

Modern Adaptive Control With Neural Networks

Chad Cox, Dr. Karl Mathia, John Edwards, and Richard Akita

Accurate Automation Corporation
7001 Shallowford Road
Chattanooga, Tennessee 37421

Abstract

Artificial neural networks (ANNs) are increasingly recognized as powerful tools for complex problem solving tasks. Unfortunately, their use in time-critical applications often demands high performance hardware systems. Here we describe the implementation of a robot neurocontroller and its implementation on a Multiple-Instruction Multiple-Data (MIMD) Neural Network Processor (NNP[®]). Our neurocontrol concept is a hybridization of neural and adaptive methods, which combines the best characteristics of a classical adaptive controller and neural networks. The adaptive controller subsystem guarantees stability, robust tracking, and generality, while the neural subsystem provides parallel neural processing and improved learning. The neural-adaptive joint controller runs on a processor system whose architecture is optimized for such control problems and has been tested on two robot manipulators, with excellent results.

1 Introduction

Artificial Neural Networks (ANNs) are increasingly recognized as powerful tools for complex control tasks. Unfortunately, their use in time-critical applications often demands high performance, and therefore high cost hardware systems [4][8]. The neural network processor (NNP[®]) by Accurate Automation Corporation is a low cost realization of such dedicated Neural Network Hardware (NNH), and a hybrid neural-adaptive robot joint controller has been developed and implemented on the NNP[®]. The controller has been applied to two robot manipulators of very different dynamics, to which the controller must adapt itself. The hybrid system uses a “classical” adaptive controller to train a neural network with the network eventually learning to anticipate the response of the adaptive controller. This, in turn, yields a hybrid neural adaptive controller which

- Responds much faster to new commands or changes in the robot dynamics than the underlying adaptive controller, while
- Retaining the stability, robustness, and generality of the adaptive controller.

Our neural-adaptive joint controller is based upon the theory developed in [10]. This theory describes a generic model-reference adaptive controller. This controller allows robust tracking of a prescribed set of joint trajectories. The theory assumes only that the manipulator can be modeled as a set of loosely coupled second order differential equations with positive “slow-varying” mass. Modulo this minimal set of constraints, the adaptive controller is generic, automatically adapting itself to the robot kinematics and joint dynamics.

Not surprisingly, given the generic nature of the adaptive controller, the required feedback laws are both computationally intensive and slow to respond to changes in system dynamics. In our hybrid implementation neural networks are used to resolve both of these issues. A Functional Link network [5] is used to implement the adaptive feedback laws, thereby facilitating a parallel implementation of this computationally intensive process. Secondly, a neural network is trained to anticipate the feedback gains which would be produced by the adaptive controller and to initialize the adaptive controller with these gains at the start of each move. This hybrid approach yields the best of both worlds.

- The neural network greatly speeds up the response of the adaptive controller, while
- The adaptive controller remains in the loop to guarantee stability and robustness and to
- Retrain the neural controller whenever the robot's load, operating environment, or joint dynamics changes.

This neural-adaptive joint controller has been implemented on two different robots. The ESAM (Extensible Stiff Arm Manipulator) at Accurate Automation and the PFMA (Proto-Flight Manipulator Arm) robot operating in a simulated weightless environment at the NASA Marshall Space Flight Center. Even though the simulated weightless PFMA has totally different dynamics than the ESAM, the same control code performed as expected on both robots without modification.

In the following section, the architecture of the hybrid neural-adaptive joint controller is described followed by a description of the hardware implementation of the controller on the ESAM and PFMA robots, and an analysis of its performance.

2 Neural-Adaptive Control Architecture

The starting point for our hybrid neural-adaptive joint controller is a generic decentralized adaptive joint controller developed in [10]. This work assumes that each manipulator joint is modeled by a system of nonlinear second-order differential equations with positive slow varying masses, and that any joint coupling is sufficiently small and thus can be treated as a disturbance. Otherwise the algorithm is completely generic: no a-priori information about the mass, length, load, or frictional characteristics of the arm is required. The control law has a number of arbitrary initial parameters. While the stability theory still holds for any selection of these parameters, the performance can be affected greatly. Therefore, it is important to initialize these parameters carefully. We have developed a neural network which learns to initialize these parameters before each manipulator motion. The network improves the initial tracking performance significantly [1][2][3][6].

