A Neural Network Approach for Generating Derivative Information from Quantized Position Measurements

Asif Nadeem Zaidi

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented as Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science
Concordia University
Montreal, Quebec, Canada

October 1995

0-612-44875-4

Abstract

A Neural Network Approach for Generating Derivative Information from Quantized Position Measurements

Asif Nadeem Zaidi

In the control of robot and other mechanized systems, there exists a need for the generation of first and higher order derivative information. Not much effort has been made to address this issue. Instead much of the work, especially in robot control, has relied on *a priori* knowledge of the system dynamics to determine the derivative information.

The goal of this thesis is to propose a generalized methodology to determine first and higher order derivative information without any *a priori* knowledge of a system's dynamics. For illustration purposes the system that we will concentrate on is a flexible joint robot manipulator. In achieving our objective, we impose three restrictions on the resulting methodology. First, the acquisition of the derivative information required for the control of a robot manipulator must be done without assuming knowledge of the robot's joint position or its dynamics. Second, we must not use any additional hardware than is absolutely necessary. Thus, for our flexible manipulator system, we must only use a sensor to measure the position variable. The derivative information (velocity, acceleration and jerk) must be determined by the proposed methodology. Third, the derivative information must be determined in real time.

We accomplish the objectives using Tap Delay Neural Networks (TDNN). We incorporate the proposed methodology into the control system of a flexible joint manipulator. Numerous simulation results show the success of our proposed scheme within robot control loops. Additionally, we compare the performance of a number of alternatives within this scheme with that using conventional linear filtering techniques. Again, we see the superiority of the proposed TDNN scheme over the linear filtering approach.

To my parents -
Mohammad Zaidi & Azra Hameed

Acknowledgments

First and foremost, I would like to thank my family for their patience and support throughout my education.

I would like to thank my supervisors Dr. Baher S. Haroun and Dr. Rajnikant V. Patel for their guidance of this thesis. I would also like to Dr. Asim Al-Khalili for guiding this thesis to its end.

This work was supported by Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR) of the Province of Quebec under Grant ER-1042. The author wishes to acknowledge this support.

Table of Contents

List of Figures

Abbreviations

ANN	Artificial Neural Network
DSP	Digital Signal Processing
FIR	Finite Impulse Response
IIR	Infinite Impulse Response
LED	Light Emitting Diode
TDNN	Tap Delay Neural Network
ASIC	Application Specific Integrated Circuit

Chapter 1
Introduction

Over the last decade, interest in the control of flexible-joint robots has increased significantly. Flexible-joint robots bring an added complexity which is not found in rigid robots. This new dimension enables the robot to function with an accuracy and reliability which can not be achieved when flexibility is ignored. This advantage also brings with it the difficult task of controlling the manipulator. This is because the bandwidth of the system has increased significantly and dynamic effects which previously were beyond the frequency range of interest in rigid manipulators now have to be considered in the controller design. Consequently, the techniques that have been utilized in the control of rigid manipulators [2] cannot be applied directly to the control of flexible-joint manipulators. The control of rigid joint manipulators typically employs measurements of the position and velocity of the joints - the states of the system. A typical flexible-joint manipulator state consists of position, velocity, acceleration and jerk (rate of change of acceleration). Therefore, control requires the additional measurements of the acceleration and jerk components. The inclusion of two additional quantities adds several dimensions of complexity with respect to the task of data acquisition. The majority of sensor systems made are primarily for determining the position component. We can find sensor systems to measure velocity and acceleration but finding sensor systems to measure jerk is impossible. The overall weight of a sensor system is also an issue when it is incorporated to measure the desired signals in a flexible robot manipulator. It is necessary to ensure that the overall weight of the sensor system does not over-burden the manipulator. If this constraint is not satisfied then there exists a risk of significantly altering the dynamics of the robot manipulator. Furthermore, the task of implementing a sensor system to measure a variable is made more complicated since a mechanism must be put in place to filter the quantization noise. A sensor implicitly quantizes the input variable and, as is well known, quantization adds noise to the resulting signal. Thus the sensor system must ensure that the effect of quantization noise is reduced as much as possible. This in itself is a problem and it adds to the cost of the overall sensor system. Another limitation that is placed is that the acquisition of the control parameters must be done in real time. This is especially true in control systems or DSP systems where speed is of the essence. A system is worthless if it cannot satisfy this constraint. Thus the problem of finding an effective methodology to determine the higher-order derivatives remains an open area of research.

Therefore, the aim of this research work is to design a methodology to determine higher-order derivative information. In order to illustrate our design, we shall concentrate on the information that is required for the control of a flexible-joint manipulator from quantized position measurements. To achieve this end, we impose three requirements on the proposed methodology. First, the acquisition of the parameters required for the control of the system must be achieved without any knowledge of the dynamics of the system. Second, a minimum amount of hardware must be utilized and third, acquisition of the signals must be done in real time.

Previously, two schemes have been employed to determine higher-order derivative information. One scheme is to have a mathematical knowledge of the system and use it to derive a formulation from which we can calculate the required values. The second scheme is to use physical sensors to measure each feedback signal. The problem associated with the first scheme is that a knowledge of the dynamics of the system is required, whereas the problem with the second scheme is the use of additional hardware. This poses a problem since it increases the weight of the overall flexible-joint manipulator and thus alters the system dynamics. Also, despite rapid advances in micro-sensor and VLSI technologies, sensors to measure velocity and acceleration are fraught with errors and, as was said earlier, sensors to measure jerk are non-existent. The previous control algorithms for flexible-joint robots have implicitly assumed that the required quantities (position, velocity, acceleration and jerk) are readily available with high accuracy. In this thesis we show that such an assumption ignores several practical difficulties and maybe detrimental to the design of the overall control algorithm.

Next, we discuss case studies where higher-order derivative information is required for the control of a system. In reviewing these works two things should be noticed. First, the authors have not indicated how the signals required for the control will be generated. Second, there is a wide variety of applications which require the use of higher-order derivative information. Thus it is of vital importance to have a standardized methodology to generate higher-order derivative information.

1.1 Examples of systems where derivative information is used

In [3], Charney and Josin replaced a PID controller [4] by a neural network controller for manipulator joint control. PID controllers are essentially designed for linear time-invariant systems and have constant gains. However, in a dynamic system like a robot manipulator a controller is required which can function well for a non-linear, time-varying system. The use of a neural network as a servo controller adds the ability for adaptation for non-linear control of a robot manipulator system. The authors tested the two controllers on their manipulator under different joint loading conditions. The results indicated that the neural network controller was superior in terms of tracking the desired trajectory when compared to the PID controller. The control system setup which was used is shown in Figure 1. Note the use of the derivatives to compute velocity feedback.

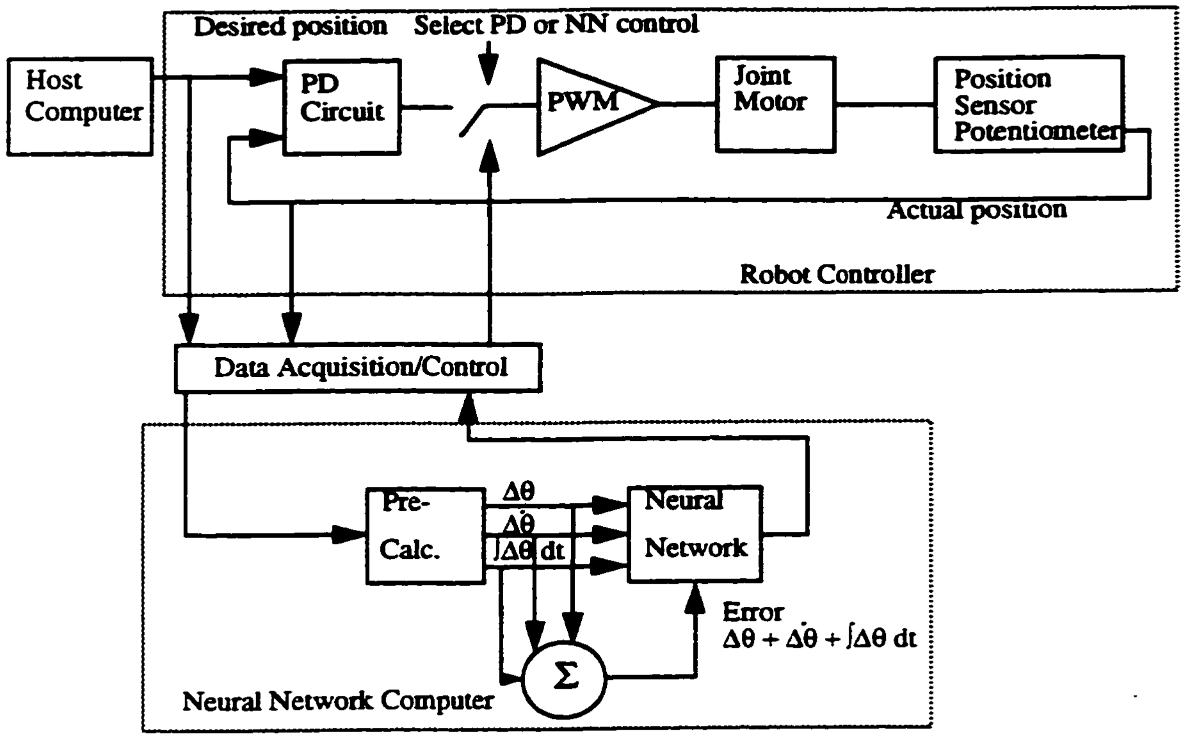


Figure 1 : Charney and Josin's control scheme.

The neural network used is of a feedforward nature consisting of three inputs and one output. In addition to the input layers, there are two hidden layers consisting of a sigmoidal activation function, and an output layer consisting of a linear activation function. The inputs to the network were determined by considering the response of the physical joint to be controlled which in this case are position, velocity and acceleration. The inputs to the network are defined as:

$$\Delta\theta = \theta_{comamnd} - \theta_{actual}$$

$$\Delta\dot{\theta} = \frac{d}{dt}\Delta\theta = \Delta\theta_t - \Delta\theta_{t-1}$$

$$\int \Delta\theta dt = \frac{\Delta\theta_t + \Delta\theta_{t-1}}{2}$$

The next work we explore deals with the control of an aircraft flight [5]. The objective of this research work is to develop a neural based aircraft fight control scheme. The crux of the research is an analysis of the trade-off in achieving adequate performance versus control in the presence of actuator nonlinearities. An aircraft control design was chosen with two control inputs and two controlled outputs. The control design problem was set up as that of following the trajectories generated from a model of the desired vehicle response dynamics from pilot command inputs.

The training architecture is represented in Figure 2. For each pilot selected trajectory $\bar{z}_{SEL}(t)$, a commanded trajectory $\bar{z}_c(t)$ is generated. Prior to training, the variables $\bar{z}_c(t)$ are discretized and scaled into $\bar{z}_c^s(t_k) = [V_c(t_k)/V_c^0, Q_c(t_k)/Q_c^0]$, where V_c^0 and Q_c^0 are of the order of magnitude of the maximum values of $V_c(t)$ (aircraft velocity) and $Q_c(t)$ (pitch rate) respectively.

As shown in Figure 2, two control inputs $\bar{u}$ are calculated by a two-hidden layer feedforward net that has eight inputs (or four inputs associated to the *Q* and *V* variables) and two neurons in the output layer. These pairs consist of the scaled output vector $\bar{z}^s(t_k)$ (Term D); the tracking error $\bar{e}_z(t_k)$ (Term C) between the scaled vehicle output vector $\bar{z}^s(t_k)$ and its desired scaled value at time t_{k+1}; the discrete time derivative of the tracking error $\dot{\bar{e}}_z(t_k)$ (Term B); and the time average of the tracking error $\frac{\int_0^{t_k} \bar{e}_z(t)}{t_k}$ (Term A). Each neuron has the standard activation function: *y=tanh(x).*

As we see from the above input parameters, they are composed of a sum of the tracking errors and control input commands and rates. This is termed the "**objective function**" and it is the task of the neural network to minimize this function.

Once the training was finished, the trained neural network was placed in the control system as shown in Figure 3. It was shown in the results that the neurocontrol learned with such a training results in very accurate control.

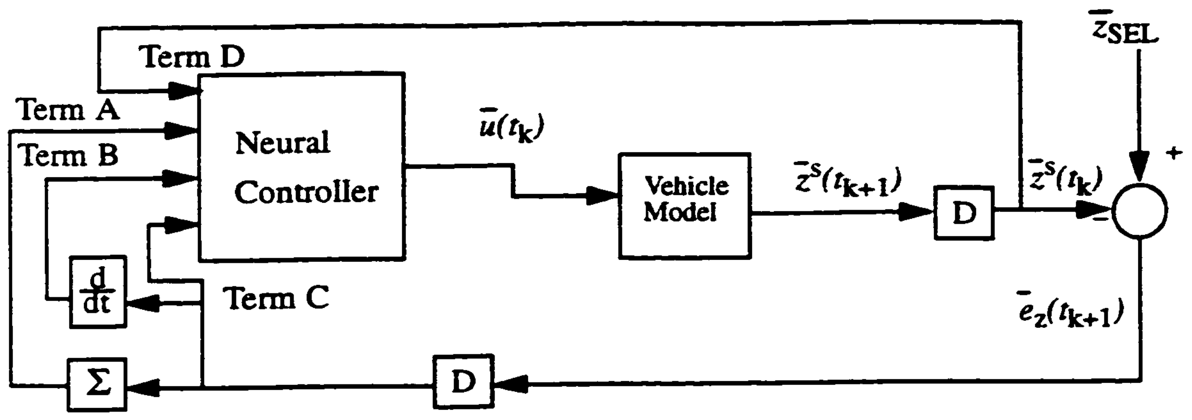


Figure 2 : Training architecture.

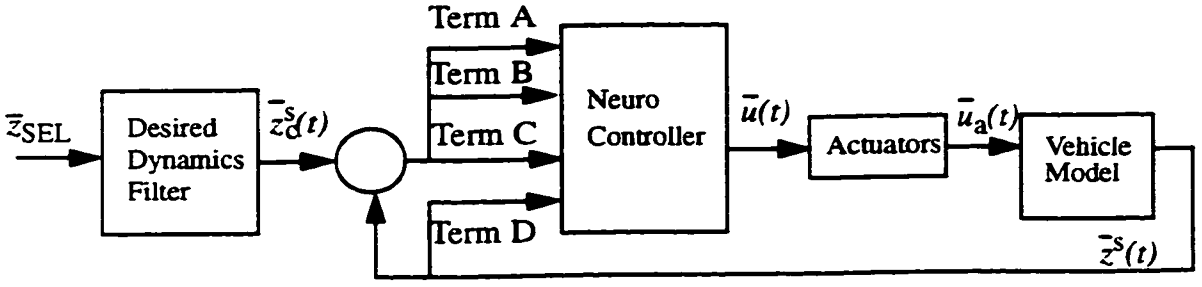


Figure 3 : Troudet's evaluation of closed-loop controller architecture.

The last system we examine is a flexible-joint robot manipulator for which the control scheme is shown in Figure 4.

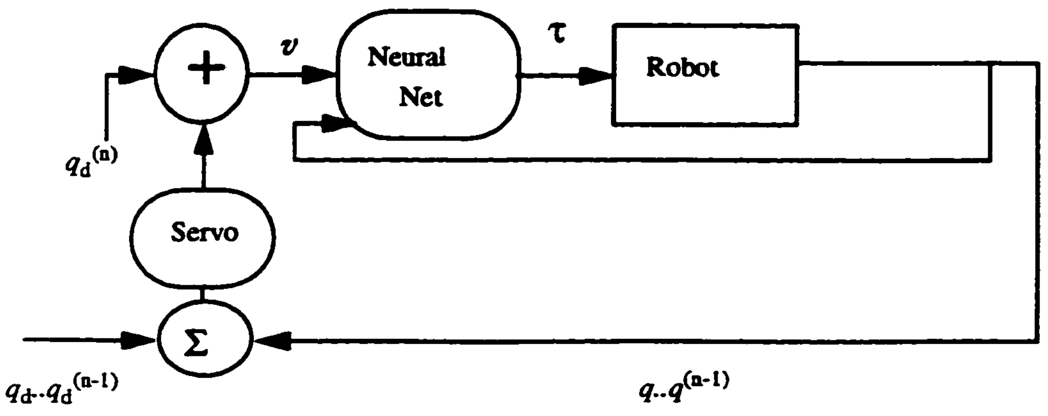


Figure 4 : Zeman's control scheme.

As we can see a partitioned control strategy has been employed. The inner loop has the responsibility of linearizing the plant. A neural network that was trained off-line is used here. This neural net serves as the controller. For a single-joint manipulator with a dynamical model of order *n* (which is four in this case), this network will have *n+1* inputs and a single output. The first *n* inputs $q, \dot{q}, \ddot{q}, \ldots, q^{n-1}$ (in this case position, velocity, acceleration, jerk) are connected to the plant. The highest order net input is connected to the outer loop feedback signal *v*. The inner loop is closed by connecting the net's output *u* to the robot's input (driving torque). Thus any given input vector $q, \dot{q}, \ddot{q}, \ldots, q^{n-1}, v$ will produce a unique net output *u*. The neural network controller and the inner loop together form the model-based portion of the controller, so named because it models the inverse dynamics of the robot. The second portion of the controller is the outer loop, commonly referred to as the servo portion. The servo design treats the robot as a black-box and does not need any information from the robot with regard to its dynamic structure. Only the order of the system (four in the case of a single joint manipulator) is needed. A linear feedback strategy (PD control) is applied to the outer loop. The servo error is defined as

$$e^4 + k_3 e^3 + k_2 e^2 + k_1 e^1 + k_0 e^0 = 0 \quad (1)$$

where k_0, k_1, k_2, k_3 are feedback gain constants which can be set to provide any desired closed-loop system poles without any specific knowledge of the system dynamics. The resulting control law (v) is given by

$$v = q_d^4 + k_3(q_d^3 - q^3) + k_2(q_d^2 - q^2) + k_1(q_d^1 - q^1) + k_0(q_d - q) \tag{2}$$

Their work showed that for a manipulator with unspecified nonlinear dynamics a control strategy can be devised. However, one of the limitations that was observed in their work was the need to generate the higher-order derivatives. These were assumed to be accurately generated. A mathematical model of the non-linear dynamical system was used to determine the derivatives. We use this system in this thesis as a demonstration of our approach for higher-order derivative estimation. Our methodology modifies the control loop by incorporating a structure called the "**derivative estimator**". This will be explained later in Chapter 3.

We now give a brief synopsis of the research work that has been done in systems that require the use of higher-order derivative information.

In [6], the authors discuss the performance of a neural network controller with a non-linear plant in a feedback control system in the presence of different noise disturbances. They concluded that a neural network controller trained with noisy data adapts better to the presence of disturbances than a neural network controller trained with clean data. While they achieved their initial aims their work was incomplete in that they did not address how they were going to generate the higher-order derivative information. This is especially important in this case since their system (a one-link rigid manipulator) required the velocity feedback.

In [7], they propose a position/force hybrid control of a robotic manipulator based on neural networks. The proposed method had shown a wider applicability with respect to changes in the manipulator's orientation. Again the control system required the use of derivative information and this fact was not addressed.

The same conclusion can be drawn in [8], with respect to the generation of derivative information. This research work is devoted to the problem of controlling a class of dynamic systems via controllers based on neural networks.

In all the above mentioned works (and, in general, for most physical systems), the overall control scheme involves the use of first and/or higher-order derivative information. An assumption is made that the derivative information is available and accurate. For example, in [1] it is said explicitly "*... we will assume that the required derivatives (of joint positions) can somehow be obtained.*" We will demonstrate in Chapter 3 that in practice this assumption does not always hold. Additional means to obtain the derivative information must be incorporated when implementing the control methodology.

1.2 Motivation and overview of thesis

We have indicated the purpose of this research work - namely that of deriving a methodology which will enable one to generate the derivative information required for the control of a flexible-joint manipulator. We have shown examples of systems where derivative information was used for the control of their respective systems. However, all of the above examples and others usually omit the consideration of how this information is going to be determined. It is implicitly assumed that the feedback quantities can be accurately determined by appropriate instrumentation. We show in this thesis that disregard of this fact is detrimental to the overall performance of the control algorithm. We propose a neural network structure to perform derivative estimation. Our criterion for judging the success of our neural network system is based on how well the overall control system will track the desired trajectory.

Our approach incorporates the use of a sensor to measure the position component. To model the quantization ability of a sensor, we implement a quantizer. The resolution of this quantizer can be adjusted by the control engineer. We investigate the effect of changing the bit resolution of the quantizer on the tracking of the desired trajectory. We then use neural networks to generate the derivative information from this position parameter - i.e. we will build neural network differentiators. Using this approach satisfies our three constraints of using a minimum amount of hardware, eliminating our dependence on a mathematical knowledge of the system and obtaining the system response in real time. Next, we explore neural network architectures where the total number of computations is reduced thereby reducing the real time required to obtain the derivative information. Again we judge the success of our derivative estimator based on the criterion specified above.

In Chapter 2, we will give a brief review on some background material that we need for understanding this work. We discuss three areas. First, we discuss the need for a model-free estimation procedure in robot control. Second, we discuss the different types of sensors that are generally available. Third, we give a brief review of DSP theory with special emphasis on quantization effects, and techniques used for digital differentiation. In Chapter 3, we discuss the methodology that was employed to generate the derivative information from position measurements. We show the results of our proposed methodology in Chapter 4. As we mentioned earlier, any methodology must ensure that the feedback quantities are obtained in real time. This is the subject of discussion in Chapter 5. We explore neural network architectures where the number of computations is reduced substantially and yet the system specifications are met. We conclude the work in Chapter 6.

Chapter 2
Neural Networks, Sensors and DSP

This chapter introduces the concepts of neural networks, sensors and digital signal processing as they are used in the context of this thesis. The material presented here is not new, but rather it is a review of the areas mentioned above. References are given at each point which the reader can then examine for further details.

We first start by giving a brief discussion on a type of neural networks known as Tap Delay Neural Networks (TDNNs). In this thesis, we study the control of a flexible-joint robot manipulator system. We give a brief model of the robot which indicates the relationship between the force required to move the manipulator along a desired trajectory and its states. We see the dependence of this force (torque) on the generation of derivatives of the joint position. We then proceed to describe a sensor system that can be used to measure the position variable. Finally, in the last section, we show conventional digital filtering techniques to generate the higher-order derivative information. We also review previous research work done in the area of digital differentiators. In this section, we pay special attention to quantization effects and comment on the result of differentiating a quantized signal.

