Real-time implementation of a dynamic fuzzy neural networks controller for a SCARA

Meng Joo Er*, Chang Boon Low, Khuan Holm Nah, Moo Heng Lim, Shee Yong Ng

School of Electrical and Electronic Engineering, Nanyang Technological University, Block S1 Nanyang Avenue, Singapore, Singapore 639798

Received 29 July 2001; revised 10 June 2002; accepted 22 August 2002

Abstract

This paper presents the design, development and implementation of a Dynamic Fuzzy Neural Networks (D-FNNs) Controller suitable for real-time industrial applications. The unique feature of the D-FNNs controller is that it has dynamic self-organising structure, fast learning speed, good generalisation and flexibility in learning. The approach of rapid prototyping is employed to implement the D-FNNs controller with a view of controlling a Selectively Compliance Assembly Robot Arm (SCARA) in real time. Simulink, an iterative software for simulating dynamic systems, is used for modelling, simulation and analysis of the dynamic system. The D-FNNs controller was implemented through Real-Time Workshop (RTW). RTW generates C-codes from the Simulink block diagrams and in turn, the generated codes (object codes) are downloaded to the dSPACE DS1102 floating-point processor, together with the supporting files, for execution. The performance of the D-FNNs controller was found to be superior and it matches favourably with the simulation results.

© 2002 Elsevier Science B.V. All rights reserved.

Keywords: Algorithms for real-time control; Adaptive control; Fuzzy systems; Neural network systems; Manipulators

1. Introduction

The promise of robots being able to replace menial, tedious, or dangerous human tasks has motivated many engineers to design and develop more sophisticated and intelligent controllers for robotics. Numerous intelligent control methodologies have been devised to solve a number of complicated problems in robot manipulators, which are represented by a multivariable non-linear coupled dynamic system with uncertainties [Refs. 1–5]. Due to uncertainties, it is difficult to obtain an accurate mathematical model for robot manipulators. Conventional control methodologies find it difficult or impossible to handle unmodelled dynamics of a robot manipulator. To accommodate system uncertainties and variations, learning or adaptive techniques must be incorporated.

Most conventional methods, e.g. PID controllers, are based on mathematical and statistical procedures for modelling the system and estimation of optimal controller parameters. In practice, the plant to be controlled is often highly non-linear and a mathematical model may be difficult to derive. As such, conventional techniques will not be able to handle modelling errors and are lack of robustness.

Fuzzy logic, however, offers a promising approach to robot controller design. In contrast with conventional controller design techniques, fuzzy logic formulates control of a plant in terms of linguistic rules drawn from the behaviour of a ‘human operator’ instead of an algorithm synthesised from a rigorous mathematical model of the plant.

Due to the growing popularity of Neural Networks, much research effort has been directed towards design of intelligent hybrid controllers using Fuzzy Logic and Neural Networks. Design of robust adaptive controllers suitable for real-time control of robot manipulators is one of the most challenging tasks for many control engineers. Robot manipulators are multivariable non-linear coupled systems and are frequently subjected to structured and/or unstructured uncertainties even in a well-structured setting for industrial use. The Dynamic Fuzzy Neural Networks (D-FNNs) algorithm used is a newly developed algorithm that has the following salient features: dynamic
self-organising structure, fast learning speed, good generalisation and flexibility in learning.

The D-FNNs controller is employed to compensate for environmental variations such as payload mass and disturbance torque during the operation process. By virtue of on-line learning, it is able to learn the robot dynamics and make control decisions simultaneously. In effect, it offers an exciting on-line estimation scheme of the plant.

The approach of Rapid Prototyping is used to implement the D-FNNs controller in real-time through Real-Time Workshop (RTW) using the Texas Instruments TMS320C31 floating-point processor, together with some supporting files. In addition, Simulink is used for modelling, simulation and analysis of the dynamic system.

