Real-time Implementation of Neural Network Learning Control of a Flexible Space Manipulator

R. Todd Newton Yangsheng Xu

CMU-RI-TR-92-11

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

August 1992

This work is supported by the Space Projects Office, Shimizu Corporation, Japan.

Contents

List of Figures

List of Tables

Abstract

This paper discusses a neural network approach to on-line control learning and real-time implementation for a flexible space robot manipulator. We at first overview the system development of the Self-Mobile Space Manipulator (SM^2) and discuss the motivations of our research. Then, we propose a neural network to learn control by updating feedforward dynamics based on feedback control input. We address in great detail the implementation issues associated with on-line training strategies and present a simple stochastic training scheme. A new recurrent neural network architecture is proposed, and the performance is greatly improved in comparison to the standard neural network. By using the proposed learning scheme, manipulator trajectory error is reduced by 85%. At last, we discussed the implementation of the proposed scheme in teleoperated control. The approach possesses a high degree of generality and adaptability in various applications and will be a valuable method in learning control for robots working in unconstructed environments.

1 Introduction

Robotic technology is useful in space exploration, for eliminating the need of astronauts working in inhospitable environments, and for promoting the productivity that astronauts might achieve. In design of a space manipulator, energy efficiency and micro-gravity effects must be considered. For this reason, the robot manipulator is normally designed as a light-weight structure which presents a significant flexibility between links and results in a challenge in control. The flexibility of a multi-degrees of freedom manipulator providing three dimensional motion is difficult to model, and thus most model-based control schemes are infeasible. On the other hand, because of the light-weight structure and micro-gravity condition, the joint torque is relatively much lower than in industrial manipulators, thus joint friction is relatively much higher. For the space robot manipulator developed in our laboratory, the torque for friction compensation is more than 30% of the total torque applied, compared to about 3% for normal industrial manipulators.

The structural flexibility and joint frictions cause an extreme difficulty in modeling the manipulator dynamics and providing a stable, efficient motion. To eliminate the high order vibrations of the structure, we apply a low-pass filter to the measured joint values. The system bandwidth, however, is decreased by the low-pass filters, and hence the speed of the manipulator motion is greatly limited. At the same time, relatively low feedback gains must be selected for each joint so to prevent low-mode vibration and maintain stable manipulation. Although the effects of friction are decreased by the use of a moving-sum integral error feedback term, the gains associated with these terms are limited to low values to avoid creating instabilities resulting from underdampened control. Working within these limitations, control of SM^2 without a dynamic model yields large trajectory errors.

We have been working on several adaptive control schemes, but the result depends on the structure of the model and computation is expensive. The approach of neural networks to dynamic modeling is not restricted by a model and is computationally efficient. This paper focuses on a neural network approach to real-time learning control and implementation for a flexible space manipulator. First, we overview the Self-Mobile Space Manipulator (SM^2) project testbed. We then introduce the concept of feedback-error-learning as a method of training a neural network. Details associated with the implementation and training are discussed. By using the proposed neural network scheme, the manipulator trajectory error is reduced by 85%. Finally, we purpose an interactive method of operator guided learning of local manipulator dynamics to improve the execution of teleoperated tasks. This direction promises to be particularly helpful during teleoperated tasks involving payloads which may substantially affect the dynamic properties of the manipulator.

2 The Self-Mobile Space Manipulator Testbed

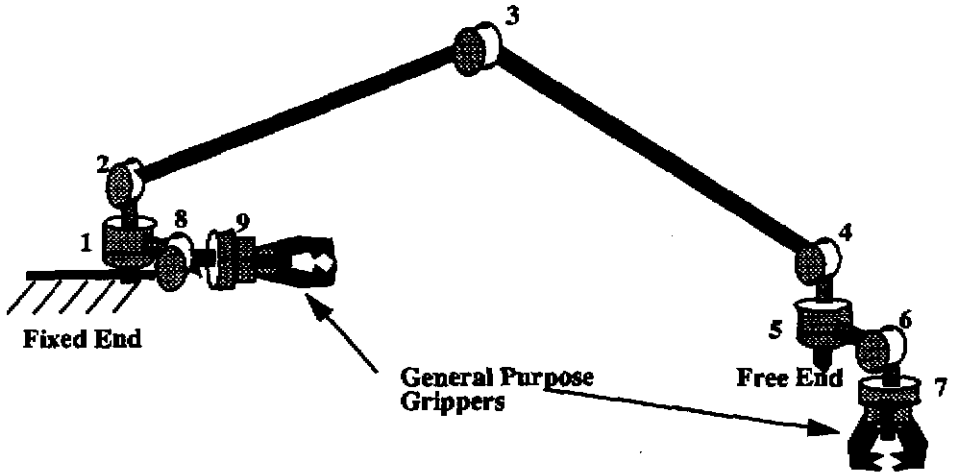

Figure 1: Concept picture of the symmetric robot, SM^2. While one end is attached to the trusswork fixing joints 8 and 9, the remaining joints 1-7 provide a 7-DOF manipulator configuration.

2.1 Basic Concept

In the Robotics Institute at Carnegie Mellon, we have developed a light-weight and low-cost robot [1,2] that provides independent mobility on the Space Station exterior and at the same time is capable of accomplishing manipulation tasks. We call this robot the Self-Mobile Space Manipulator, or SM^2. This robot would primarily be used for visual inspection, material transport, and light assembly. To achieve the mobility of the robot on the Space Station trusswork, the robot walker is of minimum size and complexity. As shown in Figure 1, SM^2 is a symmetric robot having nine joints, two slender links, and two end-effector at both ends. One type of end-effector is used to attach to trusswork providing mobility and the other serves as a general purpose gripper during manipulation tasks. Walking is accomplished by alternate grasping and releasing at the trusswork attachment locations, and swinging of the feet from one truss attachment location to the next. Using such steps with alternate feet, SM^2 can move across the exterior of the trusswork. At any given time one truss gripper is attached to a truss beam, fixing the two degrees of freedom associated with the general purpose end-effector on the same end. This provides a 7-DOF serial configuration for manipulation. Thus, the general purpose end-effector defines the tip of the robot, while the truss gripper at the other end acts as the robot's base.