2.1 Tap Delay Neural Networks

Tap Delay Neural Networks (TDNNs) have been used mainly in the field of speech-processing. They are used in an attempt to account for the dynamically changing nature of speech by building temporal delays in a multi-layer perceptron network and integrating information across time in the upper layers of the network. This property of representing the relationships between events in time is very attractive in building neural network differentiators. The most common technique to arrive at approximate higher-order derivatives is by finite differences. This is a standard procedure in numerical analysis and in digital simulations of analog systems. Using the methodology of finite differences involves taking the differences of consecutive samples from the original signal. This is illustrated in equation (3):

$$\frac{d}{dt}y_n(t)\Big|_{t=nT} = \frac{y(n) - y(n-1)}{T} \tag{3}$$

where T is the sample rate of the differentiator. If more accurate differentiation is required then the following algorithm should be applied to the digital signal:

$$\frac{d}{dt}y_n(t)\Big|_{t=nT} = \frac{y(n) - a_1 y(n-1) - a_2 y(n-2) \ldots - a_d y(n-D)}{T} \tag{4}$$

where D can be any value depending on the amount of accuracy one wants. This ability of a TDNN to represent relationships between events in time cannot be found in conventional neural networks and thus TDNNs will be used. The most extensive use of TDNNs comes in the research work of Waibel [9, 10]. In a TDNN, each cell in the network effectively receives not only its nominal input vector but "delayed" values of the input so that it can learn local correlations in the data. A schematic of a TDNN is shown in Figure 5:

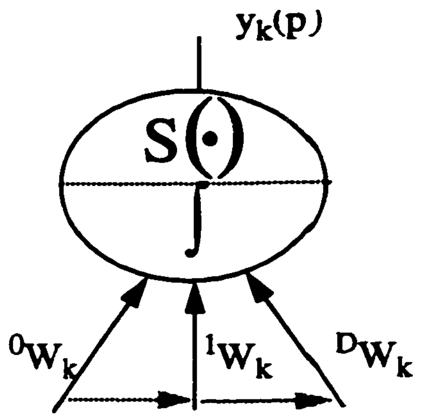


Figure 5 : The TDNN cell. The *D* delayed values of this vector are also entered. ${}^DW_k$ represents the weight vector on the *dth* delayed copy of the input.

At time "*p*", the layer 1 cells receive the "present" plus "D" (assumed two for this example) delays of the input vectors, say x(*p*), x(*p*-1), x(*p*-2), while layer 2 receives the present plus four delays of the output of layer 1, say y(*p*),...y(*p*-4). The output of the TDNN is given as

$$y_k(p) = S\left(\sum_{d=0}^{D} {}^dw_k y'(p-d)\right) \tag{5}$$

The S() can be any activation function - piecewise linear, step, hyperbolic tangent etc. [14].

2.1.1 Robotics and Neural Networks

In this section we will concentrate on one particular area where neural networks are used - namely robotics. Robot manipulators are essentially open-chain kinematic mechanisms. There is coupling between motions of individual segments. To further exacerbate the problems, the parameters of a manipulator are dependent upon its configuration, and the governing equations are highly non-linear [11]. The desired trajectory of the endpoint of the arms is specified in Cartesian space, while motions are actually obtained from actuators located at the joints. The trans-formation from Cartesian to joint co-ordinates is a computationally intensive problem and is dependent on the algorithm used and on knowledge of the robot parameters [12, 13].

As we see, the control of robot manipulators requires a knowledge of the mathematical properties of the system. In many cases, it is almost impossible to determine this accurately. In lieu of this, some means must exist whereby this dependency can be learned or predicted. Artificial Neural Networks (ANN) offer a viable option. ANN's can in principle be trained to approximate relations between variables regardless of their analytical dependency [14]. Hence, it is possible to solve the robot control problem with model-free estimation systems - using ANN's trained with a large number of examples.

2.1.1.1 Robotics Problems

There are three fundamental problems in the control of a robot manipulator: task planning, trajectory planning and motion control. **Task planning** phase is concerned with information management and co-ordination of the job to be performed. **Trajectory planning** involves finding the sequence of points through which the manipulator end point must pass. When a trajectory is given, the **motion control** problem is invoked. This involves the determination of joint torques which will allow the arm to follow the desired trajectory while satisfying the physical constraints of the robot.

ANN's find applications in all the areas mentioned above. However, this thesis work mainly deals with the last area (motion control) primarily because this is where derivative information is required.

2.1.1.2 Inverse Dynamics

The robot control problem is the problem of computing the torques which are to be applied to the joints (the task of inverse dynamics) to drive the robot manipulator along a desired trajectory. In general, the desired motions will be subject to motion constraints or performance criteria, such as minimum overshoot or minimum time. The computation of the necessary torques requires consideration of the necessary parameters such as inertia and damping.

For the purpose of this thesis, we have chosen a single-link manipulator with joint flexibility. The manipulator is described in [1] as follows: "*The manipulator consists of a motor which is elastically coupled to a uniform thin bar of length l, mass m and moment of inertia (1/3) ml*^{2}". The complete equations of motion are given in [1]. Here we will just show the relationship between the joint variables and the torque generated for the manipulator in open loop:

$$\Gamma = (\frac{1}{A_1 B_2})\,(a_1 A_2 - a_3 A_1)\,\dot{\theta} + \;(a_1 - A_2)\,\ddot{\theta} - \mu\,(a_1 + a_4)\,q^{(3)} \;+ \mu q^{(4)} \;(A_2 - A_1)\sin(q) \;+ a_2 a_4 \mu \cos(q)\,\dot{q} \qquad (6)$$

We can see two things from equation (6). First, inspite of being able to derive a mathematical equation the formulation is very cumbersome. In some systems obtaining a formula in closed form is very difficult analytically and requires a symbolic software such as MAPLE. This is the basis of the work in [1], where a controller was devised to eliminate the knowledge of the mathematical formulation presented in equation (6). Second, we notice the use of higher-order derivatives of the position variable ($q^{(2)}$, $q^{(3)}$, $q^{(4)}$) where it was assumed that these parameters would be available by the use of appropriate instrumentation. For the purposes of this thesis, we will assume that we can only measure the position component and the acquisition of the higher-order derivative information will be made by the proposed methodology. This assumption is made because we want to use a minimum amount of hardware.

In the next section we will give a brief description of a system we can use to measure the position component.

2.2 Sensors

We saw from Section 2.1, that it will be necessary to measure the position variable and the corresponding higher-order derivative information. In this section we will show the techniques that can be used to measure the position variable. While it is true that sensors to measure velocity do exist [15], we assume that the generation of the velocity component (and also the acceleration and jerk components) will be made by the proposed methodology. We will first start by giving a brief overview of sensors and then describe an incremental optical encoder system. This is the type of system typically used to measure the position of the joints of a robot manipulator.

2.2.1 Overview of Sensors

The control structure of a robot involves knowledge of the position of each joint in order to calculate the position of the end-effector thus enabling successful completion of the programmed task. The movements of the joint can be angular or linear depending on the type of robot - Polar, Cartesian, Resolute, Cylindrical, Prismatic. Refer to [10] for a more in-depth explanation. An appropriate algorithm is then used to calculate the end-effector position in the Cartesian co-ordinates of the task space of the robot.

Research work is currently being done to involve the use of external sensory feedback (exteroceptors) as shown in Figure 6. Using this would allow one to forgo measuring the position of the joint measurements [15].

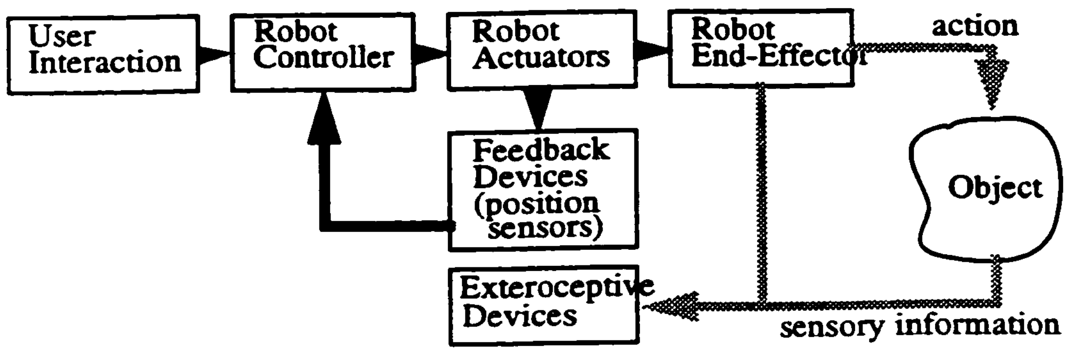


Figure 6 : Block diagram of future robot systems.

This technique is based on adaptive techniques, similar to the way in which the human body functions. However, this technique is largely in the research stages and has not found its way in commercial systems.

For the time being, internal position transducers remain the most reliable way for obtaining any information of the robot arm. There are two main types of position transducers: absolute and incremental.

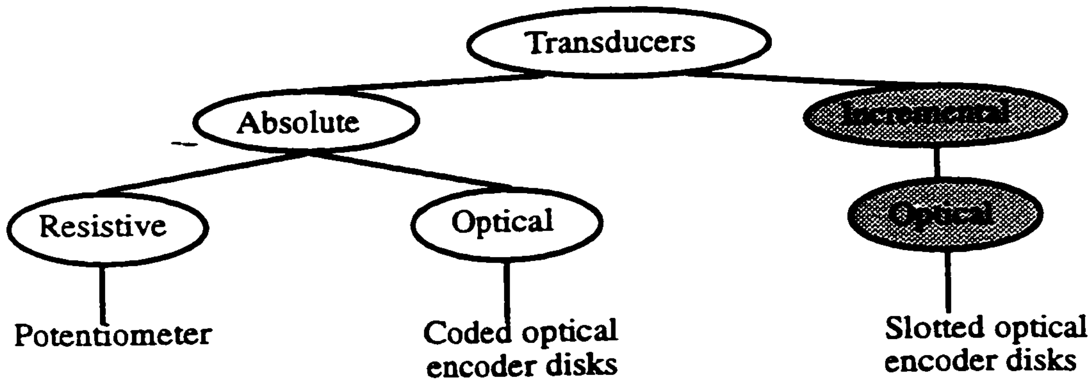


Figure 7 : Typical robot position transducers.

The output of an incremental encoder is a pulse signal that is generated when the disk rotates as a result of the motion of the robot. By counting the pulses, both angular displacement and angular velocity can be measured. Displacement is obtained with respect to a reference point on the disk as indicated by a reference pulse (index pulse) generated on that location on that disk.

An absolute encoder (also known as whole-word encoder) has many pulse tracks on its transducer disk. When the disk of an absolute encoder rotates, several pulse trains - equivalent to the number of tracks on the disk - are generated simultaneously. At a given instant, the magnitude of each pulse signal will have one of two states - i.e. a binary state, as determined by a level detector. This signal corresponds to a binary digit (a 0 or 1). Thus, the set of pulse trains gives an encoded binary number at any point in time. The pulse windows on the tracks can be organized into a specific pattern so that each of these encoded patterns correspond to the angular position of the disk at the time when the particular binary number is detected.

The same signal generation mechanism can be used in both types of encoders. There are four commonly used means to achieve signal generation:

- Optical sensor method
- Sliding contact method
- Magnetic saturation method
- Proximity sensor method

We will only describe the signal generation in the first method. The other three methods are adequately covered in [15].

The optical encoder uses an opaque disk that has a circular track (or as in the case of an absolute encoder, several tracks) with some arrangement of identical slots. A parallel beam of light (from an LED) is projected to all tracks from one side of the disk. The transmitted light is picked of from the other side of the track using photosensors. Each track has usually one photosensor as shown in Figure 8 (in subsequent sections we will show that there might be two photosensors per track to increase the resolution).

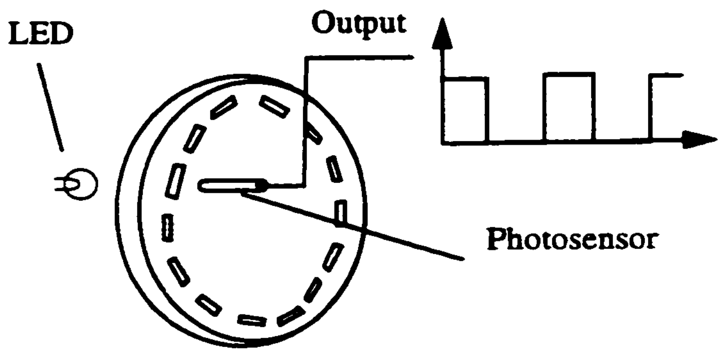


Figure 8 : Schematic representation of an optical encoder.

Since the light from the LED is interrupted by the opaque areas. the signal registered is a series of voltage pulses. This signal is then interpreted to get the angular position and angular velocity of the disk.

In the following sections we will describe in more detail an incremental optical encoder. We will also describe how it can interfaced with a digital processor to get the required measurements.

2.2.2 Incremental Optical Encoders

There are two possible configurations to for an incremental encoder disk: 1) the offset sensor configuration, 2) the offset track configuration. An example of the former case is shown in Figure 9.

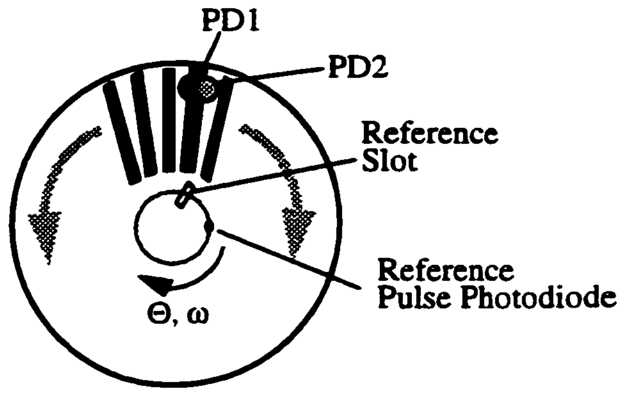


Figure 9 : Incremental Encoder Disk with Offset Sensor Configuration.

The disk has a circular track with equally spaced transparent slots. The area of the opaque region between the slots is equivalent to the area of the slots. Two photodiode sensors (PD1 and PD2) are located on the slot. They are offset from each other by half a slot length. The ideal waveforms of the output signals (v_1 and v_2) are shown in Figure 10.

In the second configuration, two identical tracks are used, one offset from the other by a quarter pitch. In this case, one photo-diode sensor is positioned facing each track without any circumferential offset. The output waveforms are the same as before.

In both variations of the incremental encoder, we see that there is a separate track containing a "Reference Slot" and a "Reference Pulse Photodiode". This track is used to generate a reference pulse per revolution of the disk. This pulse is used to initiate the counting operation (see Figure 10c). Additionally, the index pulse count gives the number of complete revolutions. Note that the pulse width and pulse-to-pulse period are constant in each sensor output when the disk rotates at constant angular velocity. When the disk accelerates, the pulse width decreases and when disk decelerates, the pulse width increases.

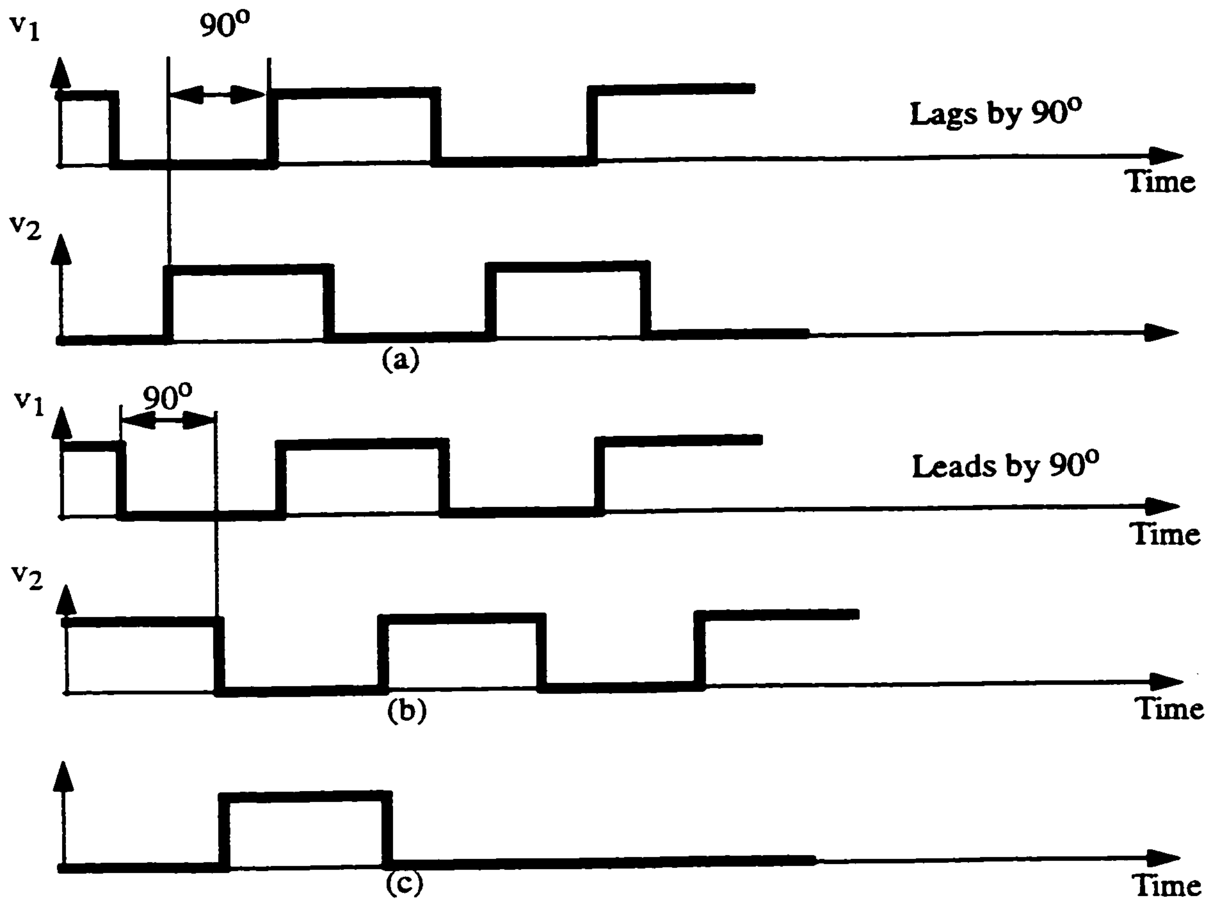


Figure 10 : Pulse signals from an incremental encoder: a) clockwise rotation, b) anti-clockwise rotation, c) reference pulse signal.

The hardware for generating the mechanism is shown in Figure 11.

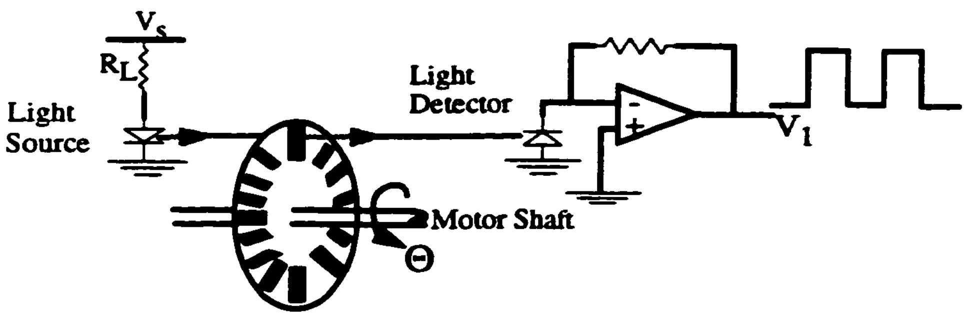


Figure 11 : Hardware for optical incremental encoder (for a single output pulse).

2.2.2.1 Direction of Rotation

The offset of the two photodiode sensors is used to determine the direction of rotation of the disk. From Figure 11a, it can be seen clearly that the disk rotates in a clockwise direction whereas in Figure 11b, the disk is rotating in an anti-clockwise direction. Thus the direction of rotation can be determined by the phase difference of the two signals using appropriate phase difference circuitry.

One method of determining the phase difference is to time the pulses. Let us assume, if the counting is initiated when the v_1 signal begins to rise and if n_1 is the number of clock cycles until v_2 begins to rise and n_2 is the number of clock cycles until v_1 begins to rise, then $n_1 > n_2 - n_1$ corresponds to a clockwise rotation and $n_1 < n_2 - n_1$ corresponds to an anti-clockwise rotation.

2.2.2.2 Position and Velocity Computation

To compute the angular position, Θ, suppose that the maximum possible count is *M* pulses and the range of the encoder is $\pm\Theta_{max}$. Then the angle corresponding to a count of *n* pulses is

$$\Theta = \frac{n}{M}\Theta_{max} \tag{7}$$

Assuming the data size is r bits, then the maximum number of pulses is

$$M = 2^{r-1} \tag{8}$$

where zero count is included.

Note that if Θ_{max} is 2π and Θ_{min} is 0, then Θ_{max} and Θ_{min} will correspond to the same position of the code disk. To avoid this ambiguity, the following is used

$$\Theta_{min} = \frac{\Theta_{max}}{2^{r-1}} \tag{9}$$

Two methods exist to compute the velocity: 1) the pulse-counting method and 2) the pulse-timing method. In the first method, the pulse count (n) over the sampling period (T) is measured. Consequently, the average time for one pulse is T/n. If there are N slots on a disk, the average time for one revolution is NT/n. Thus the angular velocity (ω) is

$$\omega = \frac{2\pi n}{NT} \tag{10}$$

In the second method, let us assume that the clock frequency is f Hz. If m cycles of the clock signal are counted during an encoder period (interval between two adjacent slots), the time for the encoder cycle is m/f. Assuming there are N windows, the average time for one revolution of the disk is Nm/f. Thus the angular velocity (ω) is

$$\omega = \frac{2\pi f}{Nm} \tag{11}$$

2.2.2.3 Advantages/Disadvantages of Incremental Encoders

An advantage of incremental encoders is that there is no wear and tear. Consequently, an incremental encoder lasts longer than most other types of sensors mentioned earlier. A single incremental encoder can serve both as a position sensor and as a velocity sensor. This eliminates the need for using a conventional analog speed sensor like a tachometer [15]. A further advantage of using an incremental encoder over a tachometer is the elimination of an A/D converter. The pulses generated are then directly counted (by an up/down counter) and timed thereby providing both position and velocity.

A disadvantage of the incremental encoder is the poor angular resolution $\Delta\alpha$. This is dependent on the number of transparent slots on the disk *n*, the width W_p of the photodiode active area which in turn depends on the photodiode mounting distance from the disk centre.

