied to multiple layers of hard-limiting units with no modification. One concern is the attenuating effect of the derivative-of-sigmoid function. When back-propagated through many layers of near-saturated sigmoids, the error signal is attenuated and may lead to slow learning. To handle this problem, it may be necessary to be gradual in increasing the noise variance - slowly push the outputs from the linear region to the hard-limits, rather than all at once, where the attenuation is high and the network will find it difficult to react.

When using a system with discrete-valued functions that are not Heaviside step functions, the method may work if a continuously differentiable approximating function is used. For example, a function whose output can take on a number of discrete values may be approximated by combining a number of sharp sigmoid functions. This was done for the thruster mapping, replacing the eight $(0,1)$ thrusters with the equivalent set of four $(-1,0,1)$ thrusters.

4.5 Application to space robot

Here, we apply this technique to the thruster mapping, as shown in the third section of Figure 3, where the network finds a solution itself, without the help of an optimal teacher.

Training without this noise-injection technique is not possible. For example if one unit of thrust is requested in the $+x$ direction, during training, the network will set T_4 and T_5 to 0.5, but at run time, for requested forces near 1.0, chances are they will both be 0 or both 1, resulting in a large error.

A good solution results when noise is added because it prevents the network from using a solution that uses non-saturated portions of the sigmoid. Such a solution would give a nearly random output and high error during training. The training algorithm must find a solution that works well *despite* the noise addition. This means the expected value of the output must be well into the saturated region to consistently work well. The results approximate the optimal solution very well, and work when the sigmoids are replaced with signums.

5 Experimental Results

Experiments were performed on the mobile robot described in Section 1 to verify the applicability of these neural network results. The neural network thruster mapping component described in Section 3 is implemented on the on-board Motorola[®] 68040 processor, as is the rest of the control system (at a 60 Hz. sample rate).

Figure 9 shows robot base position during single-axis and multi-axis maneuvers. For the single-axis maneuver (left), good tracking is obtained from both optimal and neural network thruster mapping components. In the 3-axis maneuver (right), the neural network control system closely follows the 20-second-long straight-line trajectory in (x, y, θ) . Tracking error is very small, so to avoid clutter, only the robot's actual (x, y) position is plotted.

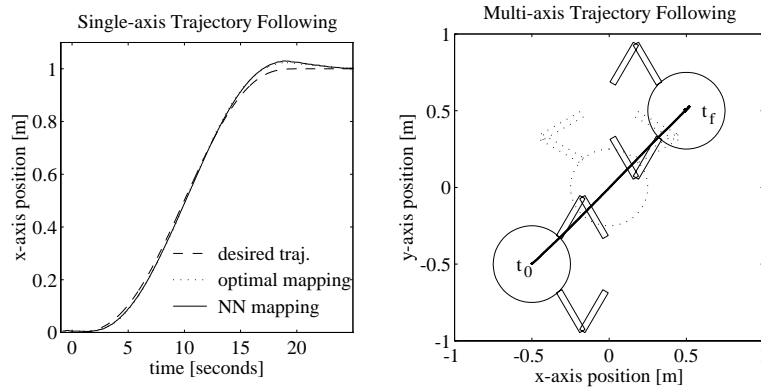


Figure 9: **Experimental Results, Single-Axis Maneuver, Multi-Axis Maneuver**

6 Summary and Conclusions

This paper has described two recent developments in neural network control that grew out of a research program using a laboratory-based prototype of a free-flying space robot. Both advances were motivated by, and developed for, a complex thruster mapping function typical of real spacecraft.

A fully-connected neural network architecture was presented that has connections beyond those provided by a layered network, yet is trainable with backpropagation. Aided by a systematic complexity control scheme, this network was shown to have certain advantages over layered networks, particularly for control problems.

A new technique was developed that extends BP learning to systems involving discontinuities, such as the on-off thrusters used to control our robot. The modification to BP is very small, simply requiring careful injection of noise on the forward sweep.

When tested experimentally on the real robot, all networks provided near-optimal performance during multi-axis trajectories, thus demonstrating the utility of these techniques.

References

- [1] M. A. Ullman. *Experiments in Autonomous Navigation and Control of Multi-Manipulator, Free-Flying Space Robots*. PhD thesis, Stanford University, Stanford, CA 94305, March 1993.
- [2] E. Wilson and S. M. Rock. Experiments in control of a free-flying space robot using fully-connected neural networks. In *Proceedings of the World Congress on Neural Networks*, Portland OR, July 1993.
- [3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In David E. Rumelhart, James L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing*, page 318. The MIT Press, Cambridge, MA 02142, 1986.
- [4] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, MA 02142, August 1974.
- [5] P. J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, October 1990.
- [6] A. S. Weigend, B. A. Huberman, and D. E. Rumelhart. Predicting the future: A connectionist approach. *International Journal of Neural Systems*, 1(3):193–209, 1990.
- [7] B. Widrow and R. Winter. Neural nets for adaptive filtering and adaptive pattern recognition. *IEEE Computer*, 21(3):25–39, March 1988.
- [8] P.L. Bartlett and T. Downs. Using random weights to train multilayer networks of hard-limiting units. *IEEE Transactions on Neural Networks*, 3(2):202–210, March 1992.