The architecture of our neural-adaptive controller is shown in Figure 1 for one manipulator joint. The adaptive controller subsystem is described above and is implemented as a functional link neural network [5]. It uses a reference input $\theta_r(t)$, its derivative, and the feedback error $e(t)$, to compute control torque $T(t)$ for the joint. The key to the architecture, however, is another neural network. This network updates the parameters of the adaptive controller for each reference joint angle, based on the prior performance of the adaptive controller under similar circumstances. As further learning trials are performed, better parameters are found and the adaptive controller works better. As the neural network estimates controller parameters from its observations, these parameters are associated with reference inputs. As multiple trials are performed, the network begins to interpolate between different points in joint space. Eventually, the network is able to select a good set of parameters for any reference trajectory that might be randomly selected.

3 Hardware Implementation

Although neural networks are characterized by massive parallelism, most neural network applications have been implemented on serial machines, where this benefit of parallel neural processing cannot be realized. Our hybrid neural-adaptive joint controller was implemented on an Accurate Automation Neural Network Processor (NNP[®]) hosted on a VME bus.

The NNP[®] is a Multiple Instruction-Multiple Data (MIMD) parallel processor whose architecture has been optimized for neural network applications [7][9]. Each NNP[®] supports up to 8K

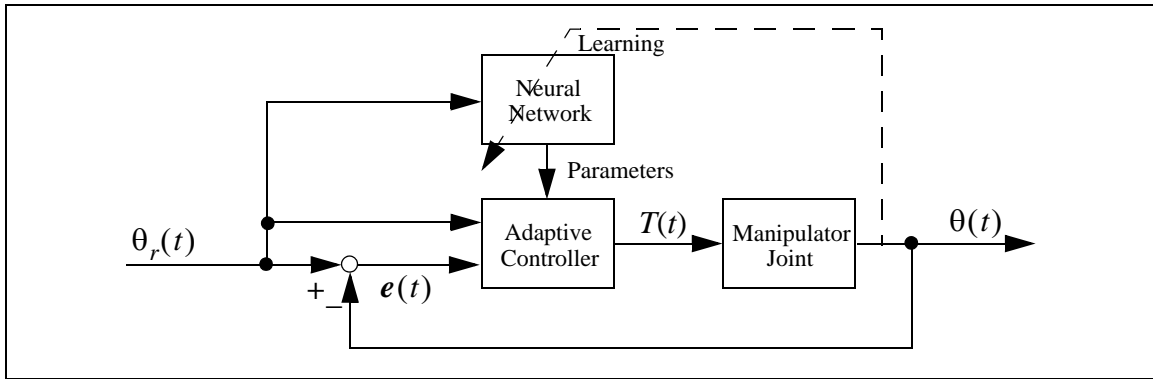


Figure 1: Neural-adaptive torque controller for one joint.

neurons, 32K weights, and runs at 140,000,000 connections-per-second (CPS) [4][8]. On the NNP[®], one CPS is equivalent to an 8 bit multiply-accumulate per second. An 8 NNP[®] system provides more than a billion CPS. The concept behind the development of the AAC neural network processor was to provide efficient Neural Network Hardware (NNH) for processing a broad variety of feedforward ANNs and recurrent ANNs with high speed at reasonable cost. The NNP[®]'s flexibility is based on its connection-based architecture and instruction set and allows the implementation of almost any connection topology. The resulting digital *special purpose* NNH consists of common electronic components and is optimized for certain neural processing tasks, i.e. its hardware resembles the local processing and distributed memory characteristics of ANNs.

Connections and activation functions are the most expensive computations in neural network implementations. Because the number of neurons in ANNs can be large, and because the number of weights can grow exponentially with the number of neurons, a NNH system must efficiently compute these connections. Computational speed can be further increased by an efficient implementation of neuron transfer functions. These two considerations were the driving force behind the NNP[®] design. A multiply-accumulate unit and the storage of transfer functions in lookup tables are the basis of the NNP[®] optimization (Figure 2). Sixteen-bit fixed point arithmetic further increases computational speed. Efficiency is increased even more by limiting the instruction set to the minimal set of instructions necessary to program neural networks. The cornerstone of an NNP[®] assembler program is the multiply-accumulate command, which multiplies one weight value with one input value and adds the product to the existing activation value. Instruction pipelining allows the completion of one command per clock cycle. In addition to the above features, computational speed can be substantially increased by using the NNP[®] in a multiprocessor environment. Up to eight NNP[®]'s in parallel form a Multiple Instructions-Multiple Data (MIMD) NNH system, whose performance increases approximately linearly with the number of NNPs[®].

The NNP[®] system takes the form of a mezzanine board mounted on a VME DSP/NNP[®] board built around two Texas Instruments TMS320-C40 Digital Signal Processors. The C-40's do the remaining pre and post processing, control an A/D-D/A board and the NNP[®] system, and communicate with a Graphical User Interface running on a Silicon Graphics workstation.