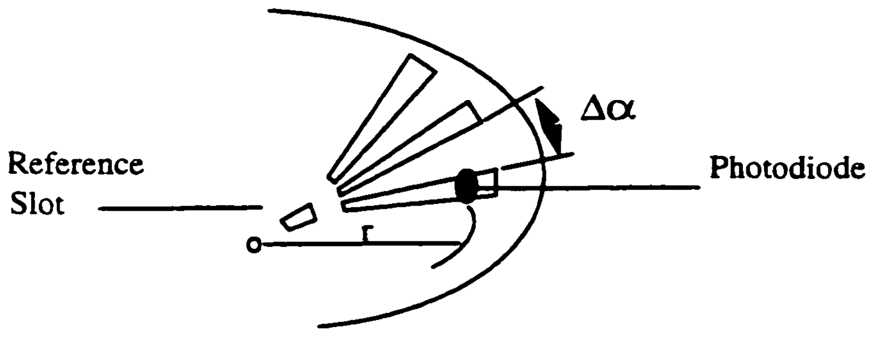


Figure 12 : Angular resolution of optical incremental encoder.

$\Delta\alpha = 360 / n$

$W_p = r \sin (360 / n)$

For example, to design an incremental position transducer with a resolution of $\pm 3^o$ one would require an encoder disk of a minimum of 120 slots. Typical commercial sensors have a resolution of 0.08^o over $\pi/2$.

2.2.2.4 Data Acquisition in Incremental Encoder

A method for interfacing an incremental encoder to a digital processor is shown in Figure 13.

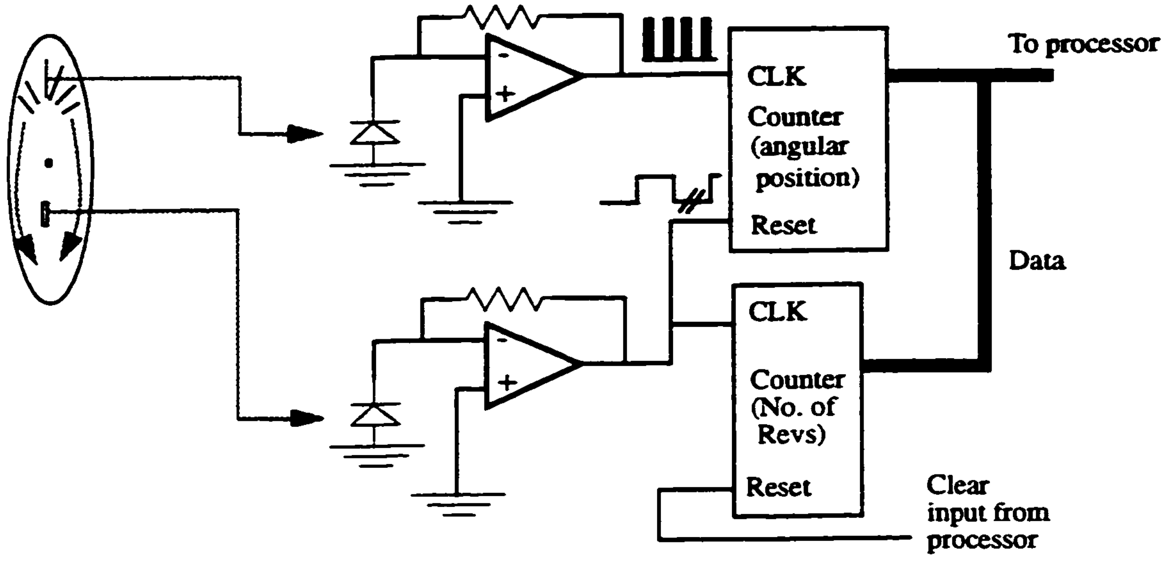


Figure 13 : Schematic of computer interface circuit for an incremental position encoder.

The optical encoder output is a pulse train. In order to measure the position, the interface hardware needs to count the pulses produced when moving from the last position. However, this latter position must be known at all times for the robot controller because this encoder can provide only an incremental-position measurement, that is it can only show how much the position has changed since the last move operation. To overcome this problem, the robot needs to be reset to a known position after the switch-on (usually referred to as the "home" position) and keep track of the joint absolute positions in the computer memory by updating it with the incremental position measurements after each move. In the event that there is a power surge, the robot needs to be reset to its home position. The main components for this circuit are thus an amplifier and a digital counter. However, the previous implementation assumes that direction is not being sensed and thus it is not known whether to add or subtract the incremental position from the previous one stored in memory. Additionally, the interface circuit requires a form of local memory so that the transducer output can hold it until the processor needs it, and a sequence detection circuit to determine the direction of rotation by comparison of the two transducer outputs). The modified circuit is shown in Figure 14:

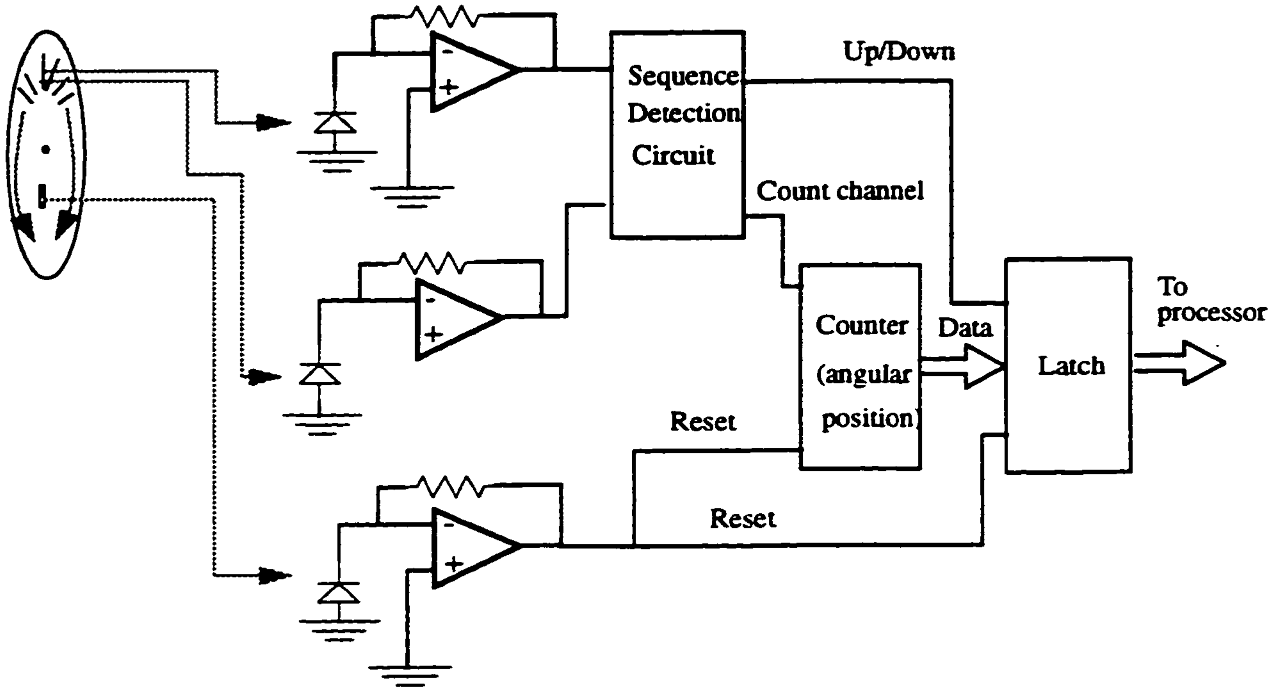


Figure 14 : Schematic of computer interface circuit to measure position with direction detection and higher resolution.

The arrangement provides an efficient means of data acquisition because the counting process can continue without interruption while the count is being read by the processor from the latch buffer.

2.3 Digital Signal Processing

In the context of this thesis, we concentrate on one particular aspect of digital signal processing. We study the task of generating first and higher-order derivatives from a measured variable. Higher-order differentiators yield samples of a bandlimited continuous time signal. In this work, we study higher-order differentiators that have been designed as non-recursive linear-phase filters that approximate the ideal frequency characteristic which varies as a power of frequency with frequency. Below, we will first give a mathematical formulation of the conditions imposed on a non-recursive filter which is to be used as a differentiator. We will then give a brief summary of the research work done in this area.

2.3.1 Higher-order Differentiators

Consider a nonrecursive digital filter with N taps having an impulse response *h(n) (n=0 to N-1)*. For the case of a linear phase filter having a symmetric impulse response we have *h(n) = h(N-1-n)*. Consequently the frequency response is $H(e^{jw}) = M(w)e^{-jw(N-1)/2}$, where

$$M(w) = \left(\begin{array}{ll} \left(\sum_{n=0}^{\frac{(N-1)}{2}} a(n)\cos(nw) \right) & N\ odd \\ \left(\sum_{n=1}^{(\frac{N}{2})} a(n)\cos\left(n-\frac{1}{2}\right)w \right) & N\ even \end{array} \right) \quad (12)$$

If N is odd,

$a(0) = h[(N-1)/2]$ and $a(n) = 2h[(N-1)/2-n]$ for $1 \leq n \leq (N-1)/2$.

If N is even,

$a(n) = 2h(N/2 - n)$ for $1 \leq n \leq N/2$.

For the case of an antisymmetric response, we have $h(n) = -h(N-1-n)$. Therefore, $H(e^{jw}) = M(w)e^{j(p/2 - w(N-1)/2)}$ where

$$M(w) = \left(\begin{array}{ll} \left(\sum_{n=1}^{\frac{(N-1)}{2}} b(n) \sin(nw) \right) & N\ odd \\ \left(\sum_{n=1}^{(\frac{N}{2})} b(n) \sin(n - \frac{1}{2}) w \right) & N\ even \end{array} \right) \tag{13}$$

If N is odd,

$b(n) = 2h[(N-1)/2 - n]$ for $1 \leq n \leq (N-1)/2$.

If N is even,

$b(n) = 2h(N/2-n)$ for $1 \leq n \leq N/2$.

An ideal k-th order differentiator has a frequency response $H_I(e^{jw}) = D(\omega)e^{jkp/2}$ where $D(\omega) = (\omega/2\pi)^k$ for $0 \leq \omega \leq \omega_p \leq \pi$. The upper passband edge frequency is ω_p. For an even order differentiator (k is even), $h_I e^{(jw)}$ is a real valued function. Thus only a non-recursive filter with a symmetrical impulse response can be used for the design of even order differentiators. A full-band differentiator ($\omega_p = \pi$) can be designed only when N is odd. When N is even, it is necessary that $\omega_p < \pi$. Thus when $H_I(e^{jw})$ is purely imaginary (i.e. when k is odd), it is necessary to design a non-recursive filter with an anti-symmetrical impulse response. For this case, a full-band differentiator can be designed only when N is even.

2.3.1.1 Review of previous research in higher-order differentiators

Digital differentiators can be classified into two categories: non-recursive and recursive. In non-recursive filters, the design approaches used are extensions of those used for first-order differentiators [16, 17, 18]. In [16], the minimax method is based on the Remez exchange algorithm which has been extended to incorporate the parameters involved in the design of the higher-order differentiators. The eigenfilter method has been extended to the design of higher-order differentiators in [17], by formulating an error function in quadratic form. The error function involves the square of the difference between the desired amplitude response and amplitude response of the designed filter. In this method, the desired amplitude response is equal to the amplitude response of the designed filter at an arbitrary reference frequency, as opposed to the being equal to the ideal amplitude characteristic. The filter coefficients are found by computing the eigenvector corresponding to the smallest eigenvalue of a positive-definite symmetric matrix. In [18], the least-squares approach is extended to the design of higher-order differentiators. The procedure involves formulating the absolute mean-square error between the practical and ideal differentiator as a quadratic function. The coefficients of the differentiators are obtained by solving a system of linear equations. The motivation of this approach is three-fold: a) to achieve a direct route that explicitly considers the ideal amplitude response in the design procedure, b) offer a closed form solution for the filter coefficients and c) devise a non-iterative method to obtain this solution with a low computational complexity.

Recursive digital differentiators have been designed using non-linear and linear programming techniques [19, 20]. In [19], the coefficients for the recursive differentiator have been optimally chosen to minimize a square-error criterion based on the magnitude of the frequency response. In [20], a linear programming technique has been used to simultaneously approximate magnitude and group-delay characteristics.

As we can see, a lot of work has been done in this area. The techniques mentioned above are just a small sample of the work that has been done in designing filters for differentiators. Our work also involves building a filter for the purpose of generating first and higher-order derivatives. We have adopted the strategy of using neural networks. We will further elaborate on the specifics of this strategy in Chapter 3.

2.3.2 Effect of quantization noise on a signal

The variables $q \ldots q^{(n-1)}$ from Figure 4 represent the variables of the manipulator needed to control the flexible arm. The "*n*" represents the order of the plant (which in this case is four). The variables are respectively position, velocity, acceleration and jerk (rate of change of acceleration). The position component can be easily measured using any commercial sensor. As we have indicated in Section 2.2, the sensors measure the joint position to a certain precision. The resulting measurement of the position signal is thus quantized. The effect of quantization on a signal introduces noise into the resulting signal [21]. This can be seen in Figure 15.

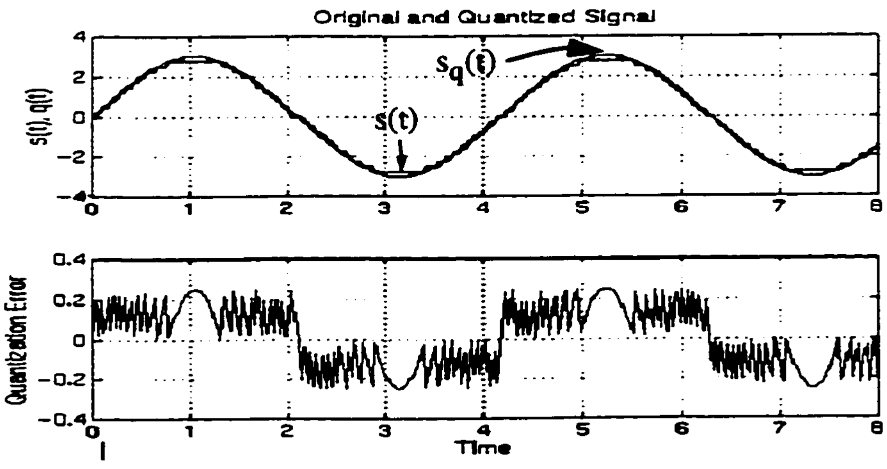


Figure 15 : Noise introduced by quantization.

As we saw in Section 2.1, the most common technique to perform differentiation is by the use of finite differences. Approximations of derivatives are obtained by repeated application of equation (1):

$$\frac{d^k}{dt^k}y(t)\Big|_{t=nT} = \frac{d}{dt}\left(\frac{d^{k-1}}{dt^{k-1}}y(t)\right) \qquad (14)$$

However, as we have mentioned earlier, our measured position signal is quantized to a specific resolution. If we apply successive differentiation to the above quantized position signal, then we get the results shown in Figure 16, Figure 17 and Figure 18.

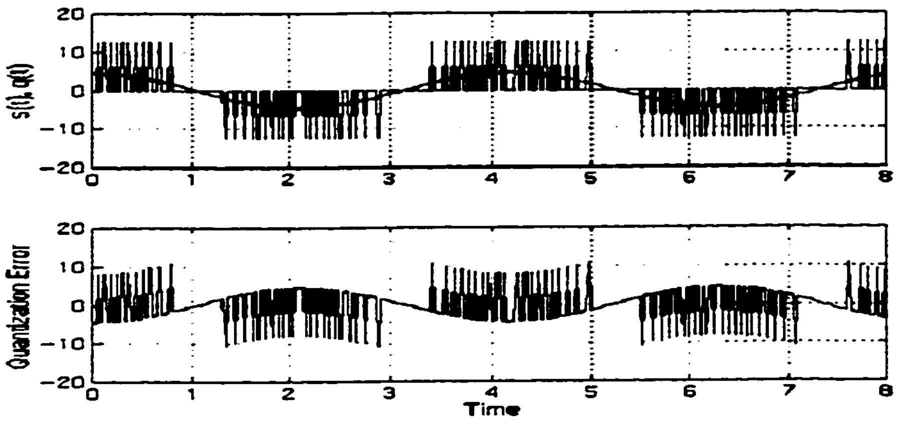


Figure 16 : Effect of differentiating quantized position signal to get velocity signal.

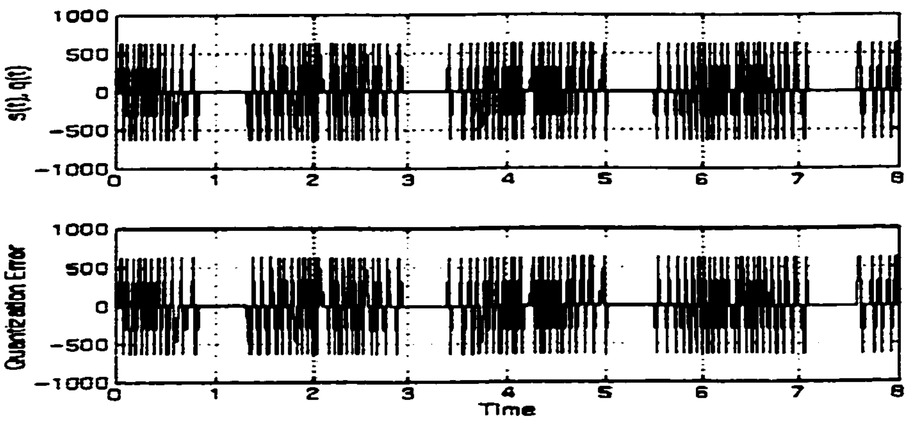


Figure 17 : Effect of quantized differentiation signal to get acceleration signal.

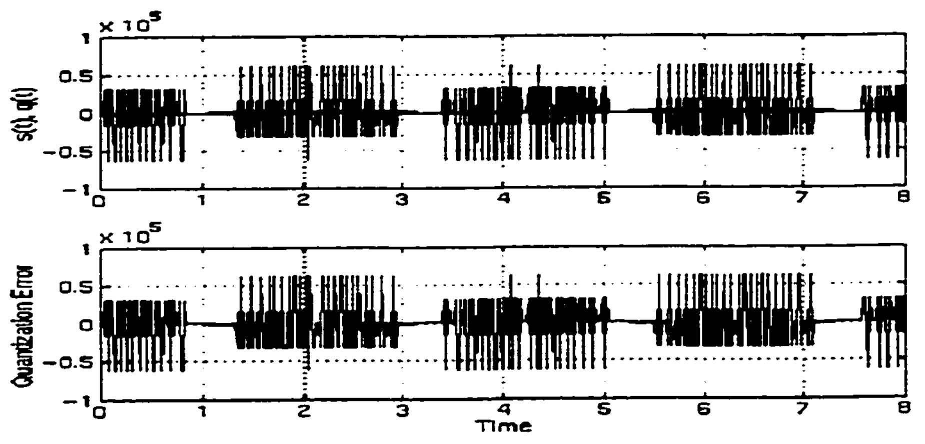


Figure 18 : Effect of differentiating quantized acceleration signal to get jerk signal.

As we see from these figures, successive differentiation of a quantized signal only adds more noise to the resulting waveform. The resulting signal is severely distorted. By the time the jerk component is determined, the signal is almost all noise. Most of the pertinent information is lost. We will show in the following chapter that with such information, the control system is unable to track the desired trajectory. Some means must exist to filter out this quantization noise. The most common approach is to use a Finite Impulse Response (FIR) or an Infinite Impulse Response (IIR) [19] filter.

2.3.3 Summary

We started this chapter by giving a brief overview of TDNNs. We explained why we will use TDNNs and not conventional neural networks. The ability of TDNNs to represent relationships between events in time can only be done through the use of TDNNs. We then proceeded to examine the role of neural networks in robot control. We are mainly concerned with how a neural network will be able to compute the inverse dynamics of a flexible-joint manipulator. We described the model of the manipulator that we will use in our investigation. We saw the complexity of the mathematical description of the robot. For the generation of the torque signal, we need to compute several control variables - namely the position variable and its derivatives; for a rigid manipulator this requires the first and second derivatives and for a flexible joint manipulator, this requires upto the fourth derivative information.

In Section 2.2, we presented various types of sensors that are available on the market. For the purposes of this thesis, we require only the use of a sensor to measure the position of the robot arm. We recommend the use of an incremental sensor as outlined in Section 2.2.2. We assume that sensors to measure the first and higher-order derivative (velocity, acceleration, jerk) are not available.

In Section 2.3, we gave a brief mathematical overview of the constraints imposed on a system that has the task of performing differentiation. Our conclusion was that for an FIR filter to perform full-band differentiation, it must have an even number of taps.

We then proceeded to review some research work that has been done in the area of digital filtering. We concluded that all of the previous works mentioned required *a priori* knowledge of the system. This does not satisfy our initial specifications as this is one of the things we want to avoid.

We concluded this section, by showing the effects of differentiating a quantized signal. Our conclusion was that filtering of the quantization noise is mandatory if we are to derive the information necessary for the control of our manipulator system.

In the next chapter, we will show how we used the concepts outlined in this chapter and applied them to achieve the main goal of the thesis.

Chapter 3
Proposed Methodology for Estimating Derivative Information

In this chapter, we propose a generalized methodology for determining first and higher-order derivatives of quantized measurements obtained using only position sensors. Our goal is to achieve this objective without use of specialized high-precision sensors, and at the same time to filter the noise arising from quantization. We accomplish this using neural networks and a type of neural networks known as Tap Delay Neural Networks (TDNN). In Chapter 4, we show the simulation results of our proposed methodology.

3.1 Investigation of problems associated with generating higher-order derivatives

We have mentioned in Chapter 2 that many control applications require the generation of position and velocity components and possibly higher-order derivatives for control of a mechanical system. Indeed, for the robot control problem considered in this thesis, we will need to generate acceleration and jerk components as well.

While sensors are abundant for measuring position, one must nevertheless take into account a process to filter the quantization noise which arises as a result of the finite resolution of the sensor. We show in Section 3.2 how a sensor with an 8-bit or a 12-bit resolution can severely distort the performance of a control system which is able to track the desired trajectory perfectly when no such sensors are used. Consequently, a means must exist to eliminate or reduce quantization noise.

Sensors to measure velocity and acceleration, while not abundant, are still available on the market. However, using sensors implies the use of additional hardware - a factor we are trying to minimize. Even if sensors were utilized to measure velocity and acceleration, extensive filtering techniques would still be needed to eliminate the quantization noise. For the purpose of this research work, we will need to measure a component called 'jerk' (rate of change of acceleration). Sensors to measure this component are non-existent. Previous approaches that have required the use of higher-order derivative information for the control of a system have generated these parameters assuming a perfect knowledge of the position measurements and knowledge of the system dynamics. We would like to develop an approach which will eliminate the dependence on such knowledge.

