An Experiment in Neuro-Computed Torque Control of a Geared, DC Motor Driven Industrial Robot

DANIEL SCHMIDT and ANDREW P. PAPLIŃSKI
Computer Science and Software Engineering
Monash University, Clayton 3168, AUSTRALIA

Abstract: - Efficient control of robotic manipulators is an important industrial problem, and this paper proposes the use of artificial neural networks as a possible solution. The computed torque control scheme in particular is examined, and neural networks are used for the task of modelling the robotic manipulators. This scheme is applied to a two axis robot, whose feedback is corrupted by noise. The computation delay present in the system is subsequently accounted for by the use of neural network state predictors. The performance of the neuro-computed torque controllers is compared against a linear controller, and a comparison of other issues such as noise problems is included. Joint space and Cartesian space trajectory results demonstrate the superiority of the neural network-based nonlinear schemes over the linear controller.

Key-Words: - Robotics, Neural Networks, Computed Torque, Nonlinear Control, Noise Robustness

1 Introduction

The Computed Torque control method [13] is a well known and efficient algorithm for trajectory control of a robotic manipulator. In this method, the nonlinearities of the robot system are cancelled by a nonlinear compensator, and a standard linear control law is subsequently applied. This compensation is provided by what is effectively an inverse dynamics model of the robot in question.

As the inverse dynamics models are usually complex and computationally expensive, the standard implementation of a computed torque controller is on a digital computer of adequate power. The conventional approach to the creation of this type of controller is to first derive a continuous time model of the system by hand, and then apply a continuous-to-discrete time transformation to allow implementation on a digital computer [6].

Two main problems exist with this approach: firstly, the inverse dynamics model must be derived by hand and therefore there is the requirement for correct parameter estimation by the designer, based on mechanical models of the links. Secondly, the discretisation process may introduce errors into the model, especially if a simple algorithm, such as Euler's method, is used [1]. The proposed solution to these two problems utilizes artificial neural networks to learn the computed torque model from discrete data samples gathered from the robot. This approach avoids the explicit estimation of model parameters, and by-passes the problems of discretisation of a continous-time model using the discrete-time neural networks.

Modelling dynamic systems with neural networks has already been examined and Narendra [11] who proposed several input/output structures for representing unknown non-linear dynamic systems with neural networks. Narendra [12] has also proposed the application of the 'decoupling' model [5], which has already been successfully applied to linear systems, for neural network system representation. The decoupling model describes the outputs of the plant in terms of their relative degree.

Neural Networks have already been applied with success to forward modelling of robotic manipulators [4] as well as being used for payload compensation [7]. These experiments indicate that neural networks are a suitable choice of structure for modelling the inverse dynamics. Methods for compensating for computation delays present in discrete control systems involving state predictors have also been previously proposed [2, 9].

This article is organised as follows: in section 2, the neural-network model used in the computed torque scheme is considered. Section 3 covers the design of the controller, and also includes the proposed method for compensation of computation delays. Section 4 details results of controller track-

ing performance on a two-axis robot manipulator, as well as discussing other issues. Section 5 summarises the main results of this paper.

2 Nonlinear Compensator

The heart of the computed torque scheme is the non-linear compensator that is used to cancel the effects of the robot manipulator's non-linear dynamics.

Consider the standard torque equation describing an n-degree-of-freedom robot manipulator:

$$\tau = D(q)\ddot{q} + H(q,\dot{q}) + G(q) \qquad (1)$$

The first term represents inertial torques present in the system, the second term arises from interactions between multiple axis and the last term is the presence of gravity, and q is the position of the axis. This equation computes the torque exerted on the axis, given some acceleration, velocity and axis configuration [8]. Knowing that torque is a function of the power delivered to the motor (i.e. the motor command) we can make this simple substitution and get

$$F(C) = D(q)\ddot{q} + H(q,\dot{q}) + G(q) \qquad (2)$$

where C is the command to the motor amplifiers. Inverting $F()$ gives us:

$$C = F(D(q)\ddot{q} + H(q,\dot{q}) + G(q))^{-1} \qquad (3)$$

Therefore, the task is to learn the relationship between arm acceleration, velocity and position, and the input command given to the manipulator. This will allow us to ask the system what command is required to generate a desired acceleration, given the current configuration of the robot. The resulting feedback signals from the encoders were very noisy, containing many large spikes, due to some problems with electrical interface. As differentiation generally amplifies noise the resulting acceleration signals, produced by taking the first difference of the velocity, were extremely noisy and meant that training a model using acceleration as inputs would be very difficult. Filtering the feedback signals was obviously an option, but would either introduce extra dynamics in the case of an IIR type filter, or a transport delay in the case of an FIR type filter, both of which are highly undesirable.

The solution was to compute instead the command required to realise a particular velocity rather than an acceleration. The velocity signals were noisy but far from unusable, even without filtering. The non-linear compensation network can now be viewed as an open-loop velocity controller. Desired velocities are inputted, and the commands required to realise them are outputted. Though the form shown in equation 3 is continuous in nature, no explicit discretisation was required as the compensator was to be created by training a discrete time neural network. Thus the final form of the compensator is:

$$C[t] = N(q[t], \dot{q}[t], \ddot{q}[t]) \qquad (4)$$

where $N()$ is now the discrete time neural network used to model the inverse dynamics. In this paper the coupling effects present in the system will be ignored by the derived controllers.

2.1 Network Structure

This inverse system has its own dynamics, and therefore requires information on the state of the robot to compute the required compensating torque. A standard robotic manipulator takes motor commands as inputs and produces positions as outputs. Using several past values of positions as inputs to the inverse system will allow it to capture the current state of the manipulator, and because the gravitational nonlinearities of the system are dependent on the position, also allow the network to capture them by using these inputs. The order of the system determines how many past values are required to capture the dynamic behaviour exactly. This would be determined experimentally.

2.2 Network Training

The nonlinear compensator network was trained using a set of training samples obtained by exciting the manipulator in open loop mode. If the resulting inverse model is to accurately and fully model the system it is required that the training data used fully captures the dynamics of the axis and covers all operating points of the system. It has been shown that if the plant is of Nth order, then N sinusoids of different frequency will be sufficient to capture the dynamic behaviour of the plant [10].

Sinusoids were therefore used as probing signals. After two periods of each sinusoid signal, the amplitude, frequency and offset were changed and the axis was excited again. The random change

in offset effectively introduced step inputs, allowing any first order behaviour to be easily captured. Coverage of every single axis position was not necessary, as the approximating abilities of the neural network would allow for interpolation between the positions that have been captured. The range of operation was divided into eight regions, and the system was probed at each region. The networks were trained with the Levenberg-Marquadt algorithm, using a multi-start optimisation approach to overcome local minima.

3 Controller Design

3.1 Linear controller

The linear controller used in conjunction with the nonlinear compensator is a PD-type trajectory tracking controller with acceleration feedforward, taking desired positions, velocities and accelerations as inputs. The linear control law used was:

$$\nu[n] = K_p(q_d[n] - q[n]) + K_d(\dot{q}_d[n] - \dot{q}[n]) + \ddot{q}[n] \tag{5}$$

where $\nu[n]$ is the desired velocity for the next sample, which is then fed into the nonlinear compensation network, yielding the final control law:

$$C[n] = N(\nu[n], q[n], q[n-1], \ldots) \tag{6}$$

where $C[n]$ is the final command sent to the motor amplifiers. The gains were selected using the method described by Ishihara [6]. This however, assumes there is no modelling error and so the resulting gains were subsequently tuned to give minimum overshoot in all configurations, because this requirement is typical of a robotics application. The final structure of the neuro-computed torque controller is shown in Figure 1.