4 Experimental Results

We have implemented our controller on two radically different robots, the NASA Marshall Space Flight Center's ESAM and PFMA robotic manipulators. Although these two robots have radically different dynamics, with the ESAM arm operating in a traditional environment and the

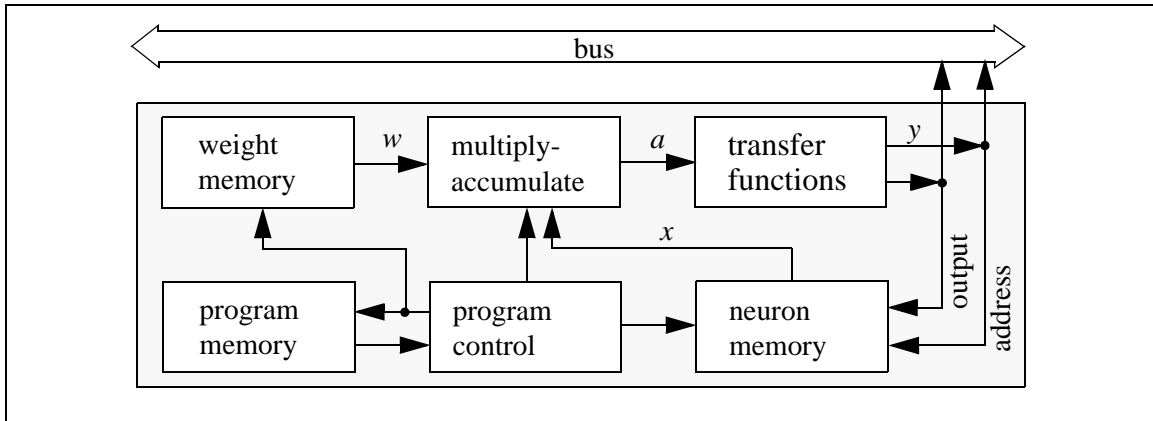


Figure 2: Block diagram of a single neural network processor (NNP[®]).

PFMA arm operating in a simulated weightless environment via a series of pulleys and counterweights, the same controller was used in both cases without modification. The ESAM experiments are described in [1].

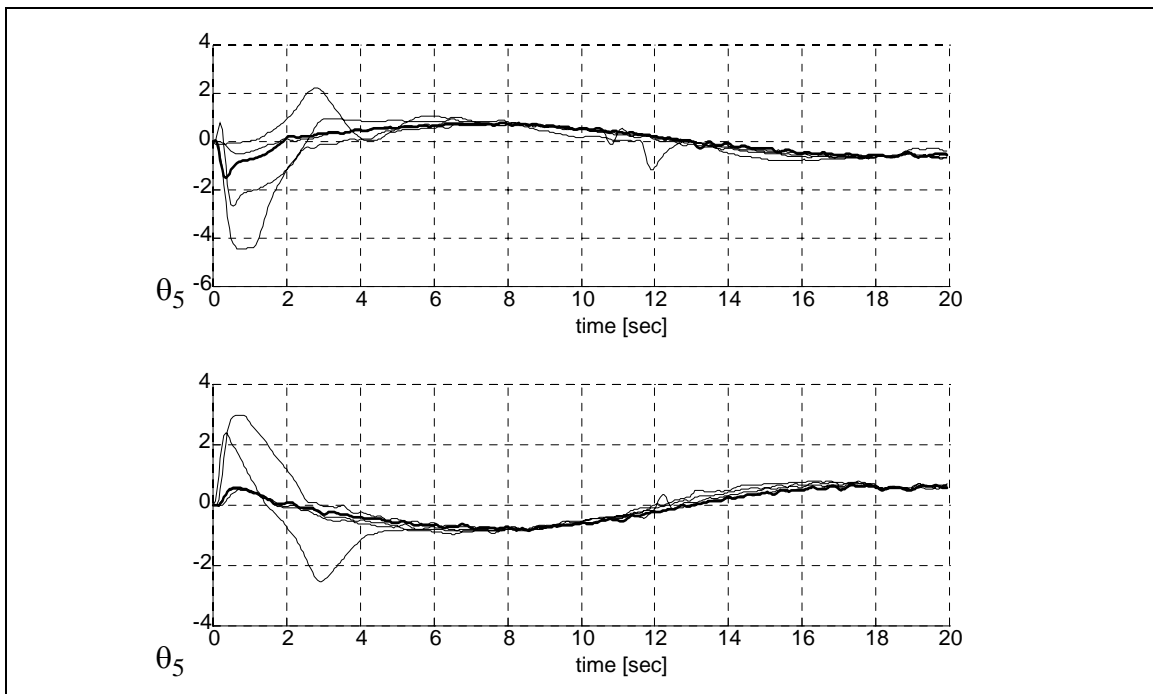


Figure 3: Tracking error in degrees for PFMA robotic manipulator (shoulder yaw angle).