3.1.1 Control system for a single-joint manipulator

The control system which we use to test our hypothesis was shown in Chapter 1 and is shown here in Figure 20. In this research work [1], a neural network was used as a controller for a flexible-joint manipulator[1]. As we can see, the position (q), velocity ($\dot{q}$), acceleration ($\ddot{q}$) and jerk ($q^{(3)}$) are needed for feedback to control the robot manipulator. We will develop an approach which will enable us to generate these feedback variables.

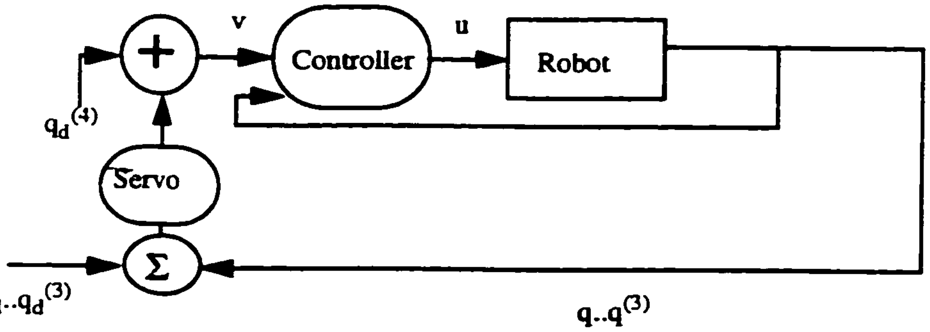


Figure 19 : The control system for an arbitrary n-th order, single-joint robot.

[1]For complex systems (like a flexible-joint manipulator) accurate knowledge of complete dynamics may not be available and neural networks can be used as controllers.

We can see that a partitioned control strategy was employed for the control system. This strategy fulfills the criterion that had originally been specified - the controller design must be attainable without the availability of the robot's dynamical equation and the overall system response must conform precisely to some independently-provided performance specification.

3.1.2 Closed-loop system response

We now examine the performance of the closed-loop system. We first show the results that were obtained as a result of the research done in [1]. This implies that we will not take into account any hardware considerations (namely the bit resolution of the sensor). We will next show the performance of the system when we incorporate the effect of quantization into the overall control system.

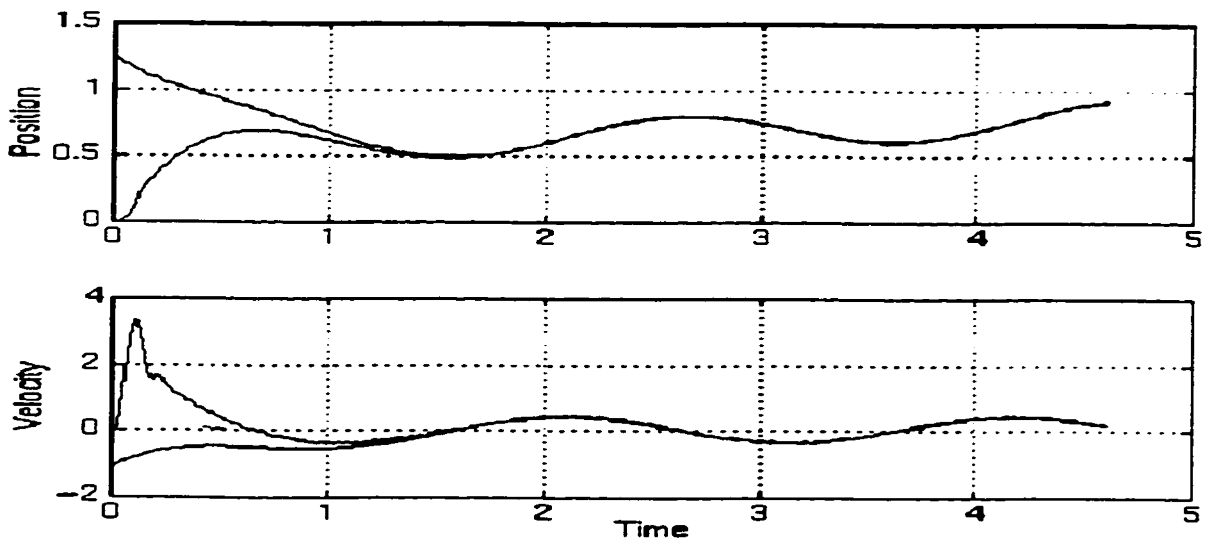


Figure 20 : Tracking performance of control system without any hardware considerations.

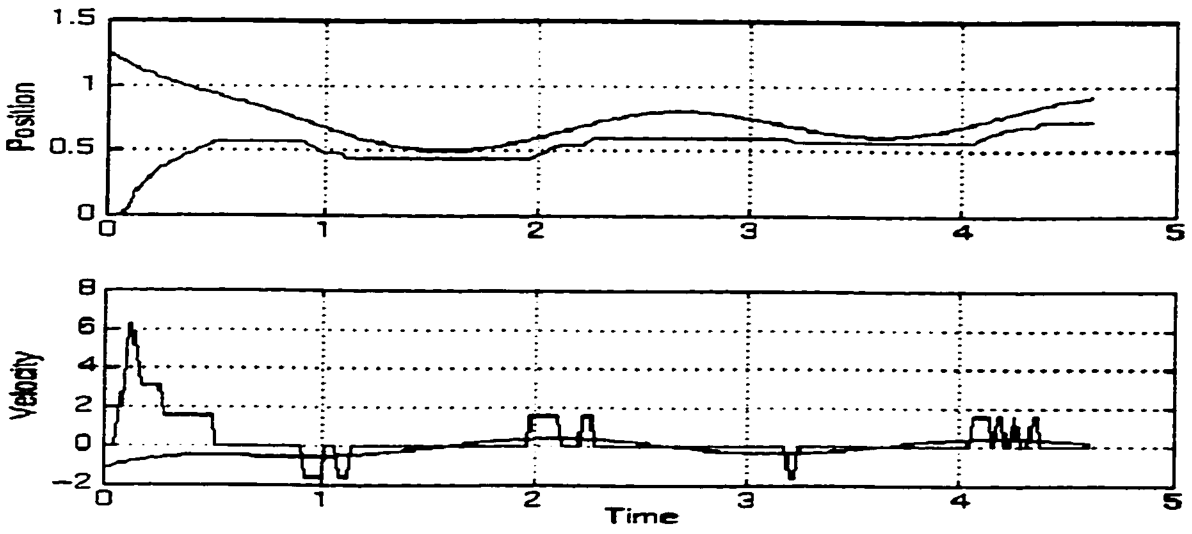


Figure 21 : Tracking performance of control system when using a 12-bit sensor.

From Figure 21, we see that with a 12-bit quantization level, the tracking of the desired trajectory has been severely degraded. However, in the interests of reducing the cost of the overall sensor system, we experiment using a lower quantization level. We show the results achieved using an 8-bit quantization level. This is illustrated in Figure 22.

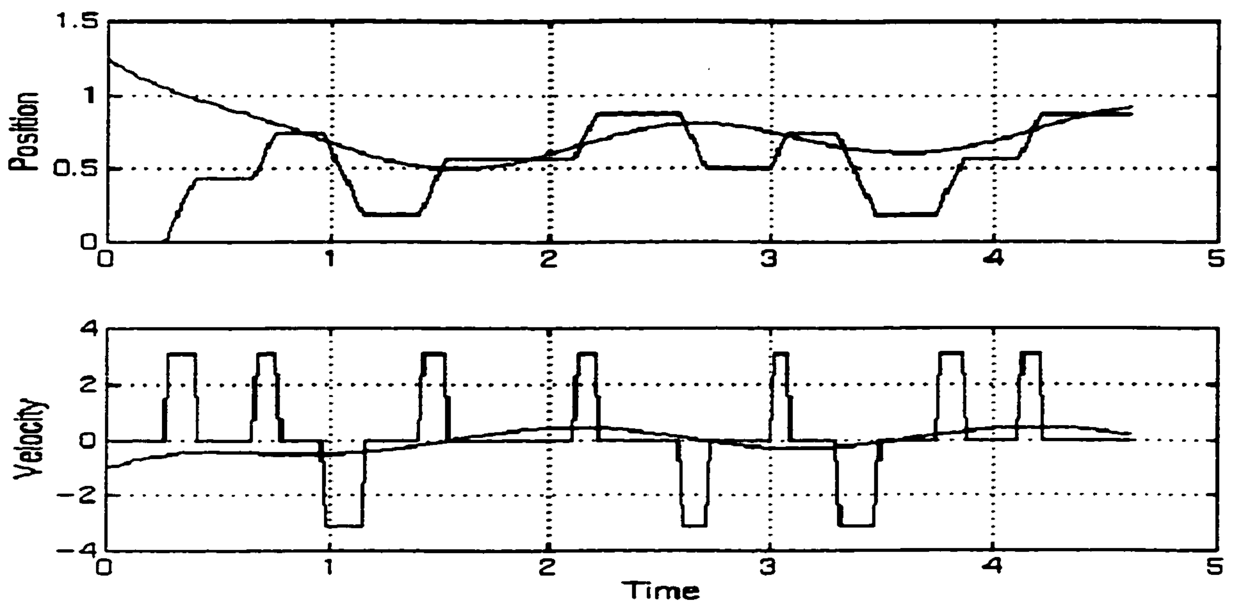


Figure 22 : Tracking performance of control system when using an 8-bit sensor.

From the above results, we can see the effects of quantization on the tracking of a desired trajectory. At an 8-bit quantization level, the tracking has been completely degraded and is not at all within acceptable limits. Typical resolution of commercial incremental sensors is usually within the range of 8-12 bits. It will be our objective to resolve the effect of this inherent quantization problem when a sensor is used to measure the position variable.

3.2 Proposed methodology to generate derivatives

Due to the finite resolution of the encoder, quantization noise is introduced in the position signal. We have seen the effect that a quantized position signal and its derivatives have in tracking a given trajectory (Figure 21 and Figure 22). The proposed methodology should ensure that the two problems mentioned earlier (that of filtering the quantization noise and that of generating higher-order derivative information) be taken care of. Solutions for filtering the quantization noise include using an IIR or FIR filter. For the problem of generating higher-order derivatives, a myriad of parameter estimation methods exist [16, 17, 18]. However, the problem with most of these methods is their dependence on the formulation of a mathematical model of the system - the very thing we are trying to avoid. To solve these two problems, we propose the following modification to the control system:

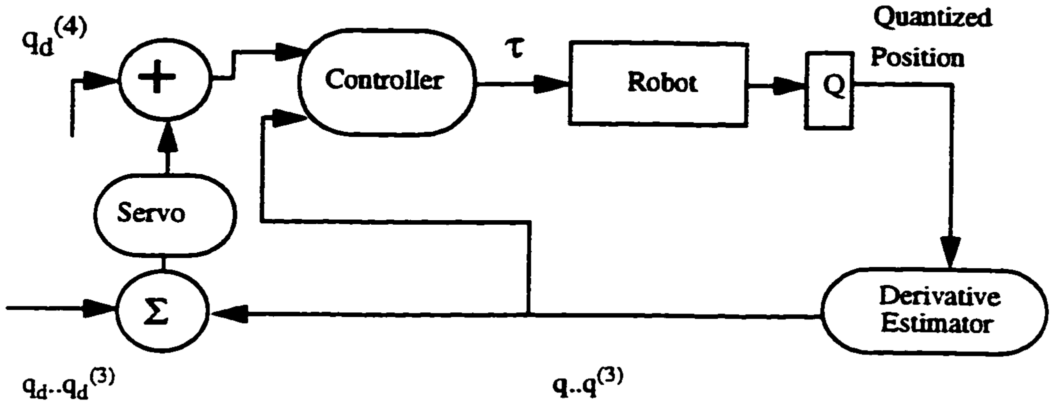


Figure 23 : Modified diagram of the control scheme.

In Figure 23 we have a block 'Q' which models the bit resolution of a sensor. This parameter can be adjusted by the user. The quantized position signal will then be transmitted into another block called the '**Derivative Estimator**'. The purpose of this neural network computational machine will be to generate the derivative information. In the following section, we will explain the structure of the derivative estimator.

3.2.1 Derivative estimator architecture

The purpose of the block NN_p in Figure 24 is to filter the quantized position signal. The filtered position signal will then be used in the servo computations to achieve the appropriate control. The training sequence used is of paramount importance in any neural network system. Additionally, using a neural network can filter other non-linearities that can arise in a dynamical system.

Our method uses a ***quantized*** position input with frequencies in the range from 1-5 Hz[2] over a range from 0 - $\pi/2$ degrees of joint motion. We have added noise to this noise to model the non-linear effects of a dynamical system. The output of the neural network is a smoothed version of the signal. The position signal used in our training sequence was $\sum_{i=0}^{5} A_i \sin(\omega_i t)$ where $w_i = i$.

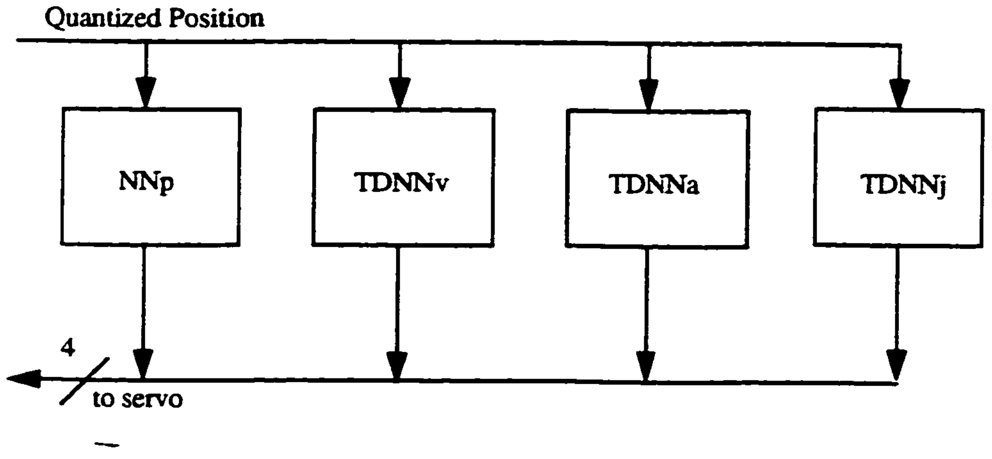


Figure 24 : Proposed Derivative Estimator Architecture.

The next step is to generate the required derivatives from the position signal. For this, we have used TDNNs. This will be described in the following section.

[2]We assume that the robot will be operating in this range. These can be changed if desired.

3.2.2 Tap delay neural network architecture

The purpose of the blocks $TDNN_v$, $TDNN_a$, $TDNN_j$ in Figure 24 is to generate from ***quantized*** robot position measurements the velocity, acceleration and jerk components respectively that are required for the controller. The TDNN structure we used is shown in Figure 25:

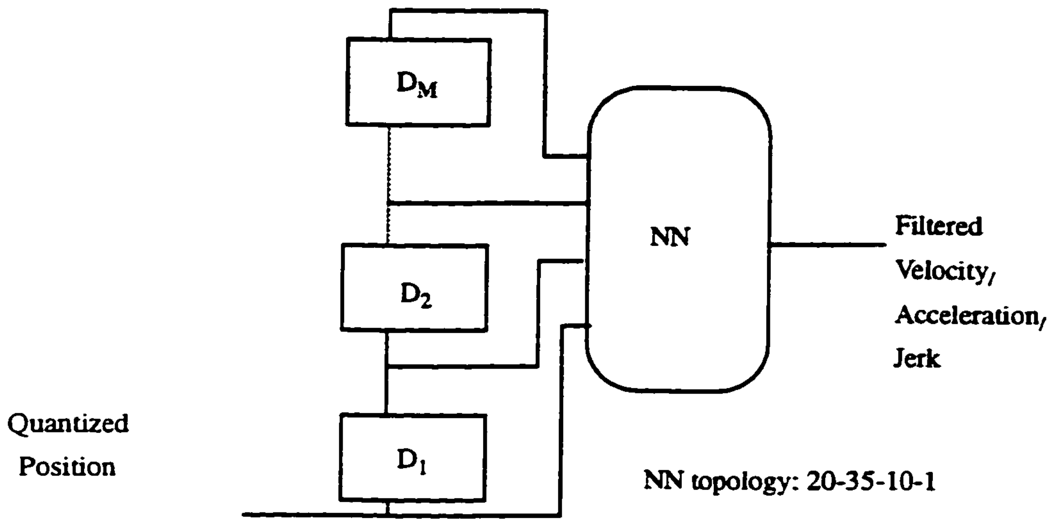


Figure 25 : TDNN architecture.

The training sequence used for generating the velocity, acceleration and jerk information is as follows: The position signal is the same as the one used previously - i.e. ***quantized*** position input with frequencies in the range from 1-5 Hz over a range from 0 - $\pi/2$ degrees of joint motion with some noise to model the non-linearities in a dynamical system. The velocity, acceleration and jerk signals used for training are of the following form respectively:

$$\text{Velocity} = \sum_{i=0}^{5} A_i \omega_i \cos(\omega_i t)$$

$$\text{Acceleration} = \left(\sum_{i=0}^{5} -A_i w_i^2 \sin(w_i t)\right)$$

$$\text{Jerk} = \left(\sum_{i=0}^{5} -A_i w_i^3 \cos(w_i t)\right)$$

3.3 Summary

At the start of the thesis, we indicated the importance of having a means to determine the feedback signals when developing a control strategy. We reviewed case studies which indicated how well a robot manipulator can be controlled under different conditions. These works relied on a mathematical foundation for the generation of the control parameters. They made the assumption that appropriate instrumentation is available for determining the feedback signals. We have taken the research of [1] where they developed a neural network control strategy for a flexible-joint manipulator and showed that when hardware considerations are taken into account the control algorithm does not perform as required.

We proceeded to propose a methodology that enables one to determine the control parameters. This methodology is dependent on the use of neural networks. This satisfies the three constraints mentioned in Chapter 1. First, we do not require any *a priori* knowledge of the robot's dynamics. Second since we are using only a sensor to measure the position component, we eliminate the use of additional hardware, and third, we achieve real time performance.

In Chapter 4, we will present the training procedure and the simulation results of the neural networks used in the derivative estimator. We also show the simulation results when the derivative estimators are used as part of the closed-loop control system.

Chapter 4
Closed-loop Behaviour with Derivative Estimator

In this chapter we explain the training procedure for the neural networks in the derivative estimator architecture presented in Chapter 3. We then show simulation results which indicate how well the neural networks performed their task as differentiators.

The neural networks were implemented using a commercial software neural network implementation environment (MATLAB 4.1) and run on a SUN SPARCstation 10.

We trained our neural networks using quantized robot position measurements. Our position samples were quantized to 8 bits and 12 bits. We trained four networks - one for eliminating the quantization noise from the measured samples of the position variable and three for the generation of the required derivative information - velocity, acceleration, jerk.

The neural networks were trained off-line. The data sampling rate was 500 samples per second. After the training was finished, the networks were tested to see how well they trained. Oversampling (sampling more information that the network was trained with per unit time) and undersampling (sampling less information that the network was trained with per unit time) were also attempted.

We term our neural network used for filtering the quantization noise a '**smoother**'. This is because it eliminates/reduces the noise arising from quantization and thus smooths the input signals. The other neural networks are given the term '**differentiators**'. This is because they perform differentiation of the input signal.

4.1 Neural network smoother

The neural network used for this application was a layered feedforward network consisting of one input and one output. The processing units were arranged in three layers: one input layer, one hidden layer and one output layer. The layers were completely connected. The hidden layer consisted of fifteen processing units. Each processing unit processes the quantized input signal by a weighted sum with a bias and this is then processed by a sigmoid activation function. Different network configurations were attempted. But we selected one which utilized as few neurons as possible.

The input signal to the network was chosen by considering the fact that the robot will move in an arbitrary joint space trajectory (subject to actuator constraints) that can be presented to it. The most general type of training signal would be white noise. However, since the expected system can only function in low frequencies, we employed a summation of arbitrarily chosen sinusoids given by

$$pos = 0.5\sin(t) + 2\sin(2t) + 1.5\sin(3t) + 3\sin(4t) + 0.2\sin(1.5t) + 5\sin(5t) \qquad (15)$$

The input signal in equation (15) is a representative of the input space of operating frequencies that the robot will operate in. The input signal is quantized to a specific resolution - either 8 bits or 12 bits. This represents the signal as measured by a sensor system. By quantizing the input signal, we add noise to the resulting signal. Consequently, using a simulated noise signal generates a uniformly populated input space during learning. As a result, during network recall, any input vector presented to it will be relatively close to some of the vectors used during training; the distances over which the net has to interpolate will remain small. The noise signal also results in an unordered presentation of the inputs during learning. This is a desirable condition for convergence to a global minimum when using, as we do, the generalized delta training rule.

4.1.1 Results from the neural network smoother

In Figure 26 and Figure 27 we see the result of using the neural network smoother. We show results when the position signal is quantized to 8 bits and 12 bits respectively. We have divided each figure into two parts: In the top half, we show the quantized position sample that was used for training (PAM signal) and the continuous signal. In the bottom half, we show the recall ability of the network. Also, in the second half, we superimpose a continuous version of the signal as an indication of how well the network was trained.

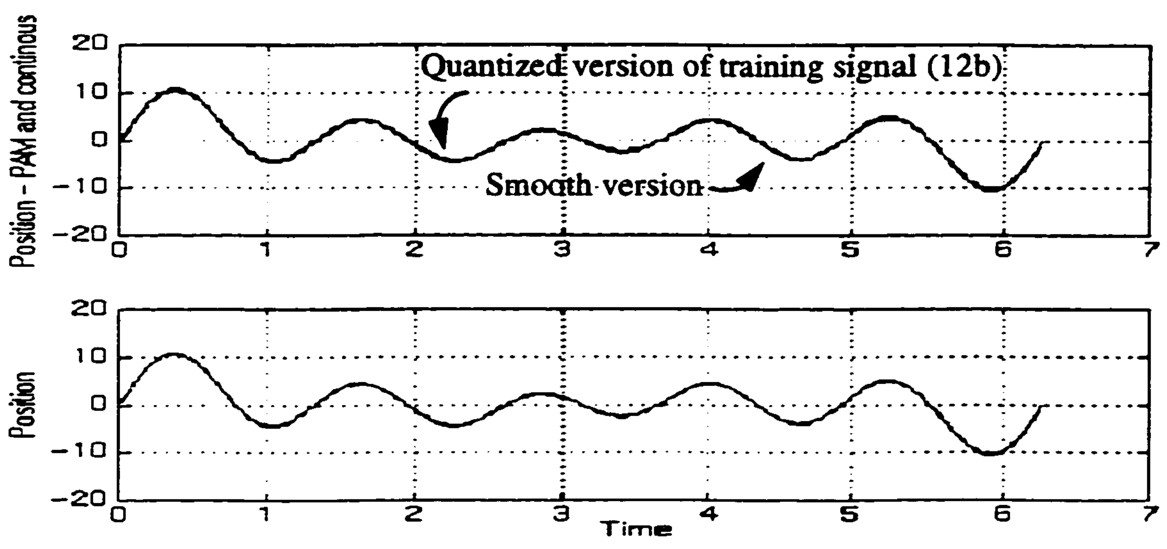


Figure 26 : Result of the NN smoother when the position signal is quantized to 12 bits.