2.2 Design and Hardware

In order to perform realistic experiments in the laboratory, we designed and built a 1/3-size laboratory robot based on a hypothetical, full-size, self-contained robot to be used on the Space Station (see Figure 2). We used scaling rules to keep the dynamics parameters (masses, stiffness, natural frequencies, linear speeds) of the scaled-down robot similar to those of the hypothetical one. Overall dimensions of the truss and robot were reduced to 1/3,

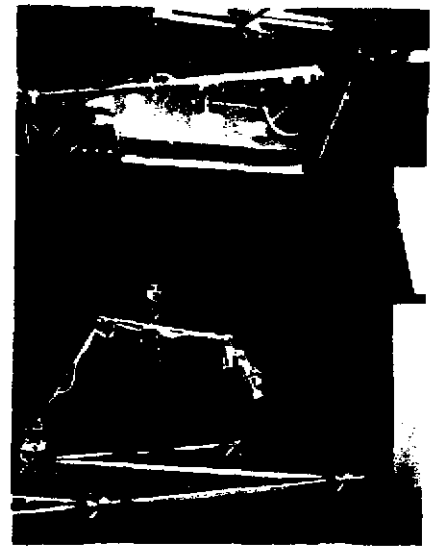

Figure 2: Photograph of SM^2 supported by gravity compensation system GCII. The axis of rotation of the GCII is aligned with the fixed end of the robot.

while local dimensions (joints and grippers) were kept equal. This allows the testbed to be used in an average size laboratory, while mechanisms are not unworkably small. Each joint contains a rare-earth-magnet DC motor; harmonic-drive speed reducer (60:1 or 100:1 ratio); and optical encoder on the motor shaft to measure joint angle. The motors and drive components were selected and arranged to give maximum power and torque in a small, lightweight package. Presently, the robot is tethered to the computer hardware. The software has been developed on the Chimera II [3] experimental real-time system. We also have installed a BIT3 bus adapter which controls communication on the VME bus between the real-time processors (2 Ironics CPUs) and host Sun.

2.3 Gravity Compensation System

The absence of gravitational forces in orbit has a dramatic effect on the design and operation of robots. To permit realistic testing on Earth we have developed two gravity compensation systems (GCI and GCII) that balance the significant gravitational effects on the robot. Each system includes a vertical counterweight system and an actively controlled horizontal system. The vertical system comprises a counterweight mechanism, and a series of pulleys and cables that provide a constant upward force to balance the robot while only increasing the inertia of the system by 10%. Horizontal motions are servocontrolled to keep the support point above the robot. An optical sensor mounted on the carriage of the GC measures the deviation from the vertical of the support cable connecting the carriage to the robot. A servocontroller tries to null this deviation by driving motors causing the above gantry to move. The first system (GCI), having a cartesian XY based axis gantry, is used for mobility experiments. The second system (GCII) shown in Figure 2, which operates in cylindrical coordinates, has a boom pivoted above the fixed end of the robot, and a carriage that moves along the boom to match the radial movements of the robot. Although the GCII system does not permit mobility, it is more suited for delicate manipulation experiments since the gantry axis coincide with the principle axis of movement of the robot in the horizontal plane. All experiments in this paper are conducted using the GCII system.

3 Learning Control with Neural Networks

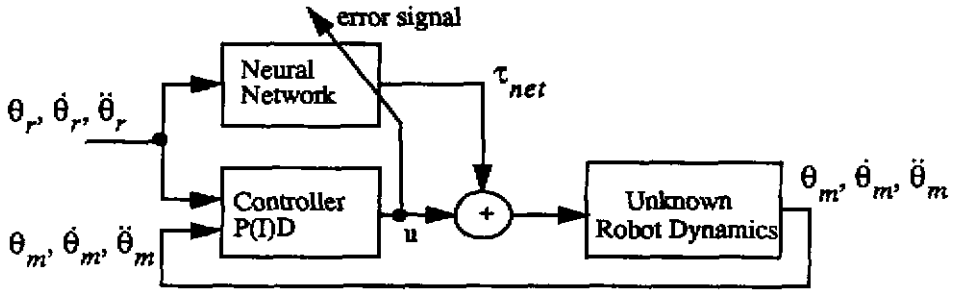

Figure 3: Block diagram of the general concept of learning control through error feedback learning. θ_m and θ_r describe the reference output of the plant and the measured output of the plant respectively. τ_{net} and u describe the output signal of the neural network and the feedback signal of the controller respectively. Note that the feedback from the controller is used to teach the neural network.

3.1 Feedback-Error-Learning Concept

To learn the dynamic model of SM^2 we employed the method of neural network feedback-error-learning [4]. Inspired from the proficiency of learned control demonstrated in biological systems, the neural network developes an inverse dynamics model through experience. More specifically, this neural network scheme uses the error obtained from the linear controller (i.e. PID or PD) to adapt the weights of the neural network (Figure 3). Using a feedforward neural network architecture trained using backpropagation [5], the network learns to become what is commonly referred to as the feedforward term of the system. The uniqueness of learning through feedback lies in the ability to learn mappings where the target values are not known, only a signal estimating the error in the present output is available. Therefore, the network learns by continually estimating the target from the feedback error signal received from the controller. Equation1 shows the target estimation, $\hat{\tau}^t_{target}$, as a function of u and τ^t_{net} representing the feedback error and the actual output of the network at time t.

$$\hat{\tau}^t_{target} = \tau^t_{net} + u \tag{1}$$

The process of modeling the manipulator dynamics is then accomplished on-line, during system execution. After each system cycle the weights of the network are updated so as to minimize the error feedback. In time the neural network learns the correct network output to produce the desired/reference output of the plant, and in turn the feedback signal decreases. Teaching neural networks by error feedback has been shown to yield improved performance is several simulation studies [4,6,7]. However, there been little work regarding the real-time implementation, and the advantages of the approach as well as various implementation issues cannot be revealed from simulation. This is one of the motivations of our research. Second, as we addressed previously, SM^2 is a flexible space manipulator and the dynamics are difficult to model, yet cannot be neglected. Therefore, we cannot use the conventional computed-torque method where an exact model is needed for complete cancellation of the nonlinear dynamics of the system. Having no feedforward compensation, however, is also

not acceptable since the feedback term can only generate approximately 50% of the total torque necessary for acceptable tracking. In addition, since SM^2 is limited to low feedback gains but possesses significant frictions, large static errors are observed. Presented in this paper, the approach of neural networks trained by feedback-error-learning is shown to greatly reduce both tracking errors and static errors.