Figure 3 shows the improvement of the tracking error for one of the PFMA's six joints during several alternating motions involving all six joints. The reference trajectories for each joint were cycloids, here the reference angles were sigmoids as functions of time and the first derivatives were bell curves. This trajectory allows a smooth movement with a gradual acceleration at the start of the motion and a gradual deceleration towards the end of the motion. For each joint, the specified manipulator movement was replicated three to five times with the neural-adaptive joint controller, improving the coefficients with each move. The upper portion of Figure 3 shows the tracking error for a wrist yaw from 15 degrees to 60 degrees. The lower portion shows the set of alternating motions from 60 degrees to 15 degrees. In the case of the first move, using "zero"

initial coefficients it takes approximately 18 seconds for the arm oscillation of this robot in a "simulated weightless" environment to damp out. For the second learning trial, the initial overshoot actually exceeds that of the first move but the oscillation damps out in about 6 seconds. Finally, with the third and fourth learning trials both the initial overshoot and the persistence of the oscillation is further reduced as the controller learns better coefficients. Similar learning characteristics were obtained for the other joints.

5 Conclusion

Our goal in the preceding has been to combine the best characteristics of a classical adaptive controller with those of a neurocontroller. The adaptive joint controller provides generality, guaranteed stability, and robust tracking, while neural networks provide massively parallel computation and learning. Our NNP[®] hardware system takes full advantage of the hybrid neural-adaptive joint controller by implementing each subsystem of the neurocontroller on a processor whose architecture has been optimized for that specific application. This, in turn, yields a joint controller whose performance, measured either in terms of computational speed or tracking error, exceeds that of either of its two antecedents.

6 Acknowledgment

This work is sponsored under the Small Business Innovation Research (SBIR) program by the National Aeronautics and Space Administration, Marshall Space Flight Center (code EB-24), contract number NAS8-38967. The NNP[®] was originally developed under SBIR funding from the Office of Naval Research, contract N00014-91-C-0268.

7 References

- [1] Cox C., J. Edwards, R. Saeks, R. Pap, and K. Mathia (1994), "Adaptive Semi-Autonomous Robotic Manipulator," *Proceedings of SPIE, Applications of Artificial Neural Networks V*, Vol. 2243, pp. 440-449.
- [2] Cox C., M. Lothers, R. Pap, and C. Thomas (1993), "A Neural Network for Joint and Motor Control," *Proc. of the World Congress on Neural Networks '93*, Portland, July 11-15, Vol. III, pp. 350-353.
- [3] Cox C., M. Lothers, R. Pap, and C. Thomas (1992), "A Neurocontroller for Robotics Applications," *Proc. Conf. on Systems, Man, and Cybernetics*, Chicago, Oct. 18-21, Vol. I, pp. 712-716.
- [4] Mathia K., J. Clark, B. Colbert, and R. Saeks (1996), "Benchmarking an MIMD Neural Network Processor," *Proc. World Congress on Neural Networks 1996*, San Diego, Sep. 15-18, pp. 1321-1326.
- [5] Pao Yoh-Han (1989), *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, Reading, Massachusetts.
- [6] Parten C., R. Pap, C. Thomas (1990), "Neurocontrol Applied to Telerobotics for the Space Shuttle," *Proc. International Neural Networks Conference*, Paris, France, July 9-13, pp. 229-236.
- [7] Priddy K., R. Saeks, R. Pap, and S. Stowell (1994), "Design of a MIMD Neural Network Processor," *Proc. SPIE Symp. on Neural Networks*, Orlando, April 5-8, Vol. 2243, pp. 318-323.
- [8] Rogers G., J. Solka, J. Ellis, and H. Szu (1992), "A Neurocomputing Benchmark for Digital Computers," *Proc. SIMTEC-WNN '92*, Clear Lake, Texas, pp. 425-430.
- [9] Saeks R., K. Priddy, K. Schnieder, and S. Stowell (1994), "On the Design of an MIMD Neural Network Processor," *Proc. World Cong. on Neural Networks '94*, San Diego, June 5-9, Vol. II, pp. 590-595.
- [10] Seraji H. (1989), "Decentralized Adaptive Control of Manipulators: Theory, Simulation, and Experimentation," *IEEE Trans. on Robotics and Automation*, Vol. 5, pp. 183-201.