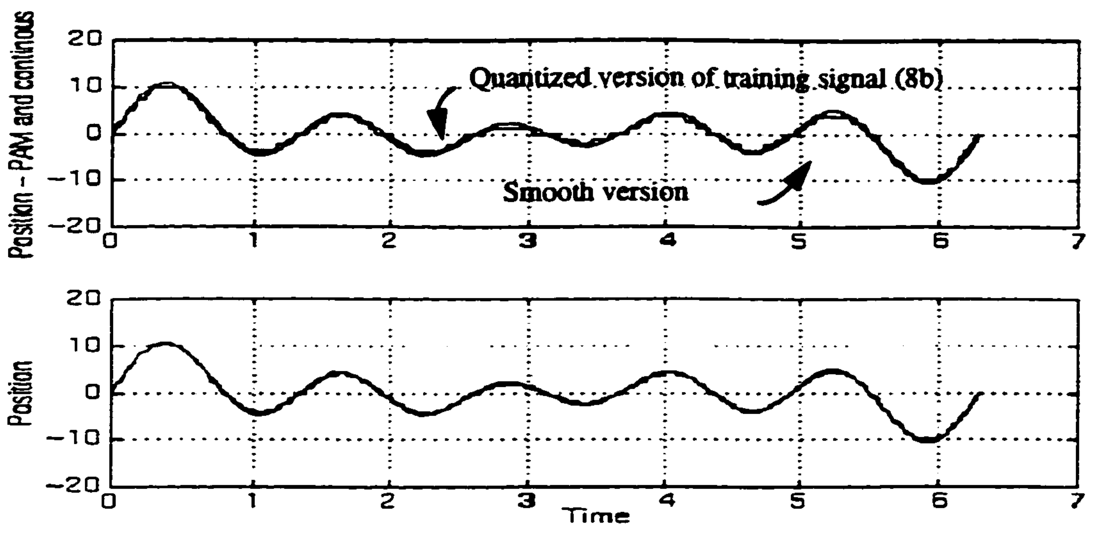


Figure 27 : Result of the NN smoother when the position signal is quantized to 8 bits.

We see from Figure 26 and Figure 27, the ability of the neural network to filter quantization noise. In the second-half of each plot, we see that we cannot distinguish between the signal that has been produced as a result of the neural network and the way it is in its non-quantized form. It is this filtered version of the position signal that we use as part of the servo computations (see equation (2)).

4.2 Neural network differentiators

The neural network used for each derivative (velocity, acceleration and jerk) was a layered feedforward network consisting of twenty inputs and one output. The processing units were arranged in four layers: one input layer, two hidden layers and one output layer. The layers were completely connected.

The input layer consists of delayed samples of the quantized robot position measurements. The total number of delays was determined by a process of trial and error and was finally set to twenty. In Section 2.3.1, it was mentioned that for an FIR filter to achieve full-band differentiation an even number of taps is required. We have tried different numbers of taps. But we could only achieve perfect training and recall when the number of taps was even. A higher number of even taps (greater than 20) was also tried. In those architectures, accurate training and recall was also achieved. However, in the interests of reducing the cost of the implementation of the neural network algorithm on a commercial DSP, the least number of taps which could successfully perform the task of differentiation was used.

The first hidden layer consisted of thirty-five processing units and the second hidden layer consisted of ten processing units. Again this was determined by a process of trial and error. Each processing unit processed the input signal by a weighted sum with a bias and this was then processed by a sigmoidal activation function. The inputs to the network were the same as those used for the smoother, i.e., they were quantized versions of the signal in equation (15). They represent signals that are measured by a sensor.

In the next section we will show the simulation results to indicate how well the neural network differentiators performed their task of differentiating

4.2.1 Results of neural network differentiators

We now show simulation results of the neural network differentiators. We present results for both 8 and 12 bits of quantization. In the top-half of each plot, we present the position signal that was used for training the neural network. In the bottom half of each plot we present the output of the neural network. Superimposed on this is the desired version of the signal.

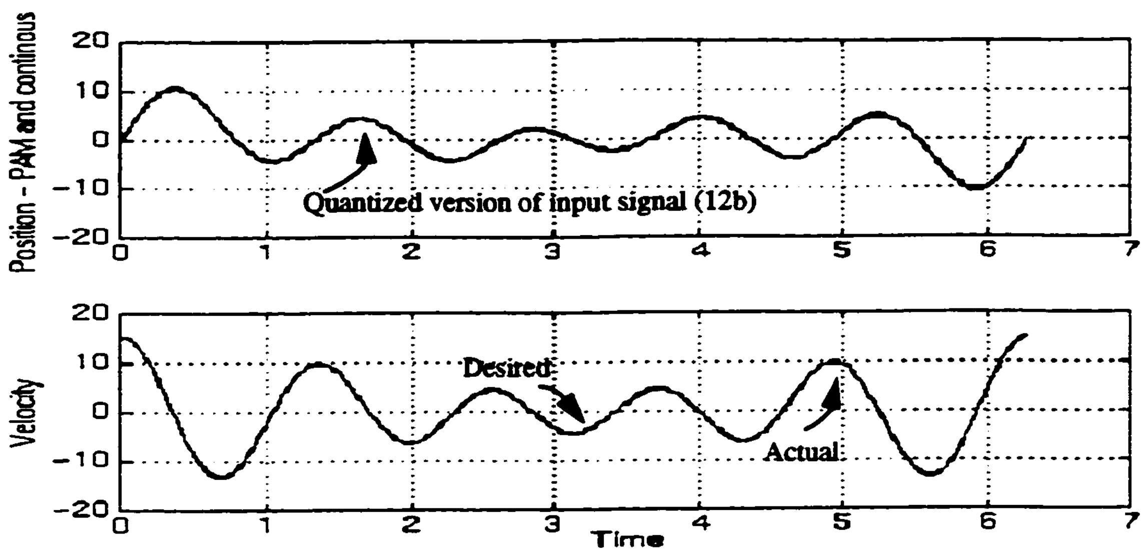


Figure 28 : Result of the neural network differentiator to generate first-order derivative when the input sample is quantized to 12 bits.

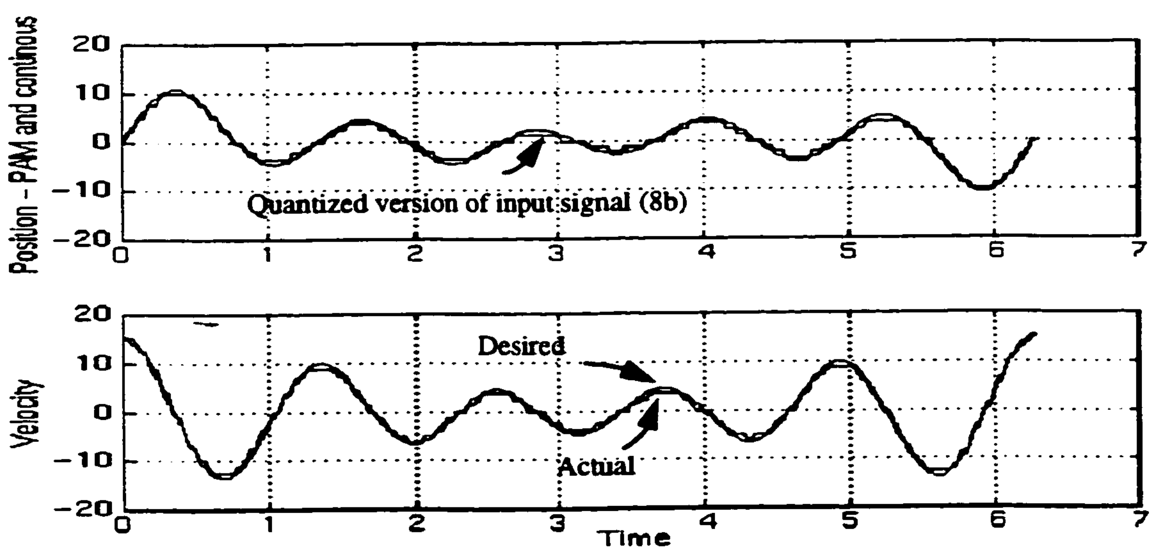


Figure 29 : Result of the neural network differentiator to generate first-order derivative when the input sample is quantized to 8 bits.

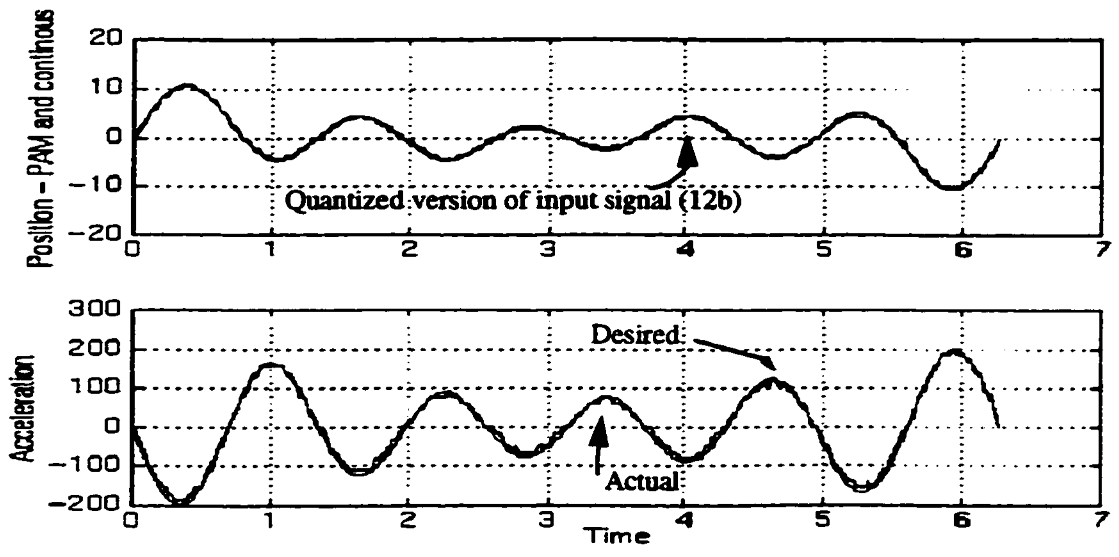


Figure 30 : Result of the neural network differentiator to generate second-order derivative when the input sample is quantized to 12 bits.

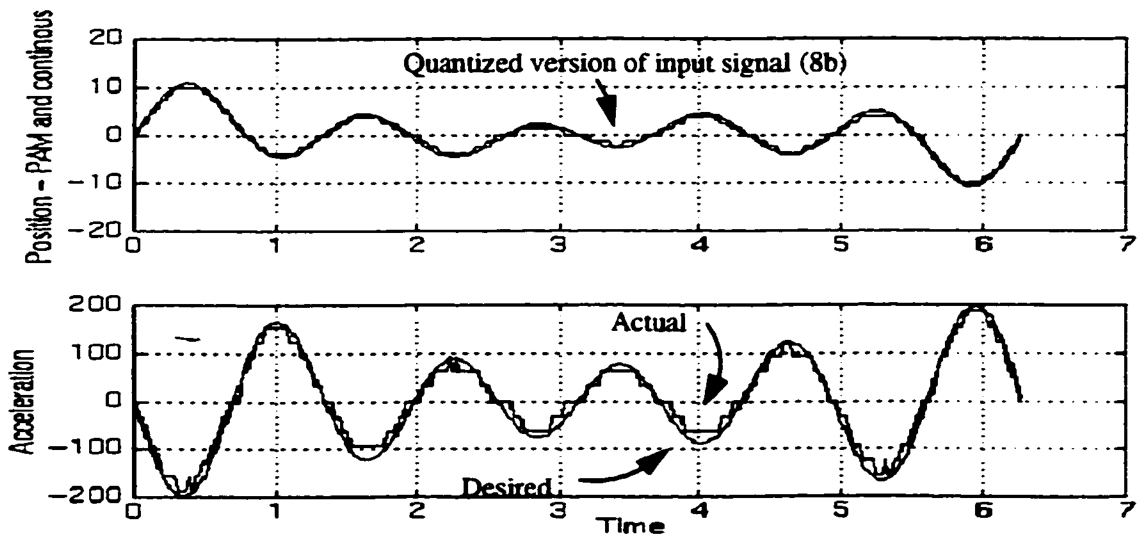


Figure 31 : Result of the neural network differentiator to generate second-order derivative when the input sample is quantized to 8 bits.

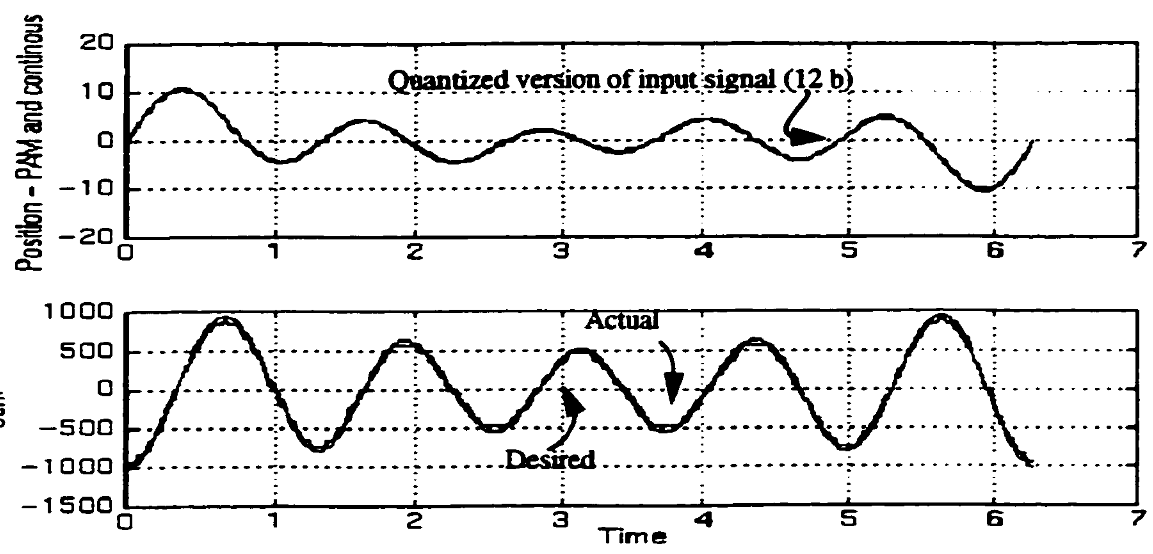


Figure 32 : Result of the neural network differentiator to generate third-order derivative when the input sample is quantized to 12 bits.

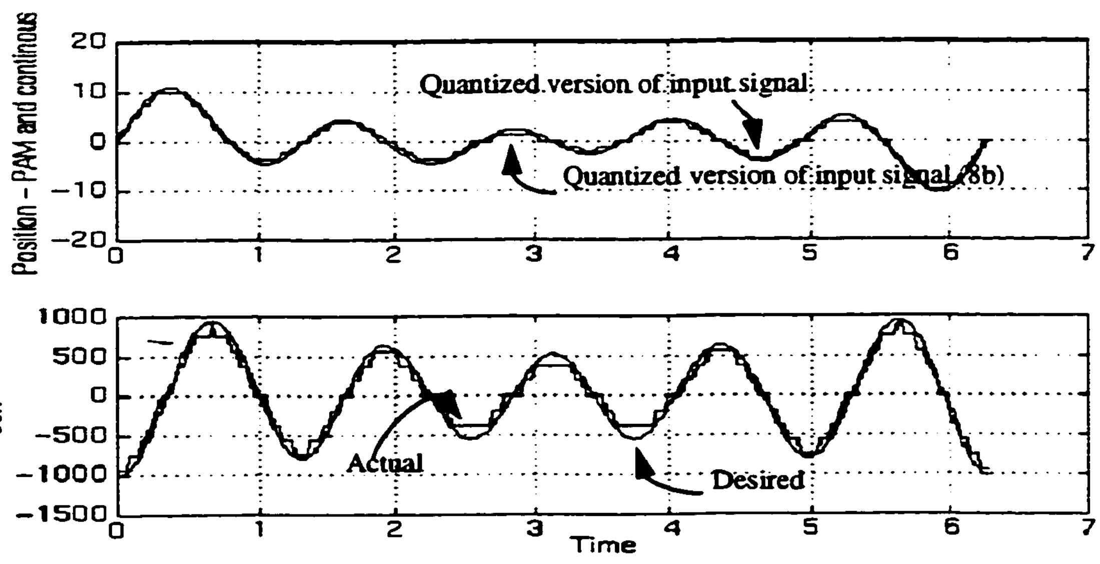


Figure 33 : Result of the neural network differentiator to generate third-order derivative when the input sample is quantized to 8 bits.

From the previous results, we see the ability of the neural networks to perform differentiation. At a quantization level of 12 bits we see that the results of the neural networks used for generating the first-order, second-order and third-order derivatives (the velocity, acceleration and jerk components respectively) are very accurate. When we decrease the bit resolution to 8 bits, we notice that the filtered signal is not as smooth as the continuous signal. Nevertheless, it is still within acceptable limits. It is the output of these networks that we will use in the servo portion of the robot controller.

We have shown the ability of the neural network smoother and the differentiators to perform their respective functions when they were given the signal with which they were trained. In the next section, we will show the ability of the neural networks to generalize - an inherent ability of neural networks.

4.3 Generalization capability of derivative estimator

The signal that we used is of the following form:

$$pos = 2\sin(t) + 0.5\varepsilon^{(-0.1t)} + 0.2\varepsilon^{(-2t)} + 2\sin(5t) + 1.2\sin(0.5t) \qquad (16)$$

This signal is composed of terms that were not present in the neural network derivative estimator at the time of training. Thus the use of this signal can be a test to see how well the neural networks can generalize. Note that we are still limiting the frequency range to 1-5 Hz. We will show the result only for the case where the position signal is quantized to 8 bits. It is implied that the results are better when the bit resolution is set to 12 bits.

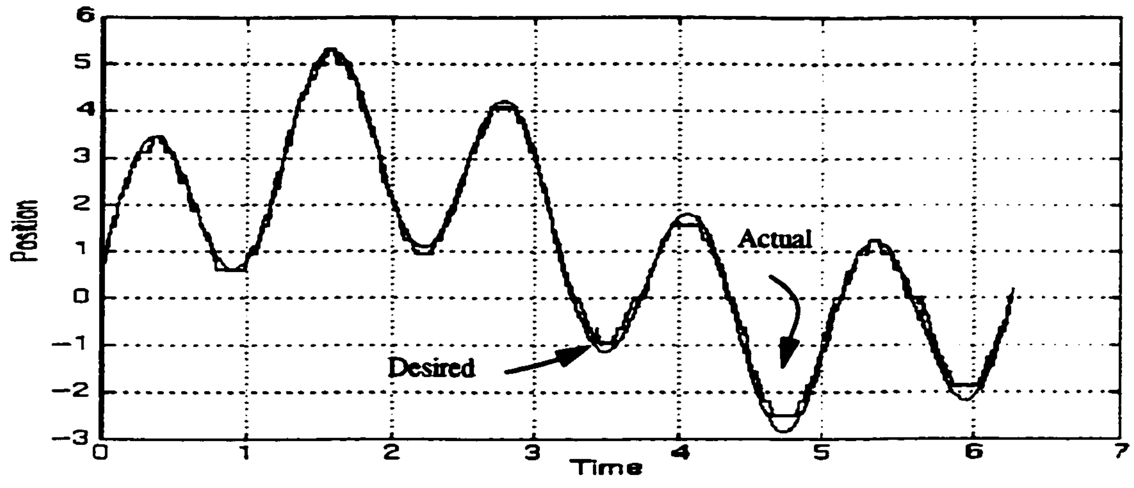


Figure 34 : Generalization capability of the NN smoother when the input sample is quantized to 8 bits.

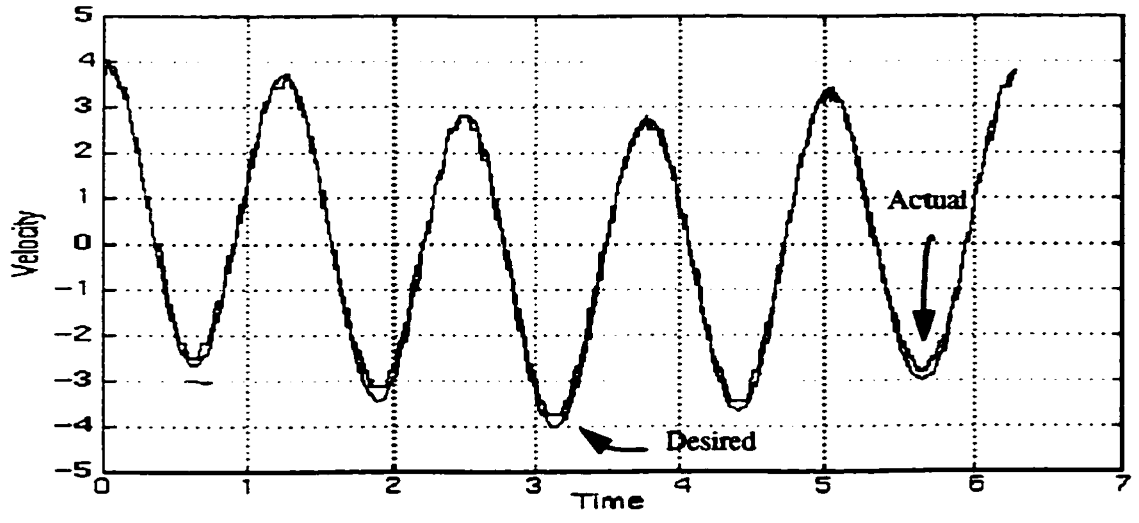


Figure 35 : Generalization capability of the NN differentiator to generate first-order derivative when the input sample is quantized to 8 bits.