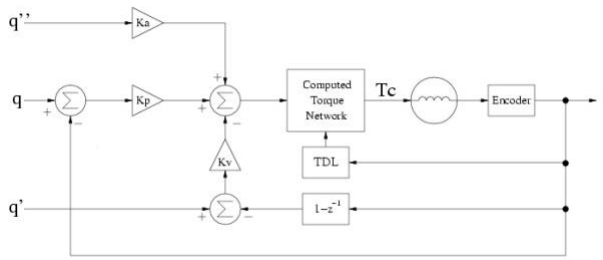

Figure 1: Computed Torque Controller

3.2 Accounting for computation delays

A computed torque controller uses the inverse model to compensate for the nonlinear dynamics of the robot. Even given a reasonably small implementation, such as the neural network one presented here, there is a finite computation delay present between the moment the position samples are read from the encoders and the control command is calculated. A naive implementation disregards this computation delay. This results in the position the robot is actually at when the command is finally issued not being the position that the controller used to make its decision. For linear discrete-time systems, it is well known that the effects of computation delay can be accounted for by the use of the state predictor. This technique effectively predicts the state of the system for the next sample. This prediction is then used by the controller to make its control decision, rather than on the feedback taken directly from the plant [9]. This idea would be extended to the nonlinear case by the implementation of a state predictor using neural networks.

The predictor must make an accurate prediction of where the axis will be at the time of the next sampling period, based on the current state of the axis and the previous command given to the axis. What is required then is a forward model of the system. The same training data generated to train the inverse models was used to train these forward models. The forward model took commands as inputs and produced velocities. These velocities could be accumulated to determine position. The forward model also required information on the current state of the axis in order to make its predictions, and to this end previous outputs of the axis were also used as inputs. As with the inverse model, the number of previous output samples used would determine the order of the model. Again, the forward model was trained using the parallel-series structure described by Narendra [11]. The final structure of the controller accounting for computation delays is shown in Figure 2.

4 Two-axis Robot case study

The techniques described previously were to be tested on a real robotic manipulator to determine their effectiveness. The tests were all performed on a Nakanishi Nak-280N SCARA robot (shown in

Figure 3), consisting of two revolute axis' chained together to allow the robot end-effector to realise any position in an x-y plane. For this experiment, the SCARA robot was oriented to operate in a plane perpendicular to the ground, thus introducing the nonlinear effects of gravity. In SCARA configuration robots the base axis is often called the shoulder, and the secondary axis the elbow, analogous to the human arm. Feedback is supplied by optical encoders, and both axis' are actuated by 48 volt, 2 amp permanent magnet D.C. motors. The sampling rate used in these experiments was 500 Hz.

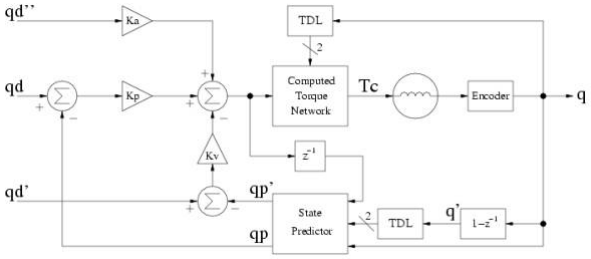

Figure 2: Computed Torque Controller with Neural Network State Predictor

Figure 3: The Nakanishi Nak-280N robot

The experimental procedure performed on this robot consisted of creating a linear PID-type controller, and neuro-computed torque controllers with and without compensation for computation delays. Performance of all three controllers would then be tested by attempting to track a Cartesian space trajectory. All controllers were tuned to give minimum overshoot to a step input, in all configurations.

4.1 Test conditions

The most important consideration at this point was that the feedback from the encoders was corrupted with a noise source of an unknown distribution. This feedback noise required the modification to the computed torque scheme as previously described. However, it also allowed the noise robustness of the neural network computed torque scheme to be tested.

4.2 Linear Controller

A linear P.I.D controller was developed. This type of controller is very representative of the controllers found on a large class of existing older robotic manipulators, and therefore would be useful for comparison purposes. This controller was designed by taking samples from the linear region of operation of the axis, deriving a linear model and applying Zieger-Nichols [3] tuning.