Modern Adaptive Control With Neural Networks

Chad Cox, Dr. Karl Mathia, John Edwards, and Richard Akita

Accurate Automation Corporation
7001 Shallowford Road
Chattanooga, Tennessee 37421

Abstract

Artificial neural networks (ANNs) are increasingly recognized as powerful tools for complex problem solving tasks. Unfortunately, their use in time-critical applications often demands high performance hardware systems. Here we describe the implementation of a robot neurocontroller and its implementation on a Multiple-Instruction Multiple-Data (MIMD) Neural Network Processor (NNP[®]). Our neurocontrol concept is a hybridization of neural and adaptive methods, which combines the best characteristics of a classical adaptive controller and neural networks. The adaptive controller subsystem guarantees stability, robust tracking, and generality, while the neural subsystem provides parallel neural processing and improved learning. The neural-adaptive joint controller runs on a processor system whose architecture is optimized for such control problems and has been tested on two robot manipulators, with excellent results.

1 Introduction

Artificial Neural Networks (ANNs) are increasingly recognized as powerful tools for complex control tasks. Unfortunately, their use in time-critical applications often demands high performance, and therefore high cost hardware systems [4][8]. The neural network processor (NNP[®]) by Accurate Automation Corporation is a low cost realization of such dedicated Neural Network Hardware (NNH), and a hybrid neural-adaptive robot joint controller has been developed and implemented on the NNP[®]. The controller has been applied to two robot manipulators of very different dynamics, to which the controller must adapt itself. The hybrid system uses a “classical” adaptive controller to train a neural network with the network eventually learning to anticipate the response of the adaptive controller. This, in turn, yields a hybrid neural adaptive controller which

- Responds much faster to new commands or changes in the robot dynamics than the underlying adaptive controller, while
- Retaining the stability, robustness, and generality of the adaptive controller.

Our neural-adaptive joint controller is based upon the theory developed in [10]. This theory describes a generic model-reference adaptive controller. This controller allows robust tracking of a prescribed set of joint trajectories. The theory assumes only that the manipulator can be modeled as a set of loosely coupled second order differential equations with positive “slow-varying” mass. Modulo this minimal set of constraints, the adaptive controller is generic, automatically adapting itself to the robot kinematics and joint dynamics.

Not surprisingly, given the generic nature of the adaptive controller, the required feedback laws are both computationally intensive and slow to respond to changes in system dynamics. In our hybrid implementation neural networks are used to resolve both of these issues. A Functional Link network [5] is used to implement the adaptive feedback laws, thereby facilitating a parallel implementation of this computationally intensive process. Secondly, a neural network is trained to anticipate the feedback gains which would be produced by the adaptive controller and to initialize the adaptive controller with these gains at the start of each move. This hybrid approach yields the best of both worlds.

- The neural network greatly speeds up the response of the adaptive controller, while
- The adaptive controller remains in the loop to guarantee stability and robustness and to
- Retrain the neural controller whenever the robot's load, operating environment, or joint dynamics changes.

This neural-adaptive joint controller has been implemented on two different robots. The ESAM (Extensible Stiff Arm Manipulator) at Accurate Automation and the PFMA (Proto-Flight Manipulator Arm) robot operating in a simulated weightless environment at the NASA Marshall Space Flight Center. Even though the simulated weightless PFMA has totally different dynamics than the ESAM, the same control code performed as expected on both robots without modification.

In the following section, the architecture of the hybrid neural-adaptive joint controller is described followed by a description of the hardware implementation of the controller on the ESAM and PFMA robots, and an analysis of its performance.

2 Neural-Adaptive Control Architecture

The starting point for our hybrid neural-adaptive joint controller is a generic decentralized adaptive joint controller developed in [10]. This work assumes that each manipulator joint is modeled by a system of nonlinear second-order differential equations with positive slow varying masses, and that any joint coupling is sufficiently small and thus can be treated as a disturbance. Otherwise the algorithm is completely generic: no a-priori information about the mass, length, load, or frictional characteristics of the arm is required. The control law has a number of arbitrary initial parameters. While the stability theory still holds for any selection of these parameters, the performance can be affected greatly. Therefore, it is important to initialize these parameters carefully. We have developed a neural network which learns to initialize these parameters before each manipulator motion. The network improves the initial tracking performance significantly [1][2][3][6].