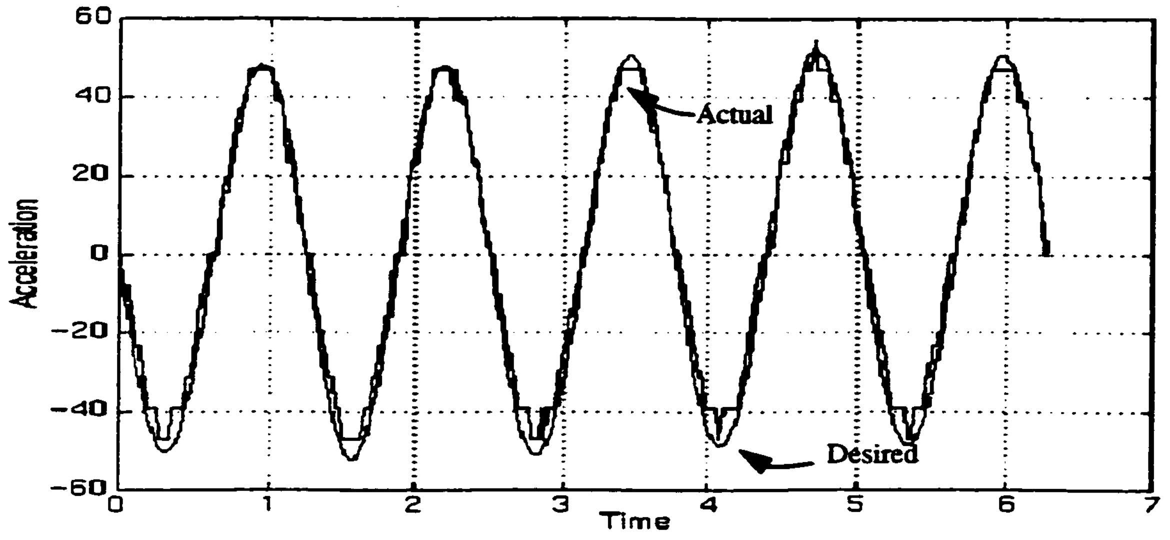


Figure 36 : Generalization capability of the NN differentiator to generate second-order derivative when the input sample is quantized to 8 bits.

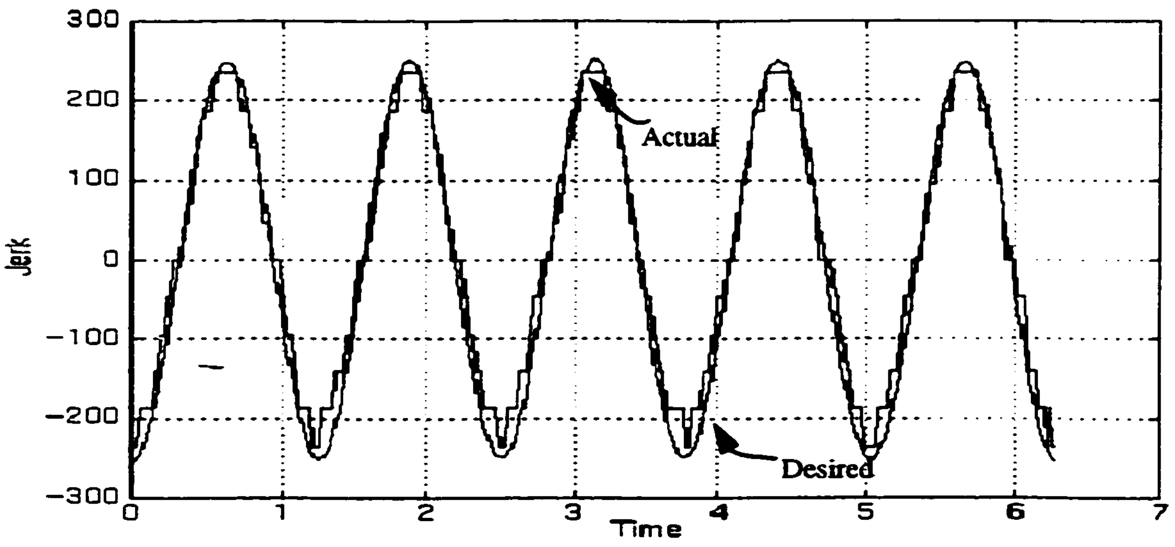


Figure 37 : Generalization capability of the NN differentiator to generate third-order derivative when the input sample is quantized to 12 bits.

Our results were very encouraging. They showed that NN derivative estimators have the capability of generalization and could be used successfully for a range of signals.

Next, we will put our neural network derivative estimator to a test within the flexible joint robot control scheme. The NN derivative estimators will be used to generate the feedback signals which will be sent to the controller. Note that this part was previously done assuming exact knowledge of position measurements (no quantization noise) or knowledge of the system dynamics.

4.4 System response with derivative estimator

After training the neural networks, we tested the closed-loop system with the following control structure:

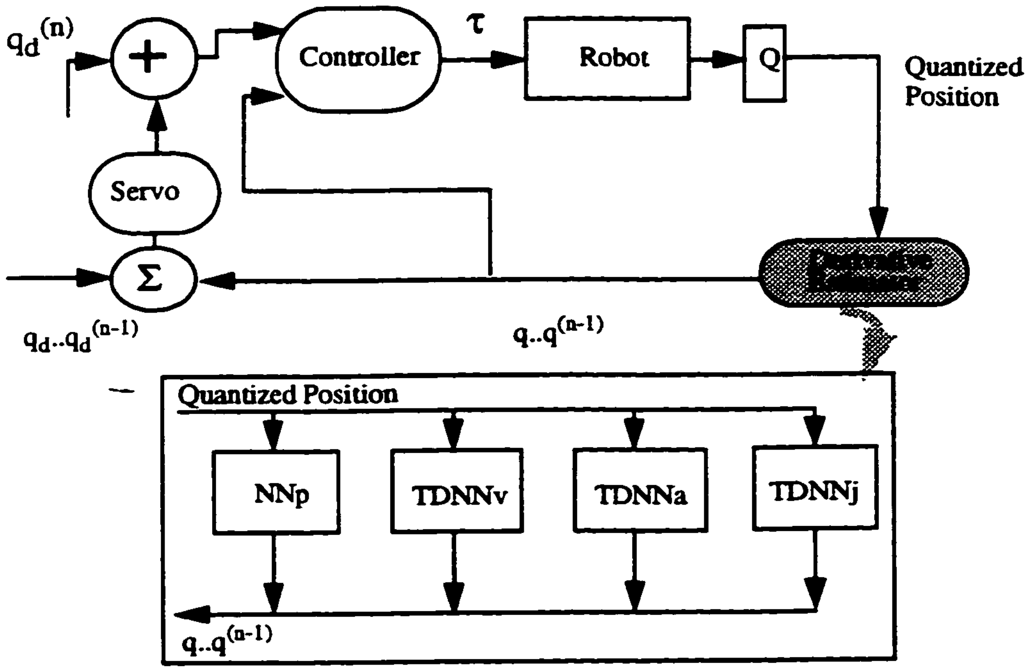


Figure 38 : Control system with the proposed methodology to determine higher-order derivative information.

The trajectory we tested was of the following form:

$$pos = 0.13\sin(3t) + 0.75\sin(0.2t) + 0.75\varepsilon^{-2t} + 0.5\varepsilon^{-0.2t} \quad (17)$$

The path shown in equation (17) is representative of the trajectory that a robot manipulator may be required to follow [11]. The results obtained are shown in Figure 39 and Figure 40. Note that the results are for tracking a desired trajectory when the sensor is quantized to a resolution of 8 bits.

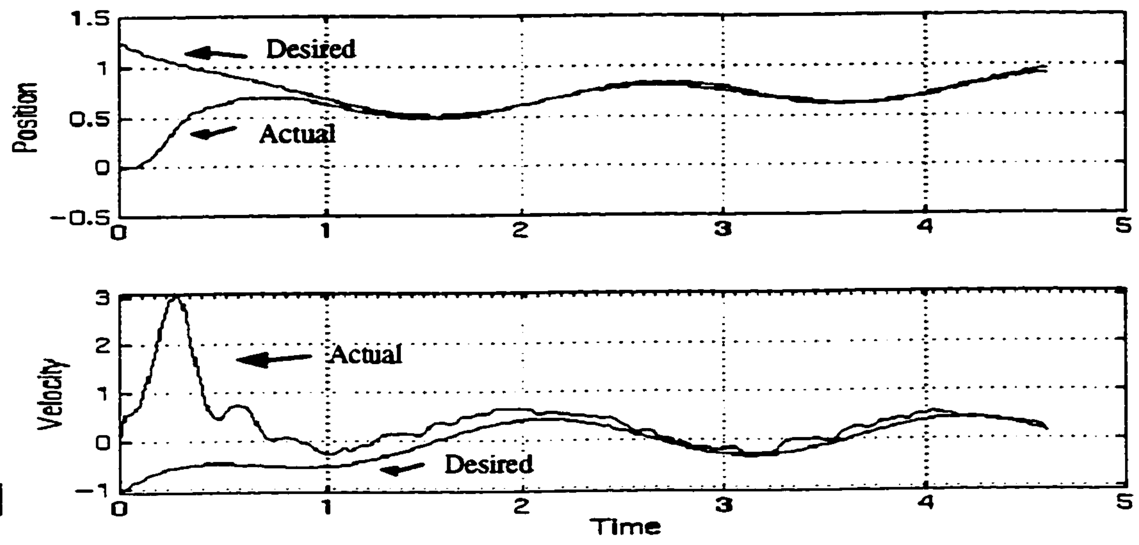


Figure 39 : Closed-loop system response showing tracking of position and velocity components with the neural network estimators.

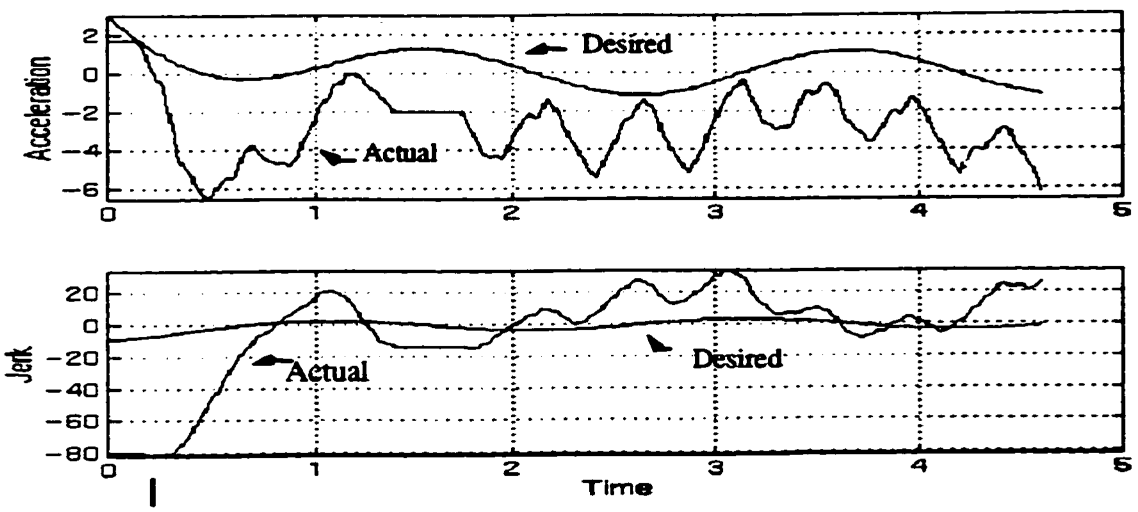


Figure 40 : Closed-loop system response showing tracking of acceleration and jerk components with the neural network estimators.

From Figure 39, we see the tracking of position is very accurate. After the initial transient response, the system error was less than 0.1%. The tracking of the velocity component, while not accurate is still within acceptable limits.

From Figure 40, we see that the tracking of the acceleration and jerk components are totally corrupted. It could be possible that these two components are not needed for the control. However, this is not the case. When we nullified the effect of the acceleration and jerk components in the servo equation (equation (16)), we get the following results for the tracking of the position and velocity components:

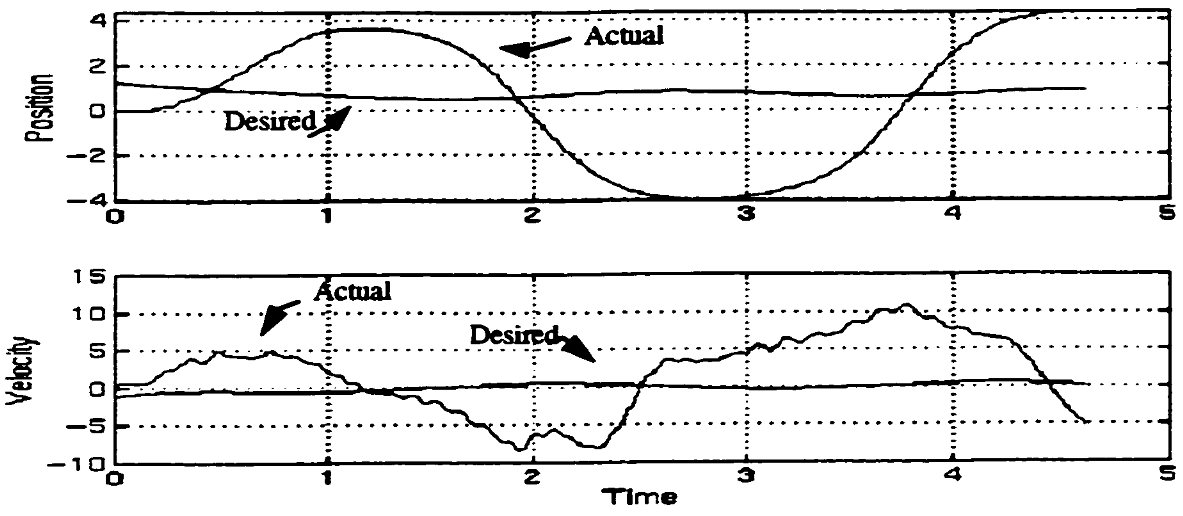


Figure 41 : Closed-loop system response when the acceleration and jerk components are nullified

We see that despite the fact that we were not able to track the acceleration and jerk components, they play a significant role in tracking of the desired trajectory. However, it is more important to track the position and velocity components than to track the acceleration and jerk components. It is the integrity of these components that we will try to maintain when we derive alternatives to the scheme.

From the results we discover another very interesting fact about our neural network based approach for determining the feedback signals. Earlier in Section 2.2.2.2 we had mentioned that a commercial sensor can measure position with a 12-bit resolution range of 0-$\pi/2$ degrees joint motion. From our results we see that we have achieved an 8-bit resolution over this range. This is indeed very promising since it reduces the overall cost of the sensor system.

As further proof of the concept of our methodology, we present results for the different desired trajectories for the flexible-joint manipulator. The desired trajectory was of the following form:

$$pos = 0.75e^{-2t} + 0.5e^{-0.2t}$$

This trajectory represents exponential signals. The derivative estimator neural networks, if trained properly, will be able to generate the required derivative information and thus allow the desired trajectory to be tracked. The results are shown in Figure 42.

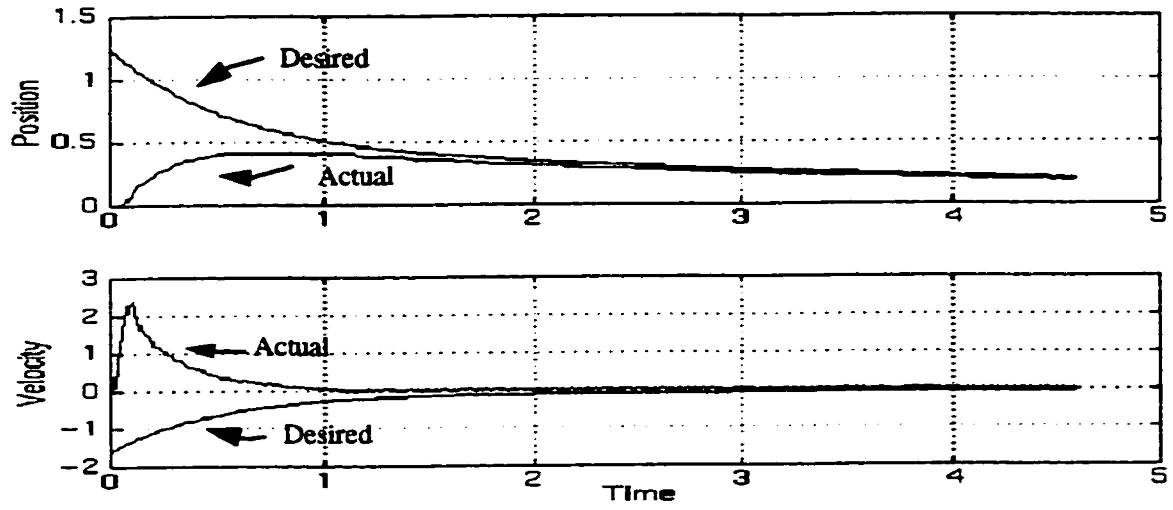


Figure 42 : Closed-loop system response with exponential terms.

From the above results we see that even when a considerably different signal than the one used in training is applied, we can achieve excellent tracking. Other trajectories we tested were of the "pick-and-place" form and were composed of exponential terms. An example of this is given below:

$$pos = 0.75e^{-2t} + 0.5e^{-0.2t} + t - 3$$

The results for the above trajectory are shown in Figure 43:

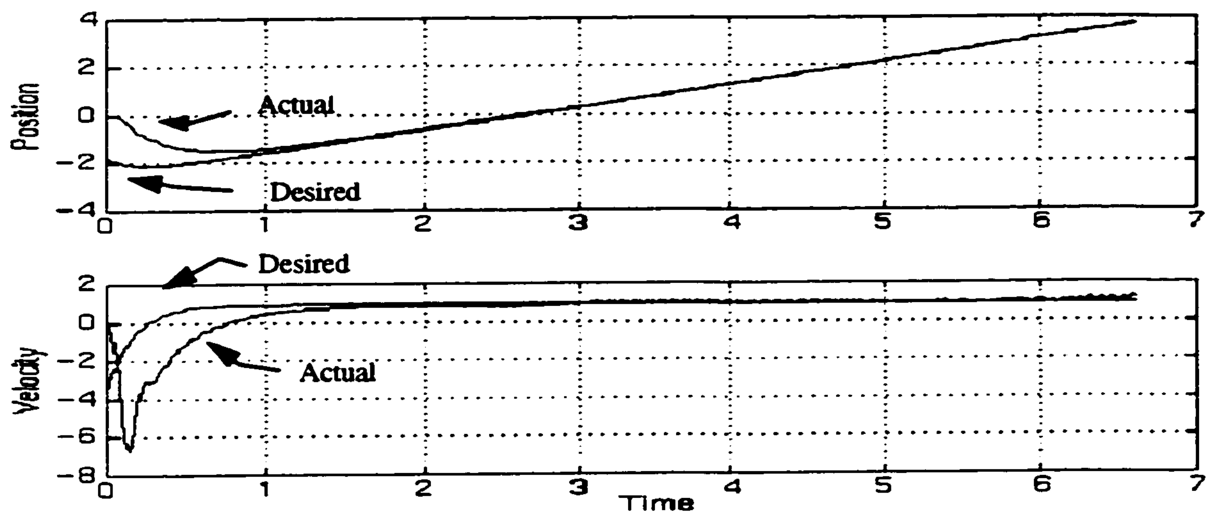


Figure 43 : Closed-loop system response of a pick-and-place trajectory composed of exponential terms.

Again we achieved successful tracking of the desired path. Several other trajectories were also tested. In all cases, the controller was successfully able to track the desired trajectory. One of the more complicated trajectories tested was of the form:

$$pos = 4.5\sin(3t) + 0.75\cos(2t) + 0.75\varepsilon^{-0.3t} + 0.5\varepsilon^{-0.2t} - 3t - 3$$

The results are shown in Figure 44:

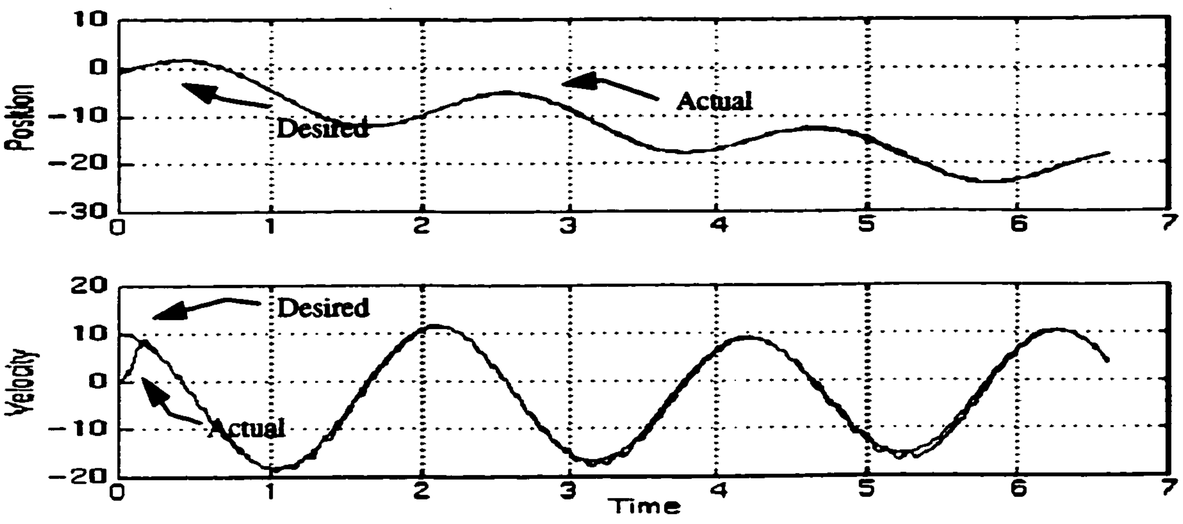


Figure 44 : Closed-loop system response of a complex trajectory of exponential and sinusoidal terms.