4.3 Compensation Networks

The structure of the compensation networks needed to be determined. This comprised two main sections: determination of the order of the networks, and the type and structure of neural network used to represent the system. The shoulder was modelled as a second order system, and the elbow as first order. Multi-layer cascade forward perceptron networks were used. These were arranged in three layers; two layers of non-linear sigmoid neurons and a final single, linear neuron in the output layer to provide scaling. The minimum number of neurons required to reasonably accurately approximate the function is the best choice, as overfitting generally leads to memorisation of noise and poor prediction. Two layers of one sigmoid neurons was found to be sufficient to learn the inverse dynamics. The networks were trained on data gathered from the axis in a manner previously described. Approximately 10,000 samples were gathered from the each axis and used in the training process.

4.4 Neural Network State Predictors

The proposed approach for overcoming computation delays in the system was through the use of

state predictors built from neural network forward models. The order of the shoulder and elbow forward models was chosen to be the same as the corresponding inverse models. Cascade-forward MLP networks with hidden layers of two sigmoid neurons were found to be sufficient to give good one-step-ahead prediction performance.

4.5 Implementation

The controller gains were selected as previously described. The controllers themselves were implemented in C++ on a digital computer attached to the Nakanishi robot.

4.6 Testing performance

A multi-link robotic manipulator's main task is to interact with the physical world, and as such the best way to test the controller would be to track a suitable cartesian space trajectory. The trajectory used traced a figure of eight (seen in Figure 4) in cartesian space in approximately 140 milliseconds. This speed trajectory required several samples of close to maximum acceleration and velocity to be realised, therefore testing the high frequency response of the controllers. The cartesian location of the figure-of-eight was chosen so as to require the joint co-ordinates to be in the nonlinear regions of operation.

The trajectories tracked by the controllers will be compared to the reference trajectory by using the mean-squared-error of the joint space co-ordinates and the mean-absolute-error in cartesian space. The cartesian space error was calculated as the Euclidean distance from the desired (x, y) position to the actual (x, y) position of the robot end-effector.

5 Discussion of Results

5.1 Tracking performance

Figure 4 shows the Cartesian paths generated by all three controllers. The linear controller is unable to respond with the correct magnitude force for half of the figure of eight and therefore is seen to lag badly. The computed torque controllers have managed to maintain the dimensionality of the figure-of-eight, that is the resulting trajectory is of similar dimensions but the symmetry is not perfect. The computation-delay compensated computed torque scheme performs better than both the other controllers. This is no doubt due to the fact that the gains of this controller could be higher than those of both other controllers. While all three controllers were tuned to give minimum overshoot, the predictors allowed the controller to look into the future and therefore be able to have reduced overshoot while maintaining the higher gains crucial to respond quickly to the fast changing input trajectory.

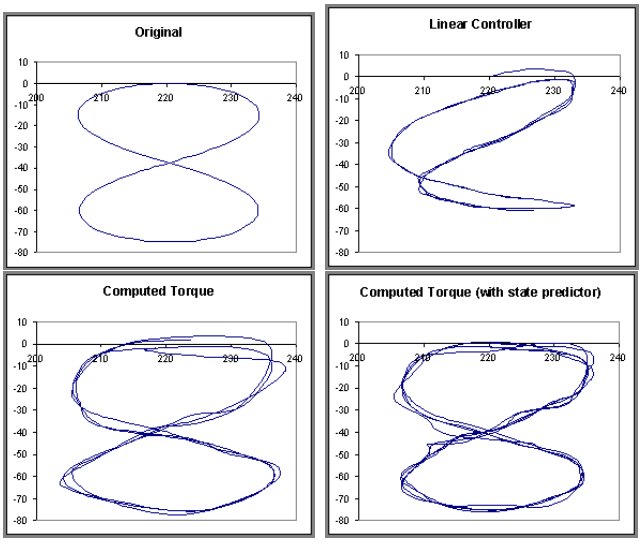

Figure 4: Cartesian tracking results. (a) Desired trajectory (b) Linear controller. MAE: 18.06 (c) Computed torque controller. MAE: 3.66 (d) Computed torque with predictor. MAE: 1.97

Table 1 summarises the joint space results, and it is clear that the computed torque with predictors also outperforms the other controllers in terms of joint-space MSE.