The architecture of our neural-adaptive controller is shown in Figure 1 for one manipulator joint. The adaptive controller subsystem is described above and is implemented as a functional link neural network [5]. It uses a reference input $\theta_r(t)$, its derivative, and the feedback error $e(t)$, to compute control torque $T(t)$ for the joint. The key to the architecture, however, is another neural network. This network updates the parameters of the adaptive controller for each reference joint angle, based on the prior performance of the adaptive controller under similar circumstances. As further learning trials are performed, better parameters are found and the adaptive controller works better. As the neural network estimates controller parameters from its observations, these parameters are associated with reference inputs. As multiple trials are performed, the network begins to interpolate between different points in joint space. Eventually, the network is able to select a good set of parameters for any reference trajectory that might be randomly selected.

3 Hardware Implementation

Although neural networks are characterized by massive parallelism, most neural network applications have been implemented on serial machines, where this benefit of parallel neural processing cannot be realized. Our hybrid neural-adaptive joint controller was implemented on an Accurate Automation Neural Network Processor (NNP[®]) hosted on a VME bus.

The NNP[®] is a Multiple Instruction-Multiple Data (MIMD) parallel processor whose architecture has been optimized for neural network applications [7][9]. Each NNP[®] supports up to 8K

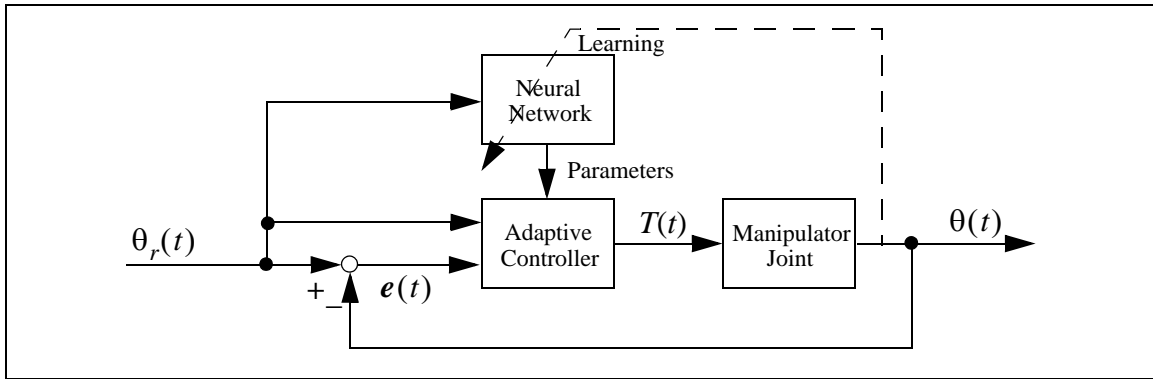


Figure 1: Neural-adaptive torque controller for one joint.

neurons, 32K weights, and runs at 140,000,000 connections-per-second (CPS) [4][8]. On the NNP[®], one CPS is equivalent to an 8 bit multiply-accumulate per second. An 8 NNP[®] system provides more than a billion CPS. The concept behind the development of the AAC neural network processor was to provide efficient Neural Network Hardware (NNH) for processing a broad variety of feedforward ANNs and recurrent ANNs with high speed at reasonable cost. The NNP[®]'s flexibility is based on its connection-based architecture and instruction set and allows the implementation of almost any connection topology. The resulting digital *special purpose* NNH consists of common electronic components and is optimized for certain neural processing tasks, i.e. its hardware resembles the local processing and distributed memory characteristics of ANNs.

Connections and activation functions are the most expensive computations in neural network implementations. Because the number of neurons in ANNs can be large, and because the number of weights can grow exponentially with the number of neurons, a NNH system must efficiently compute these connections. Computational speed can be further increased by an efficient implementation of neuron transfer functions. These two considerations were the driving force behind the NNP[®] design. A multiply-accumulate unit and the storage of transfer functions in lookup tables are the basis of the NNP[®] optimization (Figure 2). Sixteen-bit fixed point arithmetic further increases computational speed. Efficiency is increased even more by limiting the instruction set to the minimal set of instructions necessary to program neural networks. The cornerstone of an NNP[®] assembler program is the multiply-accumulate command, which multiplies one weight value with one input value and adds the product to the existing activation value. Instruction pipelining allows the completion of one command per clock cycle. In addition to the above features, computational speed can be substantially increased by using the NNP[®] in a multiprocessor environment. Up to eight NNP[®]'s in parallel form a Multiple Instructions-Multiple Data (MIMD) NNH system, whose performance increases approximately linearly with the number of NNPs[®].

The NNP[®] system takes the form of a mezzanine board mounted on a VME DSP/NNP[®] board built around two Texas Instruments TMS320-C40 Digital Signal Processors. The C-40's do the remaining pre and post processing, control an A/D-D/A board and the NNP[®] system, and communicate with a Graphical User Interface running on a Silicon Graphics workstation.