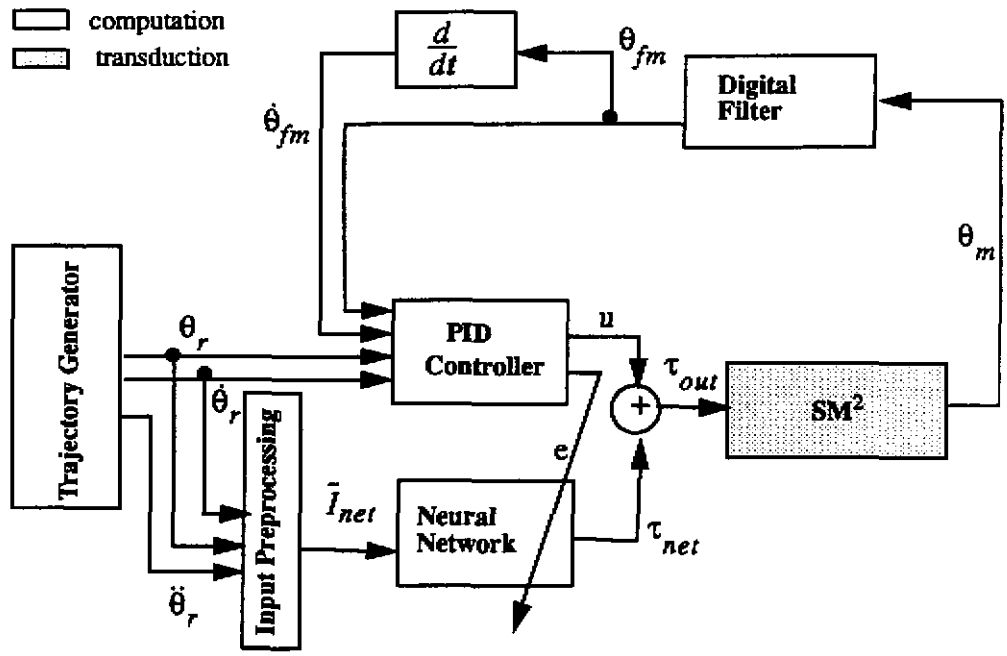

Figure 4: Block diagram of control flow for SM^2 using a neural network. θ_m, θ_{fm}, and θ_r represent the measure joint angle position, the filtered measure joint angle position, and the reference joint angle position respectively. u and e represent the PID feedback signal used for motor control and the PD feedback signal used to correct the network. τ_{net}, and τ_{out} represent the output torque value of the network and the composite torque to be sent to the joint. $\bar{I}_{net}$ represents the input vector to the neural network produced by the preprocessing step (i.e. normalizing and subcomponent composition).

3.2 Neural Network Control Architecture

The first three joints from the base of the SM^2 position the end-effector and payload in gross motion. Thus, these joints are the most difficult to control because they are restricted to low controller gains to maintain stability, avoiding excitation of structural frequencies. For this reason, our experiments will be restricted to learning control to these first three joints from the base. Since the configuration of the remaining dorsal joints will remains constant throughout the experiments, none of the networks are given inputs dependent on the dorsal joints. As a result, this forces the networks to view the dorsal joint system as a lumped mass. Each network has as inputs the particular reference velocity and reference acceleration associated with that joint and several additional terms which are dependent on the other two remaining joints. All inputs are normalized such that each maximum input value is less than one. The scaling constants are determined by the robot before learning starts. By normalizing all inputs we provide each input with an equal chance to be utilized while discouraging

potential saturation problems of neurons in the proceeding layers. Each network has one output representing the feedforward torque term. Internally each network has two hidden layers of 13 and 20 neurons. The neurons of the hidden layers possess sigmoid transfer functions with activation range [-1,1] while the output neuron has a linear transfer function. As is usually the case, the sigmoid transfer function was selected to aid in modeling the nonlinearities of the system. On the other hand, since it is unclear what the desired range of the output activations should be, the output neurons are given a linear transfer function resulting in an unbound activation range. The torque applied to each joint is the sum of the network activation torque and the controller feedback torque. A diagram of the system as applied to SM^2 is shown in Figure 4.

Since the dynamics of each joint are depended on more than simply the reference velocity and reference acceleration of that joint, we provide each network with several additional input terms that describe some of the stronger dependencies on the other joints. The terms chosen are subcomponents of the dynamics equations for the first three joints when using a lumped-mass, rigid-body model.

We could use an exact model for the flexible manipulator, but this would be more complex. These preprocessed input terms provide dependencies between joints. They also allow the network to implicitly assume some known nonlinear relations among joints. As a result, the speed and accuracy at which the network may learn the correct dynamics increases. This method has been proven successful in previous studies [4]. Table 1 shows the inputs for each of the three joints. These terms are actually a subset of those originally proposed by the earlier study [4]. Each term represents a subcomponent used in the derivation of the inertial, centripetal, Coriolis, or frictional components of the joints.

Table 1

Inputs to the three neural networks for joints 0,1, and 2 respectively.

Joint 0	Joint 1	Joint 2
$\ddot{\theta}_{J0}$	$\ddot{\theta}_{J1}$	$\ddot{\theta}_{J2}$
$\dot{\theta}_{J0}$	$\dot{\theta}_{J1}$	$\dot{\theta}_{J2}$
$\mathrm{sgn}\,(\dot{\theta}_{J0})$	$\mathrm{sgn}\,(\dot{\theta}_{J1})$	$\mathrm{sgn}\,(\dot{\theta}_{J2})$
$(\sin\theta_{J1})^2\ddot{\theta}_{J0}$	$\sin\theta_{J2}\dot{\theta}^2_{J2}$	$\cos\theta_{J2}\ddot{\theta}_{J1}$
$\sin(\theta_{J1}+\theta_{J2})\,(\dot{\theta}_{J1}+\dot{\theta}_{J2})^2$	$\sin(\theta_{J1}+\theta_{J2})\cos(\theta_{J1})\,\dot{\theta}^2_{J0}$	$\sin\theta_{J1}\cos(\theta_{J1}+\theta_{J2})\,\dot{\theta}^2_{J0}$

3.3 Trajectory Sequence Training Scheme

A common problem found in training neural networks is ensuring that the training patterns are evenly distributed across the problem domain. In the case of training three joints of a manipulator the domain is trajectories. Note that the trajectory domain should not be con-

fused with the workspace domain of the manipulator. A trajectory sequence could train evenly across the workspace, but still be very narrow in the types of trajectories it was trained with.

There are several challenges that are specific to training neural networks on-line. One problem with repeatedly training over a fixed sequence of distributed trajectories is overlearning the sequence which has been shown to lead to poor generalization [8]. Another problem is that the method of teaching a task by presenting local portions of the task one at a time is not suitable for on-line training of neural networks [8]. The lack of diversity for a sustained period of time results in the degeneration of information previously encoded within the network. In essence, the network "forgets". Both of these factors discourage the use of deterministic schemes.

Training through on-line feedback also places challenges on how the networks should update their weights. Most off-line training in large, complex domains is done by updating the network using the accumulated gradient across a set of training patterns. This method enhances learning by increasing the network's ability to reach the global solution. Unfortunately, for on-line training of manipulator dynamics this is difficult since defining what patterns should be in a training set is not straightforward. Furthermore, since the networks are training on error signals and not known targets the actual error signal is itself only an estimation. The estimations of the target values change over time, therefore reexperiencing old patterns used in a training set to gain new estimations is essential for convergence. If learning over a training set is applied here, difficulties arise in determining how to sequence the execution of the trajectories to include the input patterns of the training set and how to determine what members of the set should be replaced or reexperienced.