2. Overview of the seiko TT-3000 SCARA

2.1. Background on the TT-3000 SCARA

The SEIKO D-TRAN 3000 Series robot used in this work is a four-axis, closed-loop DC servo Selectively Compliance Assembly Robot Arm (SCARA). Each of the four axes provides a different motion and contributes to one degree of freedom of the robot (see Fig. 1). The main characteristics of the TT-3000 SCARA are its high precision, repeatability and speed. The basic SCARA geometry is realised by arranging two revolute joints (for simplicity, they will be called Joint T1 and Joint T2 joint herein) and one prismatic joint (for simplicity, it will be called Joint Z herein) in such a way that all axes of motion are parallel. The acronym SCARA characterises mechanical features of a structure offering high stiffness to vertical loads and compliance to horizontal loads.

2.2. The robot dynamic model

The dynamic equations of the SCARA can be represented by a set of highly non-linear coupled differential equations given by

\[ M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + G(\theta) + F_V(\dot{\theta}) + F_C = \tau \]  

(2.1)

where \( M(\theta) \) is the \( n \times n \) inertia matrix of the manipulator, \( C(\theta, \dot{\theta}) \) is the \( n \times n \) matrix of centrifugal and Coriolis terms, \( G(\theta) \) is an \( n \times 1 \) vector of gravity terms, \( F_V(\dot{\theta}) \) is the \( n \times 1 \) vector of viscous friction terms, \( F_C \) is the \( n \times 1 \) vector of coulomb terms and \( \tau \) is the \( n \times 1 \) vector of the input torque (generated by the joint motor). Each element of \( M(\theta) \) and \( G(\theta) \) is a complicated function which depends on angular positions of all joints of the manipulator, \( \theta \). Similarly, each element of \( C(\theta, \dot{\theta}) \) is a complicated function of both \( \theta \) and \( \dot{\theta} \). The terms \( \theta, \dot{\theta}, \ddot{\theta} \) are the \( n \times 1 \) vectors of the output link position, velocity and acceleration, respectively.

The movement of the end-effector to track a desired trajectory, at a particular velocity, thus requires a complex set of torque functions to be applied to the actuating system of the robot at each joint. The dynamic model of the TT-3000 SCARA has been developed in Ref. [6], with most of its parameters determined and verified through experiments. Based on this known mathematical model of the robot, training of the D-FNNs controller was carried out using MATLAB simulation tools.

2.3. Motion control

A two-joint control structure is used in this work where motion control of Joint T1 and Joint T2 will be executed simultaneously. In this scheme, Joint Z will not be controlled for simplicity. Motion control of the robot arm is shown in Fig. 2.

The control system of the two joints is considered a Multiple-Input Multiple-Output (MIMO) system. The degree of difficulty will increase if the number of joints to be controlled simultaneously increases. The implementation
of this control architecture on the target system is shown in Fig. 3.

### 3. Design of the D-FNNS controller

#### 3.1. Architecture of D-FNNS

The architecture of the D-FNNS developed in Ref. [7] is depicted in Fig. 4. It is constructed based on the Radial Basis Function (RBF) and functionally; it is equivalent to an TSK model-based fuzzy system.

In Fig. 4, Layer 1 defines the input variables layer. This is the layer where the input signals first enter the D-FNNS.

Layer 2 represents the membership functions associated with the input variables. The membership function is chosen as a Gaussian function of the following form:

\[
MF_{ij}(x_i) = \exp \left[ -\frac{(x_i - c_{ij})^2}{\sigma_j^2} \right]
\]  

(3.1)

\(i = 1, 2, 3...r, \quad j = 1, 2, 3...u\)

Layer 3 is the rule layer. The number of RBF units in this layer indicates the number of fuzzy rules. The outputs are given by

\[
\phi_j = \prod_{i=1}^{r} MF_{ij}
\]  

(3.2)

Layer 4 defines the output variables. Each output variable is the weighted sum of incoming signals. Corresponding to a fuzzy system, this layer performs defuzzification that considers effects of all membership functions of the input values on the output, i.e.

\[
y = \sum_{j=1}^{u} \phi_j w_j
\]  

(3.3)

The weight is chosen as follows in the TSK model:

\[
w_j = \alpha_{0j} + \alpha_{1j} x_1 + \cdots + \alpha_{rj} x_r
\]  