4 Experimental Results

We have implemented our controller on two radically different robots, the NASA Marshall Space Flight Center's ESAM and PFMA robotic manipulators. Although these two robots have radically different dynamics, with the ESAM arm operating in a traditional environment and the

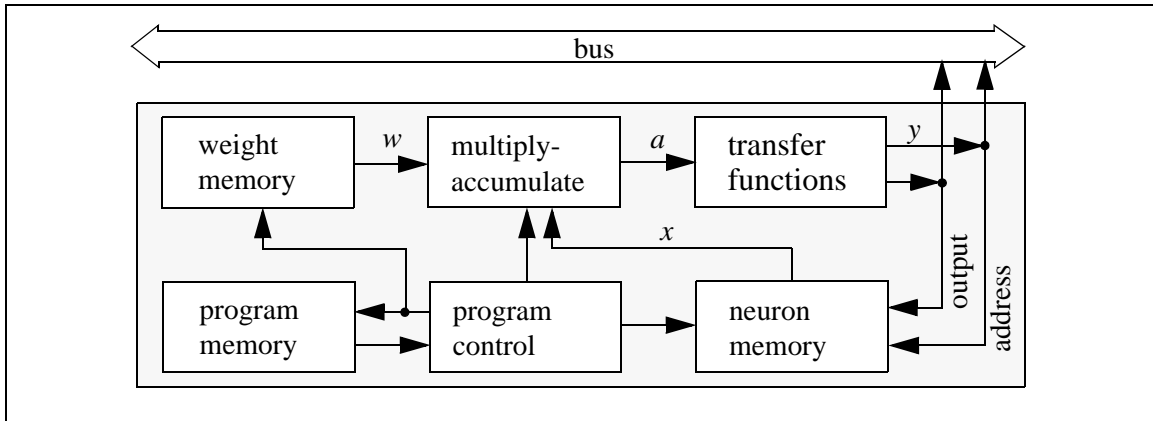


Figure 2: Block diagram of a single neural network processor (NNP[®]).

PFMA arm operating in a simulated weightless environment via a series of pulleys and counterweights, the same controller was used in both cases without modification. The ESAM experiments are described in [1].

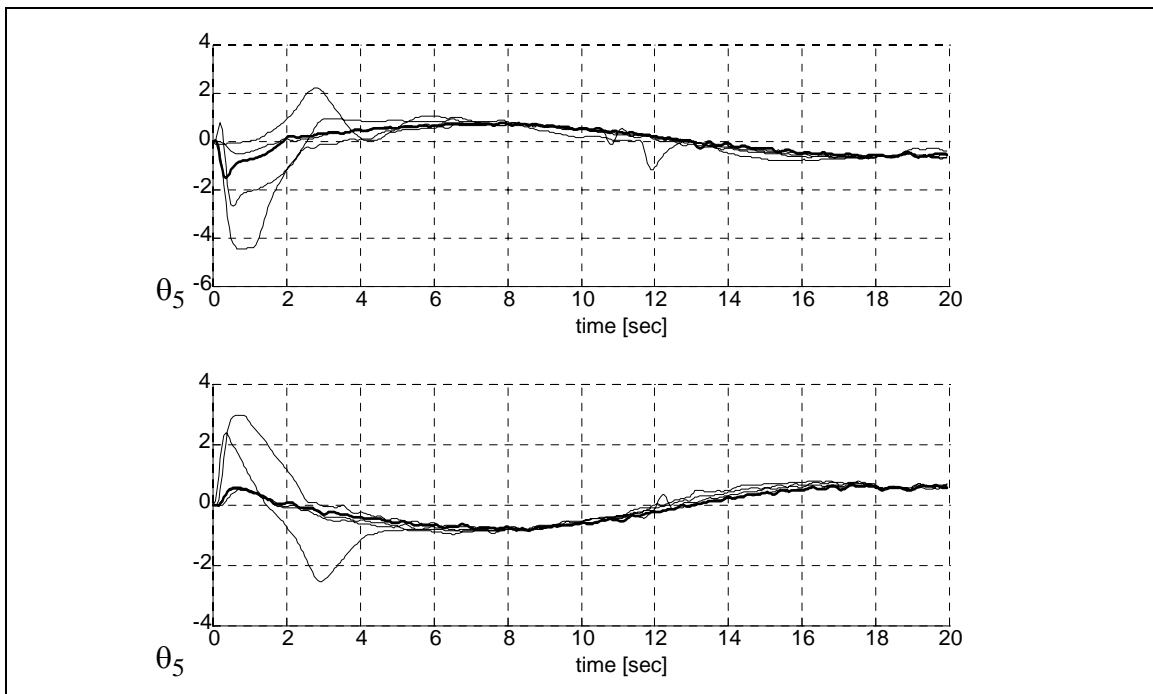


Figure 3: Tracking error in degrees for PFMA robotic manipulator (shoulder yaw angle).