Considering these issues the option of training by a simple scheme of stochastically generated trajectories appears to avoid many of the potential problems pointed out earlier. At the same time, a stochastic approach ensures a more continuously distributed sequence of training trajectories. For this reason training over a trajectory set, or epoch, is less essential. Instead, updating the network continuously based on only the previous error gradient is sufficient.

The most obvious approach to a pseudo-random trajectory generating scheme is to repeatedly produce a random point in the 3 degrees-of-freedom workspace and execute the trajectory from the present configuration to the produced endpoint configuration. The workspace of the manipulator is defined not by physical joint limitations but by polar coordinates. Because there are stability limitations due to the degree of extension of the manipulator tip and physical limitations due to the placement of the trusswork, specifying trajectory endpoints in polar coordinates as shown in Figure 5 is the easiest way to define the legal workspace for training. The polar endpoint is then transformed into joint space for trajectory calculations. There is still a problem with training using sequences of trajectories defined by connecting random points in the workspace. Although the configuration endpoints are evenly distributed, the actual trajectory defined by the difference between the two bounded random endpoints is not evenly distributed (when integrated over all possible endpoint possibilities). This is illustrated in Figure 6. As a partial solution to this problem, a critical difference value (in radians) has been defined for each joint. In order for a random endpoint to

be valid, at least one joint must have a movement greater than the critical value for that joint. The test is repeated until a valid new endpoint is found. In short, this solution does not given an even distributed, but it does provide a simple solution which does not bias toward smaller trajectories.

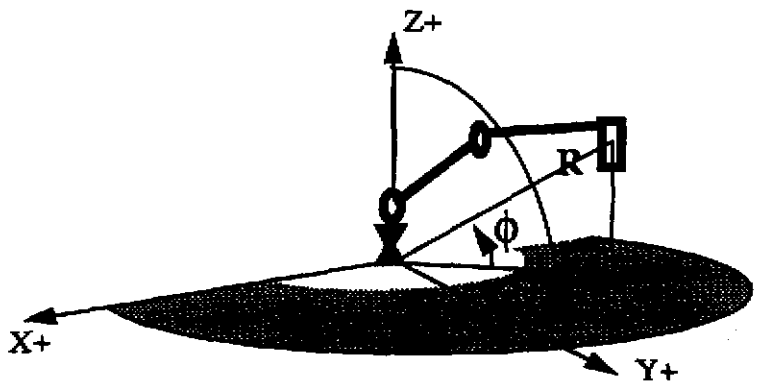

Figure 5: The representation of polar coordinates defined by θ, ϕ, and R. Each parameter has a selected minimum and maximum. For SM^2, the workspace defined by the limits forms a quartersphere with a hollow center.

The trajectory is interpolated in joint space using the trigonometric functions shown in Equation 2-4 for each joint.

$$p = nt - \sin(2\pi * nt) * (1/2\pi) \qquad \text{(EQ 2)}$$

where nt is normalized time between [0,1] during trajectory execution. Note all joints use the same nt. The current value of nt is defined as shown:

$$nt = i / (\omega\alpha) \qquad \text{(EQ 3)}$$

where ω is the sampling frequency, α is the specified period of the trajectory, and i is the index counter which is incremented after each iteration of execution.The trajectory is then defined as a function of nt :

$$\theta = \theta_0 + \Delta\theta * p \qquad \text{(EQ 4)}$$

where, for a particular field of joint space, θ is the present reference position, θ_0 is the initial position, and $\Delta\theta$ represents the difference between the initial and end positions.

An upperbound velocity magnitude is selected for each joint. The period of the actual trajectory is defined by the period of whichever joint is found to take the most time at the upperbound velocity. In this way each trajectory will contain at least one joint which is executing at maximum velocity. Although we have limited the variety of trajectories by selecting a specific trajectory scheme of interpolation and period selection, the portion of the trajectory domain has been selected to most closely match the required types of trajectories

used by most applications of the manipulator.

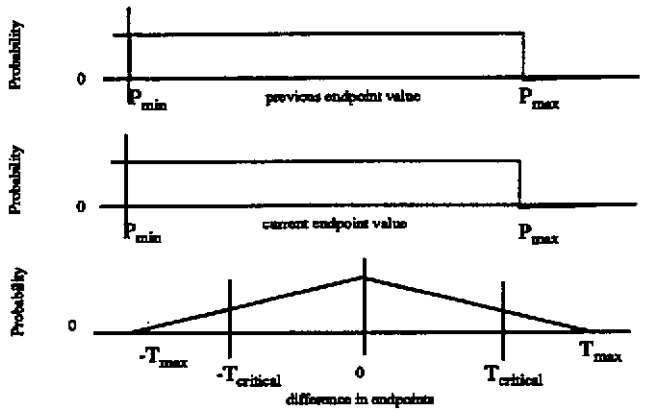

Figure 6: Example of relative probability distribution of two discrete parameter endpoints and probability distribution of the difference of the two parameter endpoints.

In summary, we have developed an algorithm for effectively training the first three joints of SM^2 over the desired workspace. This algorithm may be expanded to work with different manipulators. Below is an outline of the trajectory generation process:

Algorithm

1. Select a random point in polar coordinates and transform the point into joint space coordinates.
2. Check to see if the magnitude of the difference between the present position and candidate endpoint is greater than the critical amount for each of the joints. If no difference is above the threshold, repeat step 1 else proceed to step 3.
3. Select the time period of the trajectory by using the maximum of the periods computed from each individual joint executing the joint trajectory at maximum velocity.
4. Generate actual trajectory and execute.
5. Return to step 1.

3.4 Initial Issues

Before discussing particular experiments, several of the initial issues encountered regarding learning convergence are discussed. One of the first problems experienced during learning was the saturation of neurons in the hidden layers. This resulted from backpropagating from the linear neuron with an unbound error signal to the hidden layer neurons possessing bounded activation levels and expecting bounded error signals. To avoid saturations we have lowered the learning rate for all the connections other than the output neuron connections. This allows the output connections to establish the large weight values needed to generate the desired range of output activations before the hidden layer connections saturate in

an attempt to create activations which are out of their range.

Since we are training on-line, the initial state of the network must maintain the manipulator's stability. Therefore, it is difficult to initialize the network with random weights having large ranges (i.e. $|x| > 1$) since this results in random outputs of larger magnitudes creating instabilities in control. On the other hand, initializing the network with random weights having a small range (i.e. $|x| < 0.05$) slows learning since the weight changes are proportional to the weight value. To overcome the delay in learning associated with the use of small initial weight values, the network's learning rate is amplified by a factor of five for the first 20 seconds. This "shock" allows the weights to more quickly overcome the period where little learning is accomplished due to the small magnitude of the connection weights.