	Shoulder MSE	Elbow MSE
P.I.D	Mean: 6,530,000 SD: 45,460	Mean: 1,210,000 SD: 23,990
Computed Torque	Mean: 314,020 SD: 10060	Mean: 88,430 SD: 3340
Computed Torque (with Predictors)	Mean: 151,970 SD: 2380	Mean: 33,650 SD: 1880

Table 1: Joint space results

5.2 Noise Issues

The resulting controllers showed resistance to the feedback noise. This is most probably due to the nature of the models being trained. The inverse model

and the forward prediction models, while being dynamic, were only required to produce a forecast of one sample. That is, at every point in time, the inputs to the models came as measurements from the robot itself. Therefore, as the outputs of the models were never used as subsequent state feedback into the models themselves, any errors did not accumulate.

Additionally, given the fact that the noise only ever occurred on the velocity signals it is also possible to see why the noise did not significantly affect the training of the inverse model. The noise itself affected the measured velocity signals, introducing abnormally large spikes. The inverse model takes these corrupted velocities as inputs, and is told that normal commands generated them. A false relationship will be learnt, but given that such abnormally large velocities are never actually inputted to the inverse model once in the loop this false information is never actually retrieved.

6 Conclusion

During this project the problem of efficient control of robotic manipulators was considered. Artificial neural networks were proposed as a possible solution to this problem, and were subsequently utilised in an adaptation of the well known computed torque scheme. Results from experiments performed on a two-axis robot show good performance increases over linear controller schemes and additionally do not require large amounts of computation power to realise. It was also seen that the processes required to create the neuro-computed torque controller were robust to noise. The issue of computation delay was also considered, and solved with use of use of non-linear state-prediction built using neural networks.

Throughout the design of the neuro controllers, there was no requirement for any of the models used to be derived by hand. The neural network controllers in this project were designed with no explicit derivation of models or estimation of parameters, by utilising the learning ability of the neural network structure. This work indicates that neural-network based Computed Torque controllers may be able to efficiently control higher order manipulators with little requirement for intimate knowledge of their dynamics.

References:

[1] Conte, S.D., De Boor, C., *Elementary Numerical Analysis*, MacGraw-Hill, 1965

[2] Barratt, C., Boyd, S., Examples of exact tradeoffs in linear controller design, *IEEE Control Systems Magazine*, 9(1), pp. 115-143, 1987

[3] Dutton, K., Thompson, S., Barraclough, B., *The Art of Control Engineering*, Addison-Wesley, 1997

[4] Eskandarian, A., Bedewi, N.E., Kramer, B.M. Modelling of Robotic Manipulators Using an Artificial Neural Network, *Journal of Robotic Systems*, Vol. 11, No. 1, 1994, pp. 41-52

[5] Falb, P.L., Wolowich, W.A., Decoupling in the design and synthesis of multi-variable control systems, *IEEE Transactions on Automatic Control*, Vol. 12, 1967, pp. 651-659

[6] Ishihara, T., Direct Digital Design of Computed Torque Controllers, *Journal of Robotic Systems*, pp. 197-209, 1994

[7] Leahy, M.B, Johnson, M.A., Rogers, S.K. Neural Network Payload Estimation for Adaptive Robot Control, *IEEE Transactions on Neural Networks*, Vol. 2, No. 1, 1991, pp. 93-100

[8] McKerrow, P.J., *Introduction to Robotics*, Addison-Wesley Publishing Company, 1991

[9] Mita, T., Optimal Digital feedback control systems counting for computation time of control laws, *IEEE Trans. Automat. Control*, 1985, pp. 542-548

[10] Narendra, K.S, Adaptive Control Using Neural Networks, *Neural Networks for Control*, 1991, pp. 115-142

[11] Narendra, K.S, Parthasary, K. Identification and Control of Dynamical Systems Using Neural Networks, *IEEE Transactions on Neural Networks*, Vol. 1, No. 1, 1990, pp. 4-26

[12] Narendra, K.S, Mukhopadhyas, S. *Adaptive Control of Nonlinear Multivariable Systems using Neural Networks Neural Networks*, Vol. 7, No. 5, 1994, pp. 737-752

[13] Spong, W.M, Vidyasagar, M. *Robot Dynamics and Control*, John Wiley and Sons, 1989