(3.4)

where \(\alpha_r\)'s are real-valued parameters. It is not difficult to see that Eq. (3.3) can be re-written as

\[
y = W\phi
\]  

(3.5)

where

\[
W = \begin{bmatrix}
\alpha_{01} & \alpha_{11} & \cdots & \alpha_{r1} & \alpha_{02} & \alpha_{12} & \cdots & \alpha_{rn} \\
\cdots & \alpha_{02} & \cdots & \alpha_{rn} & \cdots & \alpha_{03} & \cdots & \alpha_{rn} \\
\end{bmatrix}
\]  

(3.6)

\[
\phi = \begin{bmatrix}
\phi_1 & \phi_1 x_1 & \cdots & \phi_1 x_r & \phi_2 & \phi_2 x_1 & \cdots & \phi_2 x_r \\
\cdots & \phi_u x_1 & \cdots & \phi_u x_r
\end{bmatrix}^T
\]  

(3.7)

#### 3.2. Adaptive fuzzy neural control scheme

The proposed adaptive fuzzy neural control scheme is depicted in Fig. 5.

The basic idea of this approach is to learn the manipulator’s inverse characteristics and to use the inverse dynamic model to generate the compensated control signal. There are two Fuzzy Neural Networks (FNNs) used in this...
A control system. FNN A is employed for weights training of the system whereas FNN B is used to generate the appropriate control signal with the trained weight structure from FNN A. The proposed control algorithm is given by

\[ \tau = \tau_{PD} + \tau_{FNNB} \]  

(3.8)

where \( \tau \) is the required control torque, \( \tau_{PD} \) is the torque generated by the PD controller and \( \tau_{FNNB} \) is the torque generated by FNN B.

The inverse robot model is obtained by FNN A, employing the D-FNNs learning algorithm of Ref. [7]. In online learning, FNN A is trained during real-time control of the manipulator. FNN B is a duplicate copy of FNN A, but its weights will be further adjusted by the error signal \( \tau_{PD} \) as the controller is in operation. This is to compensate for modelling errors and unmodelled disturbances. The parallel-connected conventional PD controller also caters for faster and more accurate tracking performance.

3.3. Weight training algorithm

The gist of weight training is to train FNN B such that the squared error between the desired torque and the actual torque, i.e.

\[ E = \frac{1}{2}(\tau - \tau_{FNNB})^2 = \frac{1}{2}(\tau_{PD})^2 \]  

(3.9)

is minimised.

To achieve fast weight adjustment, the gradient method is used to minimise \( E \). The weight is adjusted as follows:

\[ \Delta W = W(\text{new}) - W(\text{old}) = -\eta \frac{\partial E}{\partial W} \]  

(3.10)

where \( \eta > 0 \) is the learning rate.

Using the chain rule, we have

\[ \frac{\partial E}{\partial W} = \frac{\partial E}{\partial \tau_{FNNB}} \frac{\partial \tau_{FNNB}}{\partial W} \]  

(3.11)

It follows from Eqs. (3.5) and (3.9) that when \( y = \tau_{FNNB} \), we have

\[ \frac{\partial E}{\partial \tau_{FNNB}} = -(\tau - \tau_{FNNB}) \]  

(3.12)

\[ \frac{\partial \tau_{FNNB}}{\partial W} = \Phi \]  

(3.13)

Thus, Eq. (3.10) can be rewritten as

\[ \Delta W = \eta(\tau - \tau_{FNNB})\Phi = \eta \tau_{PD}\Phi. \]  

(3.14)

4. Simulation studies

4.1. Rapid prototyping

The simulation program for the D-FNNs is written in MATLAB and Simulink to facilitate Rapid Prototyping, a process that allows one to conceptualise solutions using block diagram modelling environment and have a preview of system performance prior to laying out hardware, writing any production software, or committing to a fixed design. Fig. 6 shows the Rapid Prototyping Process (RPP) in details.