In preliminary experiments using the error signal generated by the PID controller was found to cause large trajectory endpoint overshooting during learning. Since the integral term of the controller causes large increases in error feedback for near-static errors, the network was led to believe that more torque should be applied during the final portion of the trajectories. Over the learning phase, the network slowly converges to the correct action by alternating between undershooting and overshooting the target over the training trajectories. In view of these observations, manipulators with low bandwidth operation and high frictional effects should use PD feedback signals, not PID feedback signals, as the network teaching signal. For our experiments we use a PID signal to control the manipulator and a PD signal to teach the network. Although PID is used to control the manipulator, the trade-off between having less tracking and less static error and having underdampened performance limits us to considerably low integral gains.

4 Implementation

For each of the following experiments the manipulator is autonomously trained in the manner previously described. The learning period is selected as the minimal learning time required until observed improvement in performance ceases (observed from MSE). This time interval was found to be 600 seconds. After the learning phase the network adaptation is turned off, and a test trajectory sequence of 60 seconds is executed. The results of the test sequence are shown for each experiment. The control frequency of the manipulator is 40 Hz. Because of computational limitations, the neural networks are limited to a frequency of 25 Hz. Controller gains were empirically determined to provide the best conventional performance possible without creating instabilities.

4.1 Experiment I

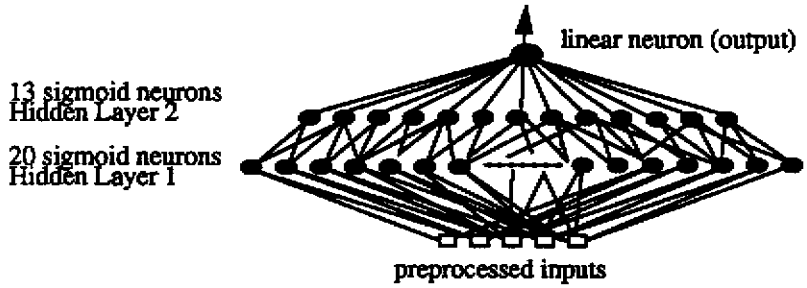

Figure 7: The neural network architecture used for each of the three joint in Experiment I. Each layer is fully connected.

The first experiment uses the standard network architecture shown in Figure7 for each of the three joints. As learning commences the mean controller feedback signal decreases as the neural network plays an increasing feedforward role. By the end of the training phase the mean feedback signal for each joint had typically decreased by 65%. A comparison of the test sequence performance between PID control and PID with learned neural control is shown in Figure 9. There is an obvious improvement in tracking using the neural networks, but there is a problem in the latter portions of several trajectory segments in the test sequence of joint 0. The errors are due to over-deacceleration outputted by the network causing a reverse in motion and resulting in static errors. Because the ratio of inertial mass to friction is large for joint 0 relative to the other joints, joint 0 relies more on applying reverse torques to deaccelerate while joints 1 and 2 rely more on frictions to deaccelerate. As would be expected from this observation, the other two joints do not exhibit this problem. Rather, both tracking error and static error decreased for joints 1 and 2. To overcome some of the limitations of this network's architecture, the next experiment focuses on providing a network architecture with temporal context in improve the modeling capacity.

4.2 Experiment II

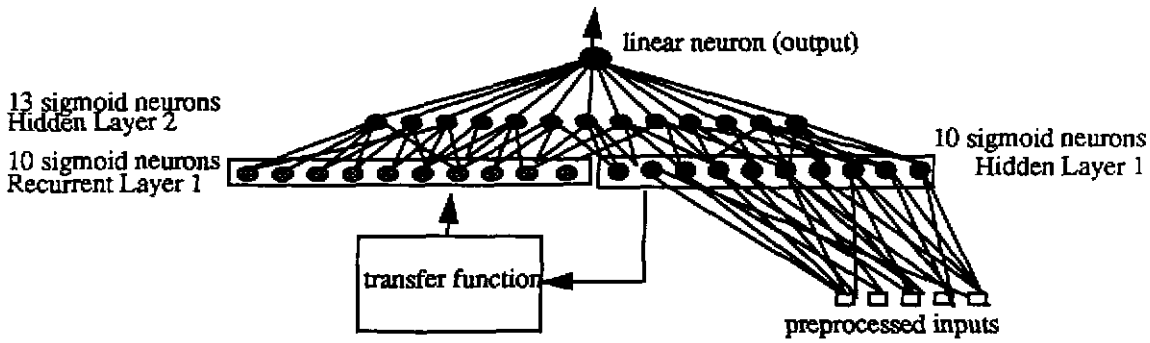

Figure 8: The new neural network architecture used for each of the three joint in Experiment II. Note the first hidden layer is composed of two halves. The activations of the right half (black neurons) are determined as before, but the activations of the left half (gray neurons) are determined by individually applying a transfer function to neurons of the right half. Note that each level is fully connected.

The major limitation of the network used in Experiment I was that its output torque was only dependent on the inputs applied at that instant. In other words, the network did not possess the ability to evaluate the new inputs using any context obtained from previous inputs. Many investigations into recurrent networks have proven successful in providing context which improves the network's capacity to learn [5,6,9]. Some of the more popular approaches feed back activation levels of output neurons or hidden neurons as input for the next feedforward cycle. For our network application, recurrency of the hidden layer activation vector is investigated since it has more information capacity in comparison to the output scalar. Network instability problems associated with directly feeding back the previous hidden layer 1 state as inputs were avoided by feeding them instead into hidden layer 2 as shown in Figure 8. This network architecture was applied to SM^2 using the standard transfer function z^{-1} (1 iteration delay) . Unfortunately the network had the same performance as that of Experiment I. The input values between the previous and present iterations differ so little that the feature roles between the corresponding neurons of two halves were identical.

Two modifications are made to aid the network in utilizing temporal knowledge. First, use the recurrent vector to present the difference between the current and previous states of hidden layer 1. Second, increase the temporal interval being applied. Combining these modifications amounts to taking the difference of the moving average of the activations over a fixed time period. In summarizing the modification, we use the same architecture as in Figure 8, but the transfer function has been modified to that shown in Equation 5.

$$\frac{1 - z^{-b}}{b} \tag{5}$$

Note b represents the interval size in iterations of the window used in compute the moving average. The moving average interval was selected to be 0.5 seconds, or 12 iterations at 25

Hz. After applying this architecture, the performance improved over that of Experiment I by 50%. A comparison can be made between Experiment I and Experiment II using Figure 9. Most notably, the problems observed in joint 0 have been overcome. At the same time several noticeable static errors existed in Joint 1 and Joint 2 using the architecture of Experiment I, but are now overcome using the recurrent architecture of Experiment II.