We indicated in Section 3.2.1 that the neural networks had been trained for upto a frequency of 5Hz. We changed the frequency of one particular component to 6Hz and tested the response of our closed-loop system. The trajectory was of the following form:

$$pos = 0.13\sin(6t) + 0.75\sin(0.2t) + 0.75\varepsilon^{-2t} + 0.5\varepsilon^{-0.2t} - 3t$$

The results are shown in Figure 45:

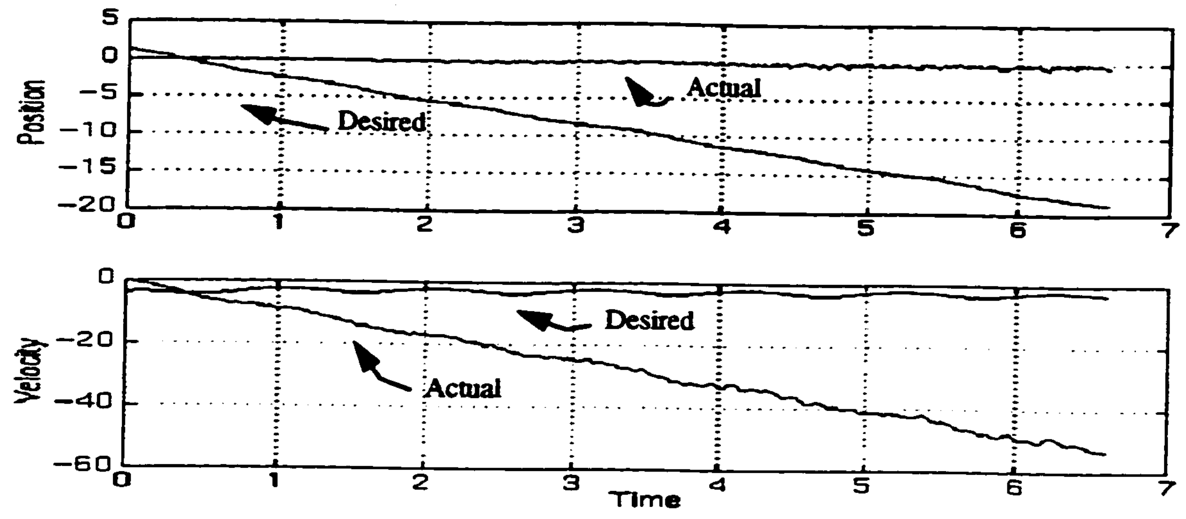


Figure 45 : Closed-loop system response with NN derivative estimators and a higher-frequency component.

We can see from the results that the trajectory cannot be tracked. This seems to indicate that for the NN derivative estimator to work, the desired trajectory must be composed only of the frequency components that are present in the original training signals. This is consistent with what is known about neural network training and recall.

At the beginning of this chapter, we stated that we tested our closed-loop control system when our neural network derivative estimators were undersampled and oversampled. The results of this investigation indicate that the same sampling frequency must be used in recall as the one used at the time of training. We indicate the results of the experiment for both under-sampling and over-sampling in Figure 46 and Figure 47:

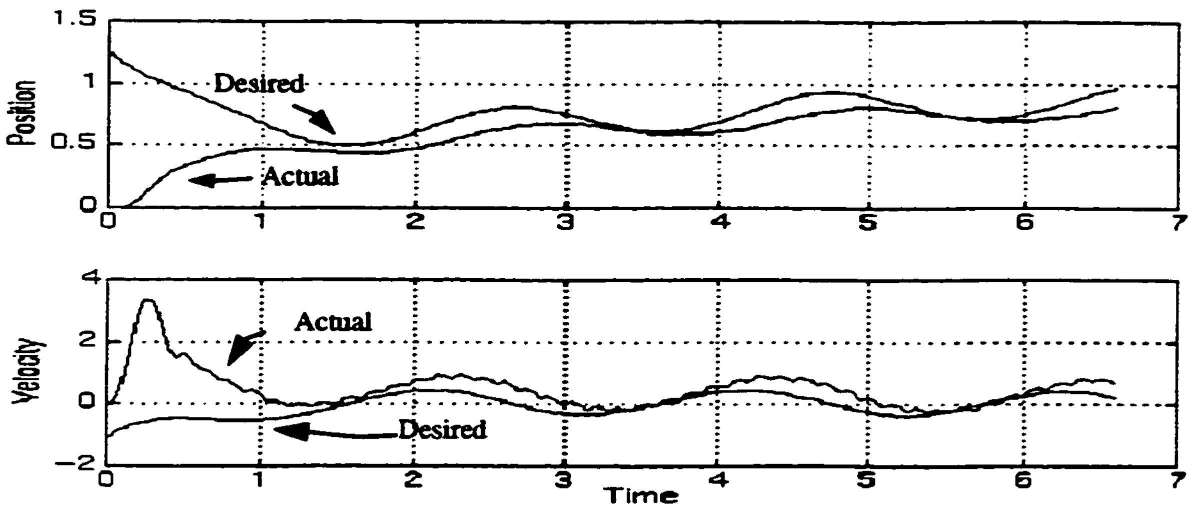


Figure 46 : Closed-loop system response when sampling rate is decreased.

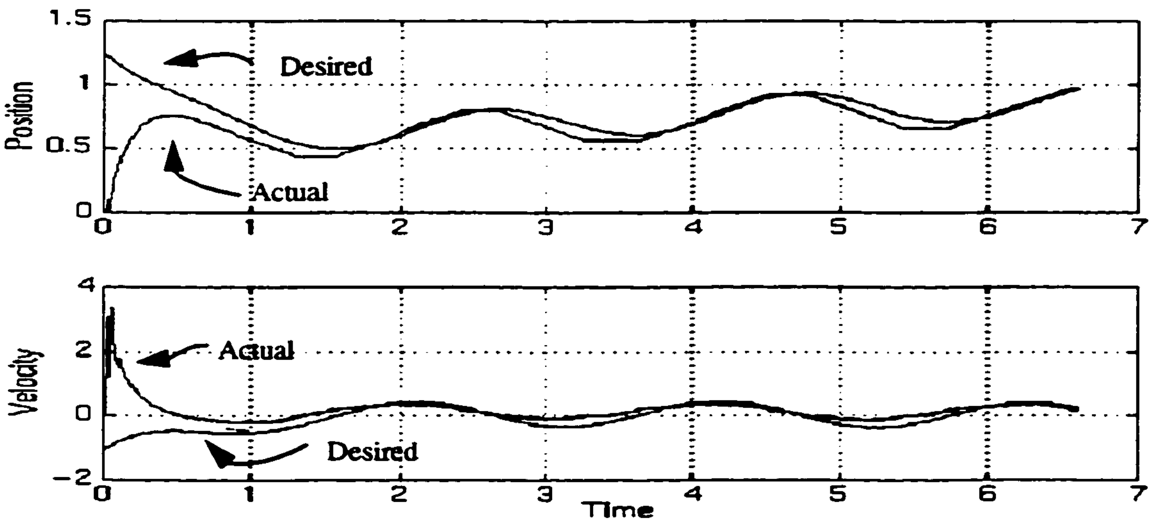


Figure 47 : Closed-loop system response when sampling rate is increased.

4.5 Summary

In this chapter, we showed that neural networks can be trained to perform as low-pass filters and also as differentiators. In addition we showed the generalization capability of our derivative estimators. In both cases, the simulation results indicate that the neural networks fulfilled the functionality for which they were trained.

Next, we performed the ultimate test on our derivative estimators: we incorporated the derivative estimator architecture into the control system for a flexible joint manipulator and investigated the tracking of several different trajectories. Our criterion for judging the success of our proposed methodology was how well the desired trajectory is tracked. The results were encouraging and showed that a neural network strategy can be used to determine the derivative feedback signals. The simulation results indicate that the error between the actual path and the desired path was well within acceptable limits. This strategy also satisfied our initial goals of having a means to determine the control parameters without any *a priori* knowledge of the robot's dynamics or exact knowledge of position measurements and the minimum use of hardware to determine the control parameters. Our proposed methodology also proved that we can use an 8-bit sensor over a $0-\pi/2$ range of joint motion to obtain the derivative feedback signals.

We can see from the structure of our neural network derivative estimator that we have to perform a large number of computations. In total, for all the neural networks, there are 6266 multiply-accumulate operations. Assuming this algorithm is coded on a typical DSP chip with a clock rate of 33MHz, the total time will amount to 208ms. It would be to our advantage to reduce the number of computations and thus reduce the time taken for the control parameters to be generated. With this in mind, we explored the closed-loop system performance with variants of the neural network derivative estimator. This is the subject of discussion in Chapter 5.

Chapter 5
Considerations for Real Time Implementation.

In this chapter, we optimize the architecture of the neural network derivative estimator so as to reduce the number of computations. We present three variants to the neural network based estimator shown in Figure 24. The first alternative architecture consists of replacing the neural network differentiators to compute acceleration and jerk signals by computational machines which perform first order differentiation i.e. the resulting signal will be determined by the difference of two consecutive samples. The first derivative is still determined by a TDNN. In the second alternative architecture, all the derivative information is determined by first order differentiation. In the third alternative, we modify the activation function. We trained our neural network differentiators using a sigmoid activation function. During recall (i.e. when the neural network derivative estimators were placed in the closed-loop system of the flexible-joint manipulator), we replace the sigmoid activation function by a piece-wise linear activation function. It is important to note that we did not train the networks again with the new activation function.

As we can see, the proposed changes modify the original structure substantially. The aim of these changes is to reduce the total number of multiply-accumulate operations and thus the computation time. Our criterion for judging the success of our new architectures will again be by how well the desired trajectory is tracked. Additionally, we compared the performance of our closed-loop system with the performance when a linear filter (5-th order elliptic filter) is used as part of the derivative estimator.

5.1 First alternative to the derivative estimator

The first alternative we propose to an all neural network based estimator is shown in Figure 48:

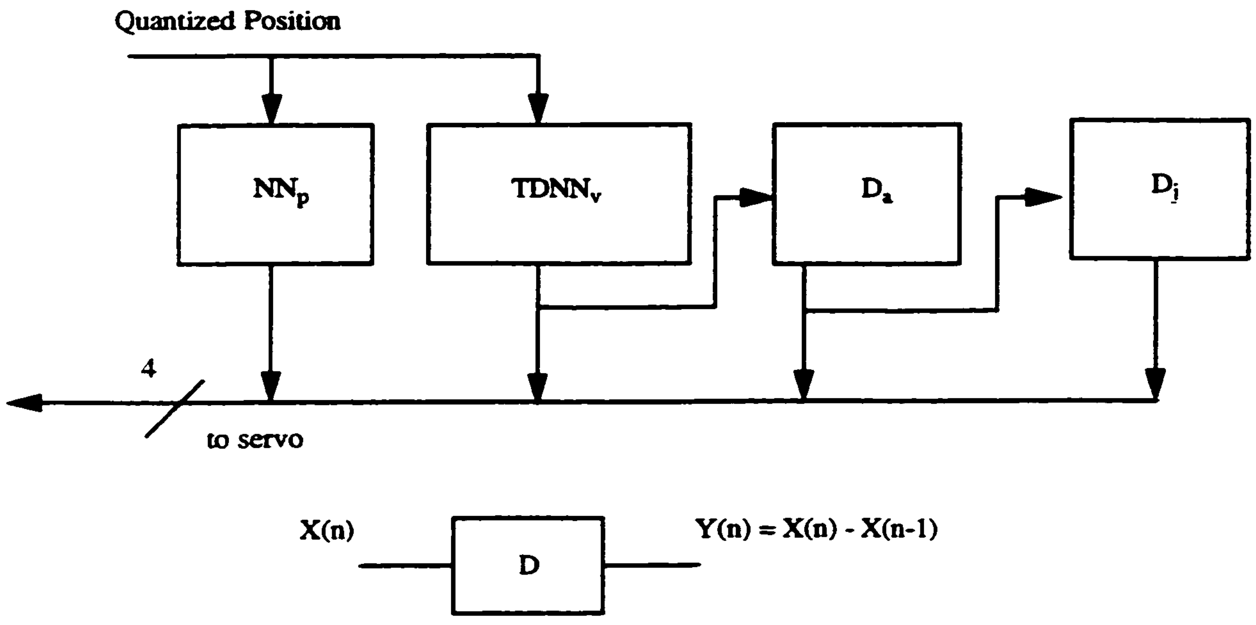


Figure 48 : Alternative 1 model.

In Alternative 1, we use a neural network to filter out the quantization noise present in the position signal and a TDNN to generate the velocity signal. The other signals (acceleration and jerk) are determined using first-order differentiation. The results that we obtained are shown on Figure 49:

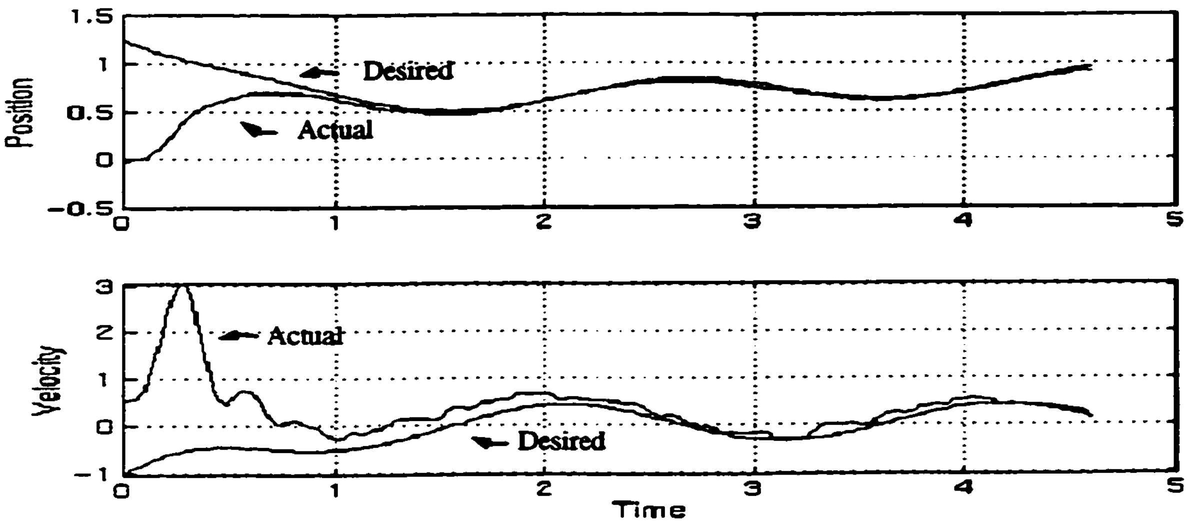


Figure 49 : Closed-loop system response showing tracking of position and velocity components with Alternative 1 model.

Again, we see the success of our new architecture. We have maintained the integrity of resulting position signal. Furthermore, we have reduced the total number of multiply-accumulate operations to 2128. The total time taken for this number of operations (assuming a 33MHz clock rate) is estimated to be 70 ms - almost three times faster than the all neural network estimator.

5.2 Second alternative to the derivative estimator

The second alternative utilizes a neural network to filter out the quantization noise, and differentiation to generate the higher-order information. The resulting architecture and its corresponding result are shown on Figure 50:

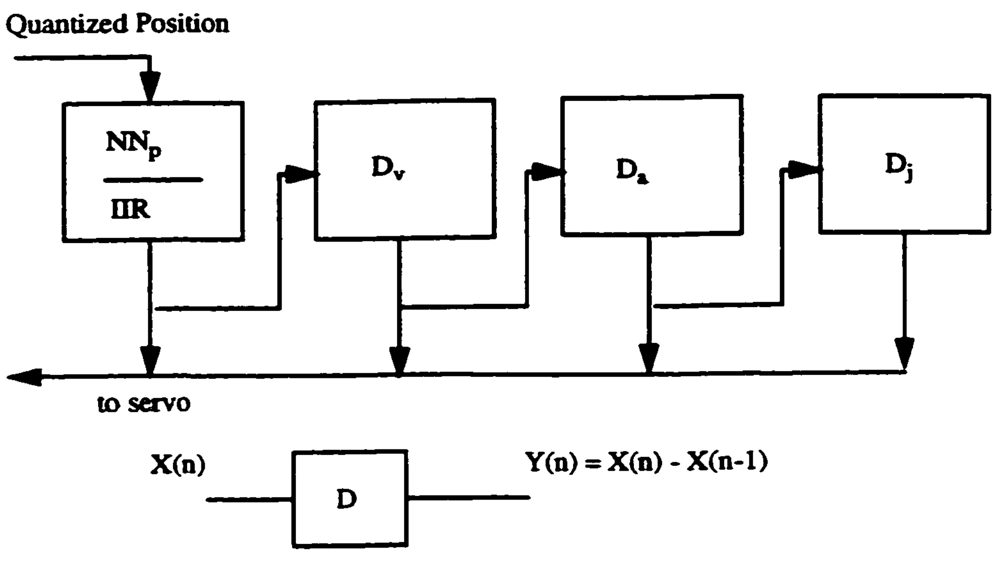


Figure 50 : Alternative 2 model.

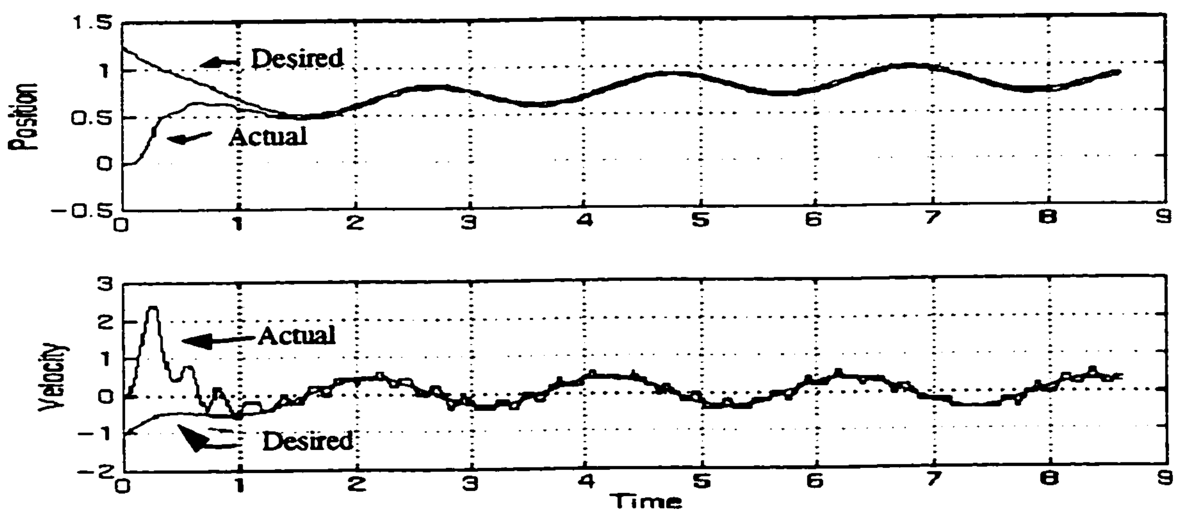


Figure 51 : Closed-loop system response showing tracking of position and velocity components with Alternative 2 model.

Again, we see the success of the resulting architecture. In fact, in this architecture we see that the tracking of the velocity is more accurate than in the previous architectures. This is because we are using first-order differentiation from the position signal. The more accurate the tracking of the original signal, the more accurate will be the tracking of the corresponding derivative. Using a neural network adds non-linearities to the resulting signal and corrupts it to a certain extent. The total number of multiply accumulate operations is reduced to 44. The total time taken for this number of operations (assuming a 33 MHz clock rate) is estimated to be 1.5 ms. As we mentioned earlier, the filtering of the quantization noise can be carried out using either an FIR or an IIR filter. We have demonstrated in earlier sections how we propose to use an FIR neural net filter. To have a basis for comparison, we also implemented a 5th-order elliptic filter for the purpose of removing the quantization noise (See Figure 50). The result is shown in Figure 52:

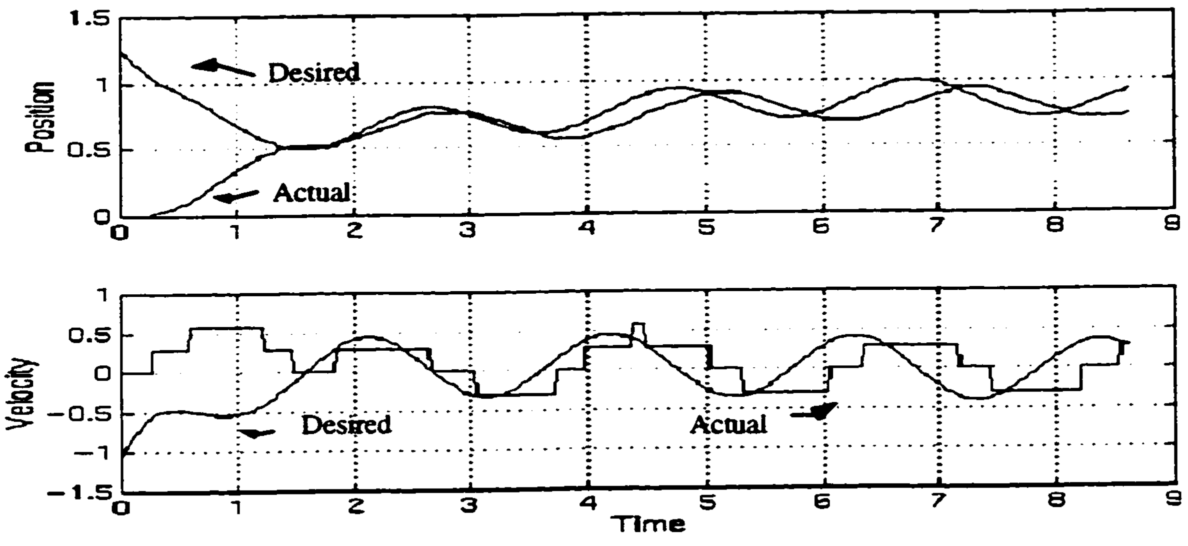


Figure 52 : Response using a linear filter.

We see that our model performs better than an IIR approach. We have tried several different types of the IIR filter - Chebychev, Butterworth and different orders. However, we obtained almost the same results with each architecture.

5.3 Third alternative to the derivative estimator

All the neural networks we have used so far employ a sigmoid activation function. Though a CORDIC solution exists to implement transcendental functions, it is inefficient in terms of wasted CPU cycles. This is especially important in an ASIC digital implementation. An approach that we recommend is to use a piecewise linear activation function. This is shown in Figure 54 along with a table of the gains in the respective regions. By varying the magnitude of these slopes, we approximate the sigmoid activation function and achieve the same performance. The magnitudes of the slopes were selected by a curve fitting approach. The results reported use the derivative estimator architecture proposed in the Alternative 2 model.

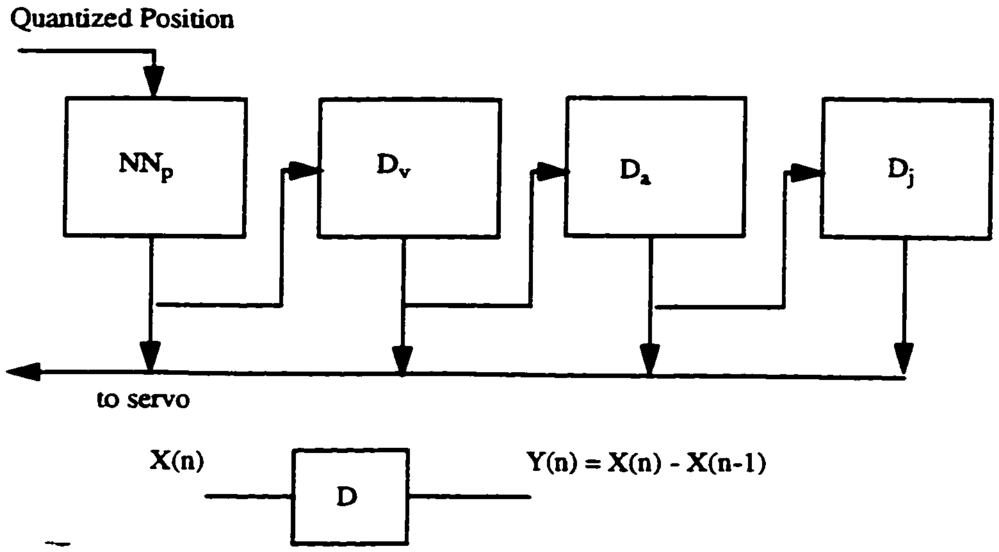
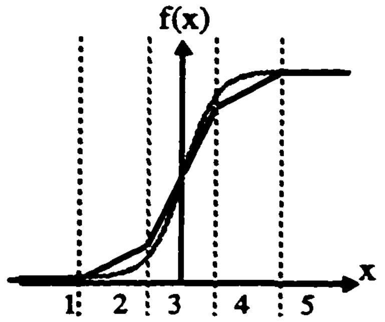


Figure 53 : Alternative 3 model[3].