Figure 3 shows the improvement of the tracking error for one of the PFMA's six joints during several alternating motions involving all six joints. The reference trajectories for each joint were cycloids, here the reference angles were sigmoids as functions of time and the first derivatives were bell curves. This trajectory allows a smooth movement with a gradual acceleration at the start of the motion and a gradual deceleration towards the end of the motion. For each joint, the specified manipulator movement was replicated three to five times with the neural-adaptive joint controller, improving the coefficients with each move. The upper portion of Figure 3 shows the tracking error for a wrist yaw from 15 degrees to 60 degrees. The lower portion shows the set of alternating motions from 60 degrees to 15 degrees. In the case of the first move, using "zero"

initial coefficients it takes approximately 18 seconds for the arm oscillation of this robot in a "simulated weightless" environment to damp out. For the second learning trial, the initial overshoot actually exceeds that of the first move but the oscillation damps out in about 6 seconds. Finally, with the third and fourth learning trials both the initial overshoot and the persistence of the oscillation is further reduced as the controller learns better coefficients. Similar learning characteristics were obtained for the other joints.

5 Conclusion

Our goal in the preceding has been to combine the best characteristics of a classical adaptive controller with those of a neurocontroller. The adaptive joint controller provides generality, guaranteed stability, and robust tracking, while neural networks provide massively parallel computation and learning. Our NNP[®] hardware system takes full advantage of the hybrid neural-adaptive joint controller by implementing each subsystem of the neurocontroller on a processor whose architecture has been optimized for that specific application. This, in turn, yields a joint controller whose performance, measured either in terms of computational speed or tracking error, exceeds that of either of its two antecedents.

6 Acknowledgment

This work is sponsored under the Small Business Innovation Research (SBIR) program by the National Aeronautics and Space Administration, Marshall Space Flight Center (code EB-24), contract number NAS8-38967. The NNP[®] was originally developed under SBIR funding from the Office of Naval Research, contract N00014-91-C-0268.

7 References

- [1] Cox C., J. Edwards, R. Saeks, R. Pap, and K. Mathia (1994), "Adaptive Semi-Autonomous Robotic Manipulator," *Proceedings of SPIE, Applications of Artificial Neural Networks V*, Vol. 2243, pp. 440-449.
- [2] Cox C., M. Lothers, R. Pap, and C. Thomas (1993), "A Neural Network for Joint and Motor Control," *Proc. of the World Congress on Neural Networks '93*, Portland, July 11-15, Vol. III, pp. 350-353.
- [3] Cox C., M. Lothers, R. Pap, and C. Thomas (1992), "A Neurocontroller for Robotics Applications," *Proc. Conf. on Systems, Man, and Cybernetics*, Chicago, Oct. 18-21, Vol. I, pp. 712-716.
- [4] Mathia K., J. Clark, B. Colbert, and R. Saeks (1996), "Benchmarking an MIMD Neural Network Processor," *Proc. World Congress on Neural Networks 1996*, San Diego, Sep. 15-18, pp. 1321-1326.
- [5] Pao Yoh-Han (1989), *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, Reading, Massachusetts.
- [6] Parten C., R. Pap, C. Thomas (1990), "Neurocontrol Applied to Telerobotics for the Space Shuttle," *Proc. International Neural Networks Conference*, Paris, France, July 9-13, pp. 229-236.
- [7] Priddy K., R. Saeks, R. Pap, and S. Stowell (1994), "Design of a MIMD Neural Network Processor," *Proc. SPIE Symp. on Neural Networks*, Orlando, April 5-8, Vol. 2243, pp. 318-323.
- [8] Rogers G., J. Solka, J. Ellis, and H. Szu (1992), "A Neurocomputing Benchmark for Digital Computers," *Proc. SIMTEC-WNN '92*, Clear Lake, Texas, pp. 425-430.
- [9] Saeks R., K. Priddy, K. Schnieder, and S. Stowell (1994), "On the Design of an MIMD Neural Network Processor," *Proc. World Cong. on Neural Networks '94*, San Diego, June 5-9, Vol. II, pp. 590-595.
- [10] Seraji H. (1989), "Decentralized Adaptive Control of Manipulators: Theory, Simulation, and Experimentation," *IEEE Trans. on Robotics and Automation*, Vol. 5, pp. 183-201.