4.3 Experimental Results

Shown in Table 2 is a comparison of the mean square error (MSE) over the entire test trajectory sequence using various control architectures. Note that learning was turned off for the test trajectory sequence execution.

Table 2
Comparison of the mean square error (radians^2)

Control Type	Joint 0	Joint 1	Joint 2
PD	0.1342	0.0064	0.0089
PID	0.0350	0.0012	0.0030
PID/NN Exp I	0.0101	0.0004	0.0010
PID/NN Exp II	0.0046	0.0001	0.0005

In Table 2 one may note that the addition of integral control decreased the controller error by typically 74%. Even so, errors are still visible from the trajectory plots using PID. The introduction of neural networks into the control scheme was found to reduce the MSE by typically 85% in comparison to PID, and 96% in comparison to PD.

Although the MSE is an effective method of measuring performance, there is ambiguity in what types of errors the MSE value is reflecting because the MSE is a mixture of both steady state errors (errors still present when the reference is constant) and tracking errors. When observing the joint trajectory plots for PID in Figure 9, we see the primary problem with joint 0 is tracking error, and in joints 1 and 2 the problems are more results of steady state error(s). The network architecture of Experiment I decreased the MSE by primarily improving the tracking errors, but steady state errors still remained noticeable. The modeling required to overcome the remaining errors was first hypothesized to be the most context dependent. This hypothesis was verified by using a recurrent network to provide context information to successfully overcome the (near-)steady state errors. Looking at the plots (Figure 9) from Experiment II, we see the differential moving-average recurrent network architecture was able to improve the control such that both types of errors decreased substantially.

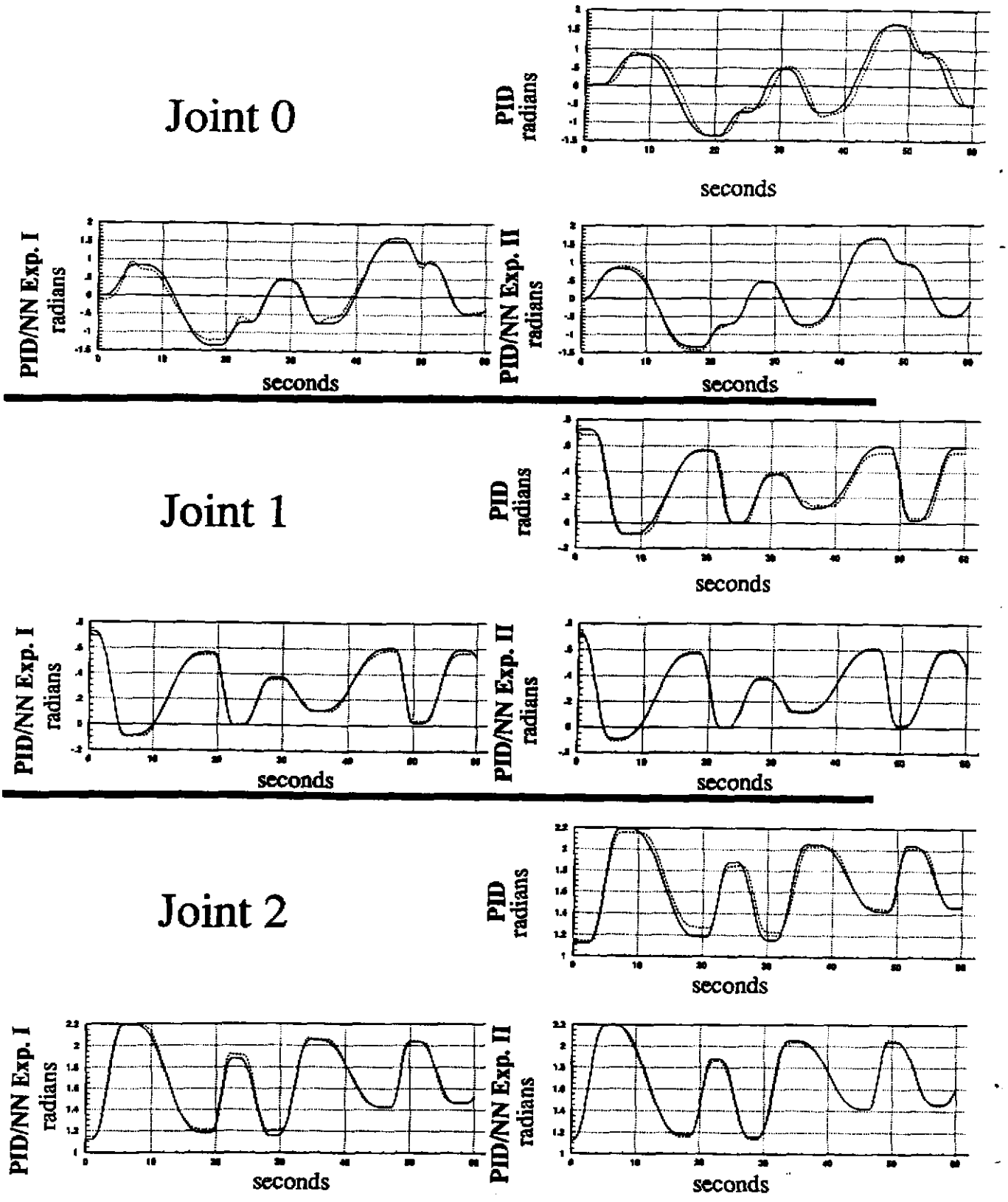

Figure 9: The above plots show the performance of the three joints for the test trajectory sequence over various control architectures. Reference trajectories are shown in solid and measured are in dashed.

5 Applying Neural Networks with Man-In-The-Loop

5.1 Teleoperated Learning

While autonomous control is reasonable for locomotion, teleoperation is an efficient and feasible approach to the control of manipulation in space applications. We have developed a teleoperation interface with the *Bird*, a commercial, 6-DOF, free-flying hand controller that provides position and orientation of a radio transmitter (stationary) relative to a receiver (moving). The moving part of the Bird is mounted on a movable structure to improve the stability of the human arm during operation as shown in Figure 10.

The study of dynamic modeling of (flexible) manipulators rarely considers human interaction with the learning/adaptation process. Here we propose an interactive method which may be used by a teleoperator to learn local portions of the trajectory domain which are relevant to a particular task. The system is setup similar to that used in the experiments shown in Section 5, but the reference position is now given by the teleoperator. The Cartesian frame received from the teleoperation device is used as the reference input in Cartesian space for the manipulator. Via the inverse kinematics of the manipulator the reference joint values are obtained.