[3]The activation function is of the form shown in Figure 54.



Region	Range	Gain
1	-∞ to -1.00	0.1
2	-1.00 to -0.75	0.3
3	-0.75 to +0.75	0.6
4	-0.75 to +1.00	0.3
5	+1.00 to +∞	0.1

Figure 54 : Piece-wise linear activation function characteristics.

The corresponding results are shown in Figure 55:

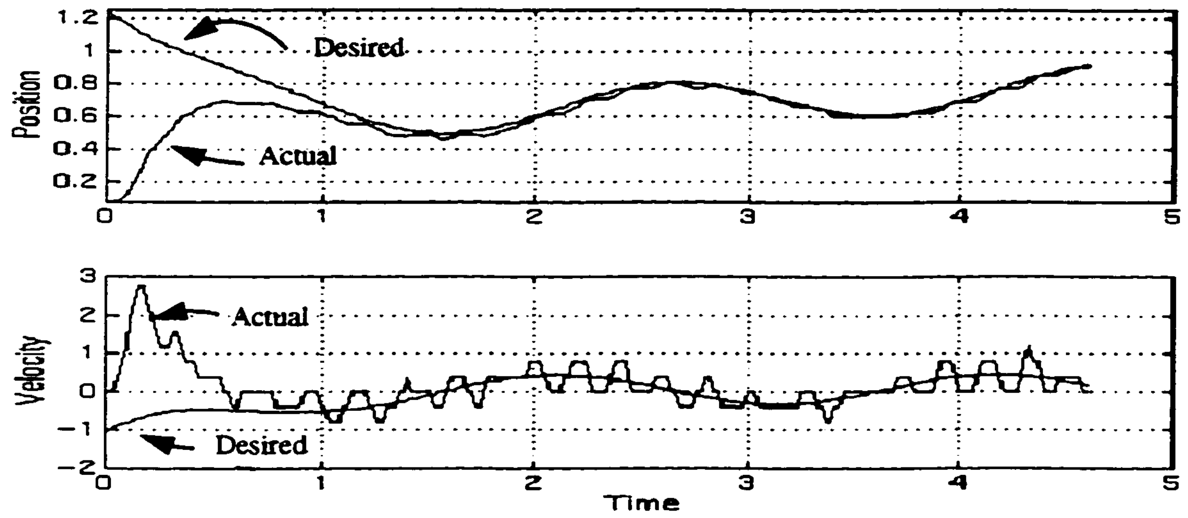


Figure 55 : Closed-loop system response when using a piece-wise linear activation function.

We see that the integrity of the desired trajectory is maintained despite there being some noise in the resulting signal. The resulting velocity signal is thus more noisy but still within acceptable limits. The total number of multiply-accumulate operations is now 40 and the total computational time is estimated to be 1.2ms - a substantial reduction from the original 200ms.

5.4 Summary

The motivation behind this chapter was the desire to reduce the total number of computations performed by the neural networks in the original derivative estimator architecture. We explored three variants of the afore-mentioned architecture.

In Figure 56, we summarize the results:

Figure 56 : Summary of performance of different derivative estimator architectures

TYPE OF DERIVATIVE ESTIMATOR ARCHITECTURE	TOTAL NUMBER OF MULTIPLY-ACCUMULATE OPERATIONS	TIME TAKEN[a] (ms)
All neural network (Figure 24)	6266	208
Alternative 1	2128	70
Alternative 2	44	1.5
Alternative 3	40	1.2

a. All time units are estimations.

We can see that substantial savings have occurred both with respect to the total number of computations and the time taken from the original all neural network based derivative estimator architecture. The ideal derivative estimator architecture is of the Alternative 3 model. Additionally, we compared this scheme to a linear filtering scheme using a 5th-order elliptic filter. Again the superiority of our approach was quite apparent.

Chapter 6
Conclusion

At the start of the thesis, we made a claim that a control algorithm developed without any regard to how the feedback signals will be measured cannot fulfill its objective of controlling the system when hardware implementations consideration are taken into account. To demonstrate this hypothesis, we used the control algorithm presented in [1] which is concerned with the problem of controlling a flexible-joint manipulator. We showed that when a sensor with a 8-12-bit quantization level is incorporated into the control system to determine the control parameters, the controller cannot track the desired trajectory.

In this thesis we proposed a methodology to alleviate this practical and essential implementation problem not addressed in theoretical approaches. This methodology consists of using neural networks. The neural networks can be divided in two blocks: the first part consists of filtering the quantization noise arising as a result of the signal being measured by a sensor, and the second part consists of neural networks to determine the first and higher-order derivatives required for feedback.

The neural network approach has been the preferred solution because it obviates the need for a mathematical knowledge of the systems dynamics and can be implemented in hardware in real time.

The simulation results showed the success of our proposed methodology. Our criterion for judging the success of our approach was how well a desired trajectory is tracked. This criterion was satisfied by our design.

As neural networks are very computationally intensive. we investigated architectures where we attempted to reduce the number of computations and thus improve real time performance. For these architectures, our simulation results were very promising. Again our criterion of tracking the desired trajectory with little error was successfully met. Additionally, we compared our scheme to that using linear filtering techniques. We have achieved far superior results both in terms of accuracy and real time response.

In this thesis, we have shown the training procedure for the neural networks used to filter quantization noise and to perform the task of differentiation. Our simulation results have shown that for practical quantization levels (8-12 bits), control of a non-linear dynamical system requiring derivative information can be achieved using neural networks for providing accurate derivative feedback.

The emphasis of this research has been on solving the problem of determining the feedback signals for a flexible-joint manipulator, but our approach can be applied to any application where derivatives are needed for the control of a system.

There are several ways in which the results of this thesis can be developed further. From a neural network perspective, we can utilize different algorithms to perform the task of filtering the quantization noise and differentiation. An algorithm we suggest is the Cerebellar Model Articulation Controller (CMAC). CMAC is capable of fast learning and is excellent at interpolation and approximation. Comparison between the performance of the neural networks trained by the CMAC algorithm and those trained by the Back-propagation algorithm (which was used in this work) should prove interesting. From a VLSI systems perspective, we can construct a DSP architecture using synthesis tools. An investigation into the design space of an optimal DSP system architectures is another possibility for investigation.

Chapter 7
References

[1] V. Zeman, *"A Neural Network Based Approach To The Control Of Flexible Joint Manipulators"*, M.A.Sc Thesis, Concordia University, 1991

[2] M.W.Spong, M. Vidyasagar, *"Robot Dynamics and Control"*, John Wiley and Sons, New York, 1989.

[3] Douglas M. Charney and Gary M. Josin, *"Neural Network Servo Control of a Joint Manipulator in Real Time"*, Int. Joint Conference on Neural Networks, pp. 1989-1994, 1992.

[4] Chi-Tsong Chen, *Analog and Digital Control System Design: Transfer Function, State Space, and Algebraic Methods*, Fort Worth: Saunders College Publication, 1993

[5] T. Troudet, S.Garg, D. Mattern, W. Merrill, *"Towards Practical Control Design Using Neural Computation"*, Int. Joint Conference on Neural Networks, vol II, pp II675-II680, 1990.

[6] Q. Li, C.L.Teo, A.N.Poo, G.S.Hong, *"Response of a Feedback System with a Neural Network Controller in the Presence of Disturbances"*, Int. Joint Conference on Neural Networks (Singapore), pp.1560-1565, 1991.

[7] M. Tokita, T. Mituoka, T. Fukuda, T. Shibata, F. Arai, *"Position and Force Hybrid Control of Robotic Manipulator by Neural Networks"*, Int. Joint Conference on Neural Networks, pp.113-121, 1991.

[8] S. Zak, *"Robust Tracking Control of Dynamic Systems with Neural Networks"*, Int. Joint Conference on Neural Networks, vol II, pp II563-II566, 1990.

[9] A. Waibel, H. Sawai, K. Shikano, *"Consonant recognition by modular construction of large phonemic time-delay neural networks"*, Proceedings of the IEEE Int. Conference on Acoustics, Speech and Signal Processing, Glasgow, Scotland, vol. 1, pp. 112-115, 1989.

[10] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, K. Lang, *"Phoneme Recognition Using Time-Delay Neural Networks"*, IEEE Transactions on Acoustics, Speech and Signal Processing, vol. 37, pp. 328-339, 1989.

[11] J. Craig, *"Introduction to Robotics"*, Addison-Wesley Publishing Co., Reading, MA 1986.

[12] R. Dubey, J. Euler, *"Real time Implementation of an Optimization Scheme for Seven-Degrees-Of-Freedom Redundant Manipulators"*, IEEE Transactions on Robotics and Automation, Vol. 7, No. 5, pp. 235-256, October 1991

[13] C.G. Lee, P.R. Chang, *"A Maximum Pipelined CORDIC Architecture for Inverse Kinematic Position Computation"*, IEEE Journal of Robotics and Automation, Vol RA-3, No. 5, October 1987.

[14] R. Hecht-Nielson, *"Neurocomputing"*, Reading, MA: Addision-Wesley,1990.

[15] Peter Hauptmann, *"Sensors: principles & applications"*, Englewood Cliffs, NJ: Prentice Hall, 1993.

[16] C.A.Rahenkamp, B.V.K.V. Kumar, *"Modifications to the McClellan, Parks and Rabiner computer program for designing higher-order differentiating FIR filters,"* IEEE trans. Acoust., Speech, Signal Processing, vol.

ASSP-34, pp.1671-1674. Dec. 1986.

[17] S.C.Pei, J.J.Shyu, *"Eigenfilter design of higher-order digital differentiators,"* IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-37, pp. 505-511. April 1989.

[18] S. Sunder, R.P.Ramachandran, *"Least-Squares Design Of Higher-Order Non-Recursive Differentiators"*, IEEE Transactions on Signal Processing, April 1994.

[19] L.R.Rabiner, K. Steiglitz, *"The design of wideband recursive and nonrecursive digital differentiators"*, IEEE Trans. Audio Electroacoustics, vol. AU-18, pp. 204-209, June 1970.

[20] A.T.Chottera, G.A.Jullien, *"A linear programming approach to recursive filter design with linear phase"*, IEEE Trans. Circuits Systems, vol. CAS-29, pp 139-149, March 1982.

[21] Alan V. Oppenheim, Ronald W. Schafer *"Discrete-time Signal Processing"*, Englewood Cliffs, NJ: Prentice Hall, 1989.

Chapter 8

Appendix

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%             PROGRAM 1
%%%%%%%    This program models the control    %%%%%%%%%%
%%%%%%%    system with the all NN estimator   %%%%%%%%%%
%%%%%%%                                       %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

y0 = [0; 0; 0; 0];
netvec = [0; 0; 0; 0; 0];
q = [0; 0; 0; 0; 0];
qd = [0; 0; 0; 0; 0];
netout = 0;
index = 1;

load POSNN
W1pos = W1;
W2pos = W2;
B1pos = B1;
B2pos = B2;

load vel
W1vel = W1;
W2vel = W2;
W3vel = W3;
B1vel = B1;
B2vel = B2;
B3vel = B3;

load nndiffacc1
W1acc = W1;
W2acc = W2;
W3acc = W3;
B1acc = B1;
B2acc = B2;
B3acc = B3;

load nndiffjrk1
W1jerk = W1;
W2jerk = W2;
W3jerk = W3;
B1jerk = B1;
B2jerk = B2;
B3jerk = B3;

xn20=0;
xn19=0;
xn18=0;
xn17=0;
xn16=0;
xn15=0;
xn14=0;
xn13=0;
xn12=0;
xn11=0;
xn10=0;
xn9=0;
xn8=0;
xn7=0;
xn6=0;
xn5=0;
xn4=0;
xn3=0;
xn2=0;
xn=0;

xxx = [xn; xn2; xn3; xn4; xn5; xn6; xn7; xn8; xn9; xn10; xn11; xn12; xn13; xn14;xn15; xn16; xn17; xn18; xn19; xn20];
```

```
for i=0:0.02:10
i
 tvec = [tvec; i];
 q = observer(y0,netout);

 xn20=xn19;
 xn19=xn18;
 xn18=xn17;
 xn17=xn16;
 xn16=xn15;
 xn15=xn14;
 xn14=xn13;
 xn13=xn12;
 xn12=xn11;
 xn11=xn10;
 xn10=xn9;
 xn9=xn8;
 xn8=xn7;
 xn7=xn6;
 xn6=xn5;
 xn5=xn4;
 xn4=xn3;
 xn3=xn2;
 xn2=xn;
 xn= y0(1);

 xxx = [xn; xn2; xn3; xn4; xn5; xn6; xn7; xn8; xn9; xn10; xn11; xn12; xn13; xn14; x
 n15; xn16; xn17; xn18; xn19; xn20];

 xxv = xxx/10;
 xxa = xxx/50;
 xxj = xxx/75;

 pos1 = tansig(W1pos*xn/1, B1pos);
 pos2 = tansig(W2pos*pos1, B2pos);
 apos(index) = pos2*2;

 vel1= tansig(W1vel*xxv, B1vel);
 vel2= tansig(W2vel*vel1, B2vel);
 vel3= tansig(W3vel*vel2, B3vel);

 acc1 = tansig(W1acc*xxa, B1acc);
 acc2 = tansig(W2acc*acc1, B2acc);
 acc3 = tansig(W3acc*acc2, B3acc);

 jrk1 = tansig(W1jerk*xxj, B1jerk);
 jrk2 = tansig(W2jerk*jrk1, B2jerk);
 jrk3 = tansig(W3jerk*jrk2, B3jerk);

 qn(1) = apos(index);
 qn(2) = vel3*75;
 qn(3) = acc3*250;
 qn(4) = jrk3*300;
 qn(5) = apos(index);

 qd = desirin(i);
 expout = [expout; qn];
 theout = [theout; qd];
 tmp1 = servo(qd,qn,160000,32000,2400,80);
 serverr = [serverr; tmp1];
 netvec = qn;
 netvec(5) = tmp1;
 neti = [neti; netvec];
 netin = netvec;
 netout = theoru(netin,y0);
 netsv = [netsv; netout];
 [T,Y] = cinode45('dynplant2',i,i+0.02,y0,netout);
 tmp2 = length(Y);
 y0 = Y(tmp2,:);
end

temp1=max(abs(neti(:,1)));
temp2=max(abs(neti(:,2)));
temp3=max(abs(neti(:,3)));
temp4=max(abs(neti(:,4)));
```

```
temp5=max(abs(neti(:,5)));
torque=max(abs(netsv));

neti(:,1) = neti(:,1)/temp1;
neti(:,2) = neti(:,2)/temp2;
neti(:,3) = neti(:,3)/temp3;
neti(:,4) = neti(:,4)/temp4;
neti(:,5) = neti(:,5)/temp5;
netsv1=netsv/torque;

indata = neti';
desout = netsv1';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%              PROGRAM 2                    %%%%%%%
%%%%%%%     This program models the control       %%%%%%%
%%%%%%%     system using Zemans methodology       %%%%%%%
%%%%%%%                                           %%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear
resolution = 18;
y0 = [0; 0; 0; 0];
netvec = [0; 0; 0; 0; 0];
q = [0; 0; 0; 0; 0];
qd = [0; 0; 0; 0; 0];
netout = 0;
index = 1;

%%% load file generated from output of nn
load try;
for i=0:0.02:4.6
i
tvec = [tvec; i];
q = observer(y0,netout);
qd = path10(i);
expout = [expout; q];
theout = [theout; qd];
%%tmp1 = servo(qd,q,560000,192000,7400,80);
tmp1 = servo(qd,q,560000,200000,7400,80);
serverr = [serverr; tmp1];
netvec = q;
netvec(5) = tmp1;
netin = prescale(netvec);
netvec = netin;
if netvec(1) > 1
netvec(1) = 1;
end
if netvec(1) < -1
netvec(1) = -1;
end
if netvec(2) > 1
netvec(2) = 1;
end
if netvec(2) < -1
netvec(2) = -1;
end
if netvec(3) > 1
netvec(3) = 1;
end
if netvec(3) < -1
netvec(3) = -1;
end
if netvec(4) > 1
netvec(4) = 1;
end
if netvec(4) < -1
netvec(4) = -1;
end
if netvec(5) > 1
netvec(5) =1;
end
if netvec(5) < -1
```

```
    netvec(5) = -1;
    end
    neti = [neti; netin];
    A1 = tansig(W1*netin', B1);
    A2 = tansig(W2*A1, B2);
    netout = tansig(W3*A2, B3);
   % netout = theoru(netin,y0);
    netsv = [netsv; netout];

   %%% Always change the value at the end of the following line to max torque
   %%% generated from simtheor.m variable max(abs(netsv))

   netout = netout * 1.6373e+04;
   [T,Y] = cinode45('dynplant2',i,i+0.02,y0,netout);
   tmp2 = length(Y);
   y0 = Y(tmp2,:);

   pos = y0(1);
   pos = convert(pos, resolution);
   y0(1) = pos;


  end

  subplot(2,1,1)
  plot(tvec, expout(:,1),'yellow', tvec, theout(:,1),'yellow')
  grid
  ylabel('Position')


  subplot(2,1,2)
  plot(tvec, expout(:,2),'yellow', tvec, theout(:,2),'yellow')
  grid
  ylabel('Velocity')
  xlabel('Time')

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%%%%%%                 PROGRAM 2          %%%%%%%
  %%%%%%%      This program trains the position      %%%%%%%
  %%%%%%%      neural network differentiator       %%%%%%%
  %%%%%%%                                    %%%%%%%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  testpos
  disp('input finished')
  inps=1;
  Lay1=10;
  Lay2 = 10;
  Outps=1;

  [W1,B1] = rands(Lay1,inps);
  [W2,B2] = rands(Lay2,Lay1);
  [W3,B3] = rands(Outps,Lay2);

  TP = [100, 10000, 0.01, 0.1, 1.05, 0.7, 0.9, 1.04];

  [W1,B1,W2,B2, W3, B3,TE,TR] = trainbpx(W1,B1,'tansig',W2,B2,'tansig',W3,B3,'purelin',indata,desoutposnn,TP);

  save resultpos W1 B1 W2 B2 W3 B3 indata desoutposnn

  c1 = tansig(W1*indata, B1);
  c2 = tansig(W2*c1, B2);

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%%%%%%                 PROGRAM 3          %%%%%%%
  %%%%%%%      This program trains the velocity      %%%%%%%
  %%%%%%%      neural network differentiator       %%%%%%%
  %%%%%%%                                    %%%%%%%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  testvel2
  disp('input finished');
```

```
inps=20;
Lay1=35;
Lay2=10;
Outps=1;

[W1,B1] = rands(Lay1,inps);
[W2,B2] = rands(Lay2,Lay1);
[W3,B3] = rands(Outps,Lay2);

TP = [500, 100000, 0.001, 0.1, 1.05, 0.7, 0.9, 1.04];

[W1,B1,W2,B2,W3, B3,TE,TR] = trainbpx(W1,B1,'tansig',W2,B2,'tansig',W3, B3, 'purelin', indata,desoutvel,TP);

save resultvel W1 B1 W2 B2 W3 B3

c1 = tansig(W1*indata, B1);
c2 = tansig(W2*c1, B2);
c3 = purelin(W3*c2, B3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%              PROGRAM 4              %%%%%%%
%%%%%%%     This program trains the acceleration %%%%%%%
%%%%%%%     neural network differentiator    %%%%%%%
%%%%%%%                                         %%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

testacc;
disp('input finished');
inps=20;
Lay1=35;
Lay2=10;
Outps=1;

[W1,B1] = rands(Lay1,inps);
[W2,B2] = rands(Lay2,Lay1);
[W3,B3] = rands(Outps,Lay2);


TP = [100, 60000, 0.001, 0.1, 1.05, 0.7, 0.9, 1.04];

[W1,B1,W2,B2,W3,B3,TE,TR] = trainbpx(W1,B1,'tansig',W2,B2,'tansig',W3,B3,'purelin',indata,desoutacc,TP);

save resultacc W1 B1 W2 B2 W3 B3

c1 = tansig(W1*indata, B1);
c2 = tansig(W2*c1, B2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%              PROGRAM 5              %%%%%%%
%%%%%%%     This program trains the jerk     %%%%%%%
%%%%%%%     neural network differentiator    %%%%%%%
%%%%%%%                                         %%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

testjrk;
disp('input finished');
inps=20;
Lay1=20;
Lay2 = 35;
Outps=1;

[W1,B1] = rands(Lay1,inps);
[W2,B2] = rands(Lay2,Lay1);
[W3,B3] = rands(Outps,Lay2);

TP = [100, 5000, 0.01, 0.1, 1.05, 0.7, 0.9, 1.04];

[W1,B1,W2,B2,W3,B3,TE,TR] = trainbpx(W1,B1,'tansig',W2,B2,'tansig',W3, B3, 'purelin',indata,desoutjrk,TP);

save nndiffjrk1 W1 B1 W2 B2 W3 B3

c1 = tansig(W1*indata, B1);
```

```
c2 = tansig(W2*c1, B2);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%                 PROGRAM 6                  %%%%%%%
%%%%%%%        This program performs the           %%%%%%%
%%%%%%%        digitization of the input signal    %%%%%%%
%%%%%%%                                            %%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [realno] = convert(number,resolution)

nombre=number;

if number<0
 number=-1*number;
end

temp1=fix(number);
number=number-temp1;

result=number*2;

bin=fix(result);

if bin==1
 result=result-1;
end

answer(1)=bin;

for i=2:resolution

result=result*2;
bin=fix(result);

if bin==1
result=result-1;
end

answer(i)=bin;

end


for i=1:resolution

if answer(i)==1
 temp(i)=2^(-1*i);
end

realno=sum(temp);

end

realno=temp1+realno;

if nombre < 0

realno=-1*realno;

end

number=0;
```