Here, the RPP is used to design and develop the D-FNNs controller via Matlab Simulink, RTW, Real-Time Interface (RTI) and dSPACE DS1102 floating-point processor. Firstly, all user-defined function blocks are written in C code for C-Mex Sfunctions [8]. In this implementation, FNN A and FNN B are implemented in C-Mex Functions. This hand-written C code ensures the compatibility in the final downloading phase of the RPP. Next, a Simulink model of the D-FNNs controller is constructed to test any errors that might be caused by the user-defined C-Mex Sfunctions. Simulink provides an Application Programming Interface (API) that allows users to troubleshoot user-defined C-Mex Sfunctions in the Simulink environment. API interfaces between Simulink and Microsoft Visual C++, providing programmers a debugging tool for effective debugging [9]. As soon as the desired simulation result is obtained, the next step is to construct the D-FNNs controller model using Simulink. Since the dSPACE processor can be easily interfaced with Simulink, a build function in Simulink will automatically convert the Simulink model into a targeted C code and download it to
the designated hardware (dSPACE DS1102) via RTI [10]. Finally, to read or write the internal variables of the control system, dSPACE ControlDesk [11] provides a user-friendly Graphic User Interface (GUI) environment that enables the user to observe vital data in the system.

4.2. Simulation model of D-FNNs controller

As foreshadowed earlier, the RPP begins in Simulink. The first step in the design is to develop the SCARA model prior to the development of algorithms in Simulink. Once a simulation model, which can capture the dynamics of the robot with sufficient accuracy, has been developed, the RPP continues by using Simulink to develop the algorithm and analyse the results. If the results are not satisfactory, the modelling/analysis process is iterated until satisfactory results are achieved. Fig. 7 depicts the interconnection of the D-FNNs controller and the robot model.

Fig. 8 shows the simulation model of the D-FNNs control system for Joint T1 and Joint T2 of the SCARA. Using this model, iterative tuning is performed on the input and parameters of the D-FNNs controller. The aim is to achieve zero overshoot within an acceptable steady-state error of less than 2% in response to a step input. To ensure compatibility with Simulink’s RTW and real-time implementation, the D-FNNs controller needs to be written in C-Mex Sfunctions.

4.3. Simulation results

To evaluate the performance of the proposed D-FNNs controller, a series of simulation results were performed. Figs. 9–12 depict trajectory tracking for Joints T1 and T2. Between time \( t = 0 \) and \( 0.2 \) s, the maximum trajectory tracking error for Joint T1 is 0.025 rad. When \( t > 0.2 \), the tracking error decreases from a maximum value of 0.025 rad to a small value of \( \sim 0.01 \) rad. Likewise, for Joint T2, between time \( t = 0 \) to 0.5 s, it has a maximum tracking error of 0.5 rad. When \( t > 0.5 \) s, the error reduces to a maximum value of \( \sim 0.019 \) rad. The reason for these observations is that the D-FNNs learning algorithm is still learning the inverse model of the robot manipulator. Therefore, FNN B may not have sufficient number of neurons to compensate
the control signal at the initial stage. Besides, the initial position of Joint T2 at \( t = 0 \) is 0 rad. As a result, there are significant errors in the initial positions. When \( t \) approaches infinity, the steady-state error of each joint reduces to a low level of \( \pm 1.14\% \) for Joint T1 and \( \pm 1.8\% \) for Joint T2. These results clearly indicate that the D-FNNs has successfully constructed membership functions to model the SCARA and weight training of neurons is able to reduce the tracking error effectively.

5. Implementation of the D-FNNS controller

5.1. System set-up

Fig. 13 shows an overview of the system set-up. A DS1102 DSP controller board [12] sits in the 16-bit ISA slot on the PC motherboard. The DS1102 contains a Texas Instrument TMS320C31 floating-point Digital Signal Processor, running at 60MHz with 128 KiloWords (KW) of SRAM. The Connector Panel CP1102 provides connection between the DS1102 DSP controller board and the interface board. The interface board contains signals like control signals for the various axes of the robot arm, the encoder count signal, signals of the overrun limit switches and the power supply. The separate PWM DC servomotor amplifiers are used to control the respective joints movement of the SCARA. An external power supply unit supplies the 24 V DC needed by the PWM DC servomotor amplifier and the 5 V DC needed for the interface board.

5.2. Hardware constraints

There are some restrictions that need to be considered during implementation of the controller on the dSPACE platform. Since the targeted hardware is a DSP platform (dSPACE DS1102 card), it is essential to understand the limitations and restrictions of the hardware. Firstly, there is limited time for the dSPACE processor to execute the entire D-FNNs controller. When more membership functions (neurons) are formed to model the inverse dynamics of the robot, the size of matrices increases. As a result,
the computation time for each iteration increases. When the computation time required for each iteration exceeds the sampling interval, a 'Task Overrun' error occurs, causing the processor to halt [13]. The execution time per C instruction is 33.33 ns. Furthermore, the total memory available in DS1102 is 128 KW, which includes the C program code generated from RTW, the memory declaration in the entire D-FNNs controller model, and the stack memory required for the processor [12]. Therefore, C codes of Sfunctions and the D-FNNs controller model for RTW must be optimised.

5.3. C-Mex Sfunctions

Since the D-FNNs algorithm is well defined, a Linear Process model is chosen for this software process [14]. A careful risk analysis is conducted on the D-FNNs algorithm to reduce the risk of hardware incompatibility in the final phase of its implementation. After the analysis, it was found that the number of input data pairs required for weight adjustment function in the D-FNNs learning algorithm must not exceed an array size of $2 \times 75$. This maximum size is an estimation made with reference to the maximum memory space of the dSPACE DS1102. Besides, some C programming statements are prohibited in the dSPACE platform during hardware implementation. For example, `Malloc()` function takes up a long execution, causing a 'task overrun' error [15]. To speed up the program computation time, inline functions may be used [13]. However, it increases the program size significantly. This increment in the program size may exceed the available memory space of DS1102 and prohibit the D-FNNs controller model from being downloaded into the dSPACE processor for executions. Thus, a compromise between these two critical requirements needs to be achieved.

D-FNNs learning Sfunction (FNN A) is divided into several sub-functions to improve its maintainability and reusability. These sub-functions are developed and tested individually using White-box and Black-box testing techniques. Software technical review is also conducted at every stage to keep errors to a minimum level so as to prevent amplifications of these errors in later stages.

In the coding phase, two compiler are used in this implementation. The first compiler is Borland C++ 5.0, and the other is the Matlab Mex compiler. Borland C++ 5.0 is used to program C sub-functions of D-FNNs Sfunctions. These sub-functions will be transferred to C-Mex Sfunction C code template file for final compilation by the Mex compiler after all necessary tests of these codes are completed. Since the Mex compiler does not have any debugging tools, it is quite difficult to develop the C code using only the Mex compiler.

5.4. D-FNNs software

The D-FNNs software consists of primarily two modules: (1) The main control module, and (2) The GUI for signal monitoring and parameters tuning. Module (1) is written in Simulink whereas Module (2) is developed using ControlDesk.

5.4.1. Main control module

The main control module handles sampling of the joint’s count information and also computes the joint’s position error and acceleration. The results are then passed to the D-FNNs controller to control the SCARA. This module also manages the set-up of the I/O hardware and data transfer to the interface board.

To implement the D-FNNs controller in real-time, it is essential that the controller Simulink model for RTW be built so that it can be downloaded into the DS1102 card via RTI. In this RTW model, a set of I/O devices is created in the Simulink library to provide an interface between the downloaded Simulink model and the SCARA. This set of I/O device drivers is automatically embedded into the Simulink library once the dSPACE card is properly installed. Once the Simulink model for RTW is constructed, the downloading process is done automatically by clicking the ‘build’ button in the RTW menu with some necessary settings [16]. In this phase, the entire controller Simulink model is compiled into a single C code for the targeted hardware, DS1102. Next, RTW will check the total memory required for the controller model. If the C code of the model requires more than 128 KW of memory space, RTW will stop the entire automatic downloading process.

5.4.2. Tuning of GUI parameters and signal monitoring

The second module provides an efficient means for analysing and monitoring trajectories of the joints during real-time operation of the robot arm. There are two ways of changing model parameters while the model is running on the target processor:

1. Using Simulink’s external mode.
2. Using dSPACE’s ControlDesk software.
In this work, dSPACE ControlDesk is used to implement parameters tuning, signal monitoring and GUI. ControlDesk (dSPACE’s experiment software) provides all the functions for controlling, monitoring and automation of real-time experiments and makes the development of controllers much more effective.

The dSPACE real-time hardware is managed by the Hardware Manager integrated in ControlDesk. In this implementation, the dSPACE ControlDesk is used to observe real-time results and to change the input parameters online. The Object and TRC files (generated by RTI when performing a RTW build) are loaded in order to run real-time applications.

Fig. 14 shows the various signals of the SCARA robot being monitored in real time by GUI. During the ‘RUN’ mode, the robot behaviour can be monitored via GUI. Values like desired positions, actual positions and errors of both Joint T1 and Joint T2 can be displayed in both graphical and numerical format.

Real-time changes can be made to the input values of Joint T1 and Joint T2 by adjusting the knobs as shown in Fig. 14, which control the movement of the SCARA. Pressing the ‘HOME’ button will instruct the SCARA to return to its home position, while the ‘RUN’ button will instruct the robot arm to move. FNN B can also be enabled/disabled online to see its effect on the robot performance. Besides the ability of making real-time changes, the ControlDesk also enables viewing of real-time changes/performances, such as angular position of Joint T1 and its error. In order to complement the ControlDesk, the dSPACE Trace [17] Module is used to capture and perform a detailed analysis of the waveforms.
6. Experimental results

Experimental studies were carried out to evaluate the performance of the D-FNNs controller. They were carried out in three parts. The first part is to evaluate the trajectory tracking capability of the controller without external disturbances. The second evaluation is to test the robustness of the D-FNNs controller by injecting a disturbance in the circular motion plane. This disturbance is injected by hooking a spring on the SCARA and pulling it in the motion plane. The third test is to compare the performance of the controller with a conventional Proportional Derivative (PD) controller.

6.1. Experimental results (without disturbances)

Figs. 15–18 depict trajectory tracking in the absence of external disturbances at a sampling time of 1 ms. At $t = 4$ s, reference trajectories are fed to the D-FNNs control system. It can be observed that the error of Joint T1 increased to a value of $0.13$ rad at $t = 4$ s. This is due to the difference between its initial position and the set point. In contrast, Joint T2 has a smaller tracking error at this instance since its initial position is closer to the initial set point. When $t > 4$ s, the D-FNNs algorithm starts to learn the inverse model of the robot, and fine tune FNNB accordingly. The compensated torque $\tau_{\text{FNNB}}$ successfully reduces the trajectory tracking error to a low level of $\pm 0.015$ rad for Joint T1 and $\pm 0.0125$ rad for Joint T2.

Figs. 19–22 show the responses in the absence of external disturbances for a sampling time of 4 ms. It is evident that the actual trajectories are able to follow the desired trajectories closely.

6.2. Experimental results (with disturbances)

Figs. 23–26 show the responses when an external disturbance is injected into the motion plane for a sampling time of 1 ms. The results obtained show that even when there are external disturbances, the D-FNNs controller is able to reduce the trajectory tracking error to a low level of...
0.015 rad for Joint T1 and 0.0125 rad for Joint T2. At \( t = 7 \) s, the control action begins. Initially, the tracking error is high because of the robot’s initial position. However, by virtue of the D-FNNs online learning and weight training, tracking errors for both joints are reduced to a low level very quickly. Fig. 23 shows that there is a slight deviation when the set point is at 40\(^\circ\). This is because the total output torque from the D-FNNs controller is saturated.

Figs. 27–30 show the responses under external disturbances for a sampling time of 4 ms. Again, it is evident that actual trajectories are able to follow the desired trajectories closely.

6.3. Experimental results (with and without D-FNNs controllers)

Figs. 31–34 demonstrate tracking capabilities of the control system with and without the D-FNNs controller.
Fig. 26. Trajectory tracking error for Joint T2 (sampling time = 1 ms).

Fig. 27. Trajectory tracking for Joint T1 (sampling time = 4 ms).

Fig. 28. Trajectory tracking error for Joint T1 (sampling time = 4 ms).

Fig. 29. Trajectory tracking for Joint