Although the low-level control loop remains the same, the operator may now control which trajectories should be learned. Using a pressure sensitive transducer mounted on the 6DOF device, the pressure applied by the operator is proportional to the learning rate applied to the networks. In this way the operator may direct which actions should be used in modeling the dynamics. Another button may be used to reset the network weights after a particular task is completed. The operator is also provided with audio feedback to aid in guiding the learning process. By selecting a particular joint, the operator hears a signal whose frequency is

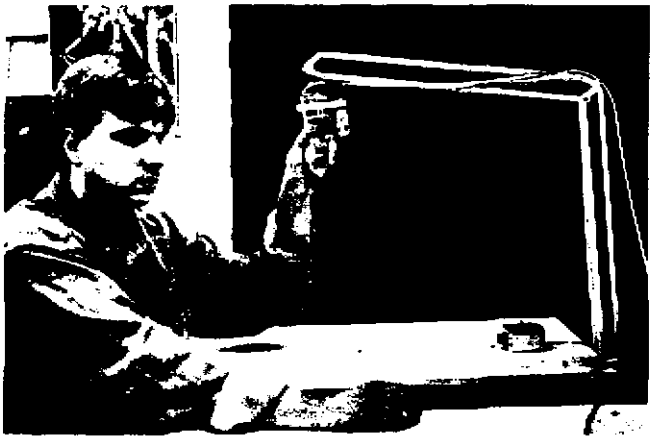

Figure 10: Used for teleoperation experiments with SM^2, the Bird receiver mounted on a movable frame is shown above. The Bird handle position in Cartesian space corresponds to the manipulator end-effector in the cartesian workspace. For our experiments, the teleoperator may look directly at the manipulator.

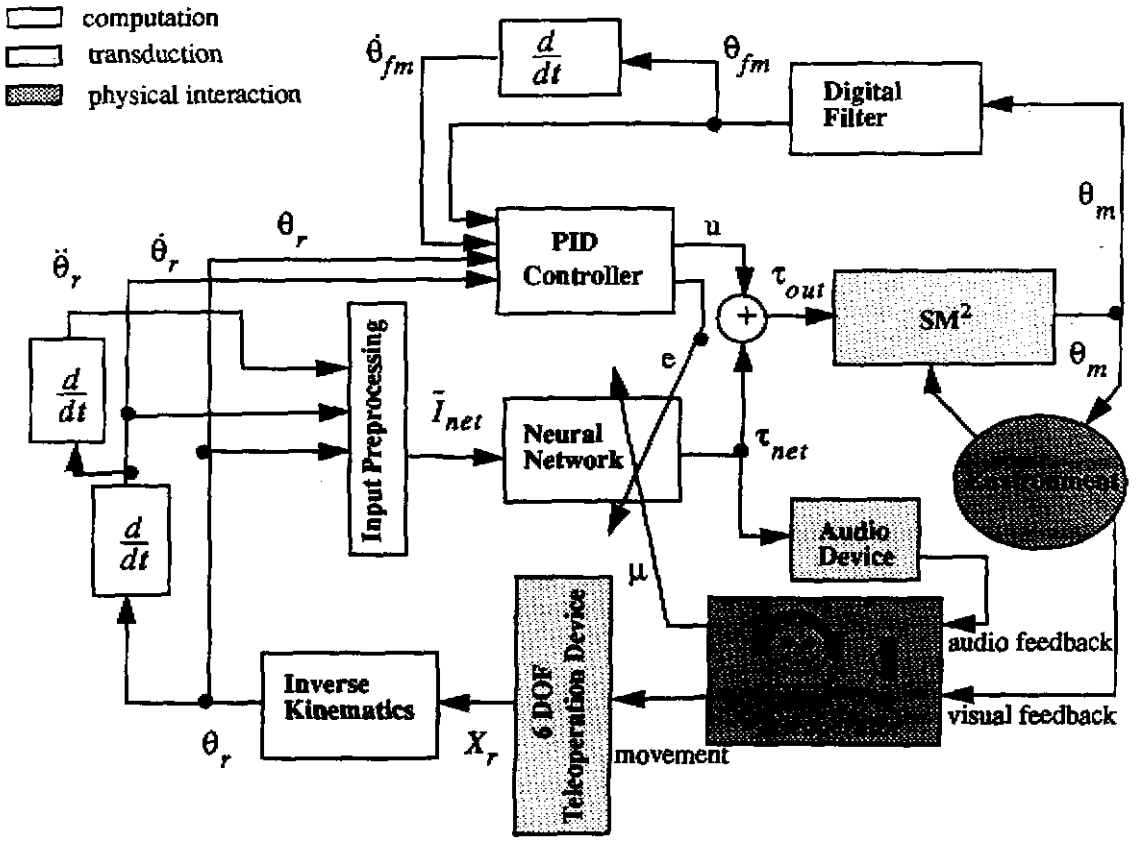

Figure 11: Block diagram of control flow for SM² using teleoperation. X_r represents the Cartesian reference frame as received from the 6DOF device. μ represents the learning rate which is now controlled by the teleoperator. The other symbols are defined as previously shown in Figure 4.

proportional to the output of the neural network associated with the particular joint. The audio feedback seems to be a very helpful que to the teleoperator in conveying which trajectories the networks are actually learning to play a role in. This information aids the operator in ensuring that the networks have learned the dynamics for all the types of movements associated with the particular task at hand. With the combination of visual interpretation of the manipulator's actions and audio interpretation of the network's internal state, the operator may better understand what the network is learning. The block diagram of the entire system discussed is shown in Figure 11.

Because the space teleoperation scenario is often envisioned with the control station being farely remote, effort has often been devoted to the study of teleoperation with transmission delays from several seconds to several minutes [10,11]. At present our project is concerned with close-range teleoperation therefore transmission delays are not considered. Although small delays exist in our control loop (50 ms), we consider them marginal and are more interested in the effects of limited manipulator control on teleoperation. Unfortunately, little literature has been found to address the effects on teleoperation by limiting control via low feedback gains and low-pass filtering [12]. Before investigating the specific application of

neural networks to teleoperation, a more general investigation of the effects of control on the teleoperator's ability to direct the manipulator must precede. For now, we present several observations regarding teleoperator behavior and use these observations to support the application of neural networks for the modeling/adaptation on process. Finally, a simple experiment is presented which supports our hypothesis that improved dynamics via neural networks enhances teleoperation performance.

5.2 Summary of Preliminary Observations

Teleoperators performing with transmission delays commonly adopt an effective "move-and -wait" strategy [10]. More specifically, the operator adopts a strategy of sequentially moving open loop and then waiting a delay time for correct feedback. This "move-and-wait" strategy was also observed by teleoperators using SM^2, but the reasons for adopting it are different. Because SM^2 is restricted by low feedback gains and low-pass filtering, lags in tracking are noticeably present. At the end of a discrete teleoperated movement, a period of typically 0.25 to 1.0 seconds is required for the manipulator to settle to some static position. Although the pause the teleoperator adopts is quite small, for complex tasks the additional time resulting from the accumulation of pause intervals can become significant. By incorporating the neural networks to learning the local dynamics, the tracking lag diminishes and the operator is provided with a more accurate response therefore decreasing the pause period.

In our preliminary experiment we are concerned with only the improvement of the manipulator to execute the desired movements of the teleoperator. For this reason, we will avoid introducing other factors which would affect how the network would adapt. More specifically, we will avoid tasks involving carrying payloads and exerting forces on external objects. Our selected task is to simply execute a sequence of motions along a beam placed in front of the manipulator. The beam has four markings approximately 5X5 cm in size spaced 25 cm apart. The sequence of movements defining the task is shown in Figure 12. Five subjects were repeatedly timed at completing this task. Each subject was required to touch a marked segment after each trajectory before proceeding to the next trajectory. Each subject first executed the task four times with PID control and four times with PID/Neural control. This was empirically determined to be a sufficient amount of time for the subject to learn the task. Next, the subject executed the task again, four times with PID/Neural control and four times with PID control. The average task completion times for each subject using each type of control is shown in Table 3. Note that each time the subject used the PID/Neural control, a brief teleoperated training period was executed by the subject.

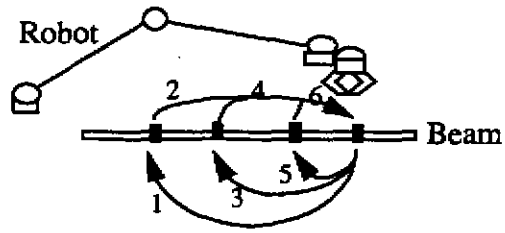

Figure 12: Sequence of movements for teleoperation task.

Table 3

Comparison of Average Task Completion Times (in seconds)

	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5
PID	31.2	27.5	32.2	34.6	26.1
PID/NN	25.0	24.2	28.2	30.0	21.6
improvement	19.9%	12.0%	12.4%	13.3%	17.2%

A consistent increase in task execution speed is observed across the subjects. Despite the inevitable presence of other factors not discussed, a decrease in tracking error and pausing periods are observed. It remains to be seen how much improvement can potentially be gained by teleoperated learning. Furthermore, to what extent is the network learning the manual characteristics of the teleoperator opposed to the dynamics of the manipulator?

6 Conclusions

We have presented a real-time learning control scheme using neural networks and successfully implemented the scheme for a light-weight space manipulator. The approach significantly improved the motion precision of the manipulator by learning the flexible robot dynamics on-line. The trajectory error is reduced by 85% compared to the results obtained from conventional linear control. The approach also possesses a high degree of generality and adaptability. It may be applied to different manipulators or even different domains. The approach of neural networks also provides rapid adaptability implicitly. This is particularly crucial where changes in payloads or in actual manipulator configuration may cause substantial dynamics model updating. Particular to a light-weight space robot, the approach overcomes the difficulty in real-time dynamics modeling in a micro-gravity environment.

We have discussed issues associated with on-line training strategies and presented a simple stochastic training scheme. We have presented a new recurrent neural network architecture which improves performance in comparison to the standard neural network. The proposed moving-average differential recurrent network succeeded in utilizing context to improve performance over the original network architecture by 50% where commonly used recurrent network approaches failed. The method of teleoperator guided learning of the manipulator dynamics was proposed to allow temporary local modeling to aid in particular teleoperated tasks.

Portions of future research directions may be conducted independent of the robotics domain. First, an effective method should be developed to find an optimal recurrency transfer function for the modeling of any unknown plant using the feedforward recurrent network design. Second, an obvious future direction will be to investigate the neural networks' adaptive capabilities with tasks, both teleoperated and autonomous, involving payloads. Third, future studies should further evaluated the potential of teleoperated guiding of dynamics modeling using neural networks.Interaction between the human, the manipulator, and the neural network must also be addressed. Particularly intriguing may be the identification of psychophysical issues associated with teleoperator guided training of neural networks in various real-time domains.

7 Acknowledgments

This work is supported by the Space Projects Office, Shimizu Corporation, Japan. We would like to thank the following colleagues for their technical contributions and support: Ben Brown, Mark Friedman, Randy Casciola, Greg White, David Stewart, Jie Yang, Shigeru Aoki, and Takeo Kanade.

References

[1] Y. Xu, B. Brown, S. Aoki, and T. Kanade, Mobility and Manipulation of a Light-Weight Space Robot, *IEEE International Conference on Intelligent Robots and Systems*, 1992.

[2] H. Ueno, Y. Xu, B. Brown, M. Ueno, T. Kanade, On Control and Planning of a Flexible Space Manipulator, *IEEE International Conference on System Engineering*, 1990.

[3] D. Stewart, R. Volpe, and P. K. Kosla, Integration of Software Modules for Reconfigurable Sensor-Based Control Systems, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1992.

[4] H. Miyamoto, M. Kawato, T. Setoyama, and R. Suzuki, Feedback-Error-Learning Neural Network for Trajectory Control of a Robotic Manipulator, *Neural Networks*, Vol.11, pp. 251-265, 1988.

[5] J. McClelland, D. Rumelhart, and PDP research Group,*Parallel Distributed Processing*, Volume 1, Cambridge, MA: MIT Press, ISBN 0-262-18120-7, 1986.

[6] K. Wilhelmsen and N. Cotter, Neural Network Based Controllers for a Single-Degree-of-Freedom Robotic Arm, *International Joint Conference on Neural Networks*, 1990.

[7] K. S. Narendra and K. Parthasarathy, Identification and Control of Dynamical Systems Using Neural Networks, *IEEE Transactions On Neural Networks*, Vol 1 No. 1, 1992

[8] D. Pomerleau, *Neural Network Perception for Mobile Robot Guidance*, PhD thesis, Carnegie Mellon University, 1992.

[9] B. Pearlmutter, Learning State Space Trajectories in Recurrent Neural Networks, CMU Technical Report CMU-CS-88-191, 1988.

[10] W. Ferrell, Remote Manipulation with Transmission Delay, *IEEE Transactions on Human Forectors in Electronics*, 1965.

[11] J. Funda, *Teleprogramming: Towards Delay-Invarient Remote Manipulation*, PhD thesis, University of Pennsylvania, 1991.

[12] T. L. Brooks, Telerobotic Response Requirements, *IEEE International Conference on Systems, Man and Cybernetics Conference Proceedings*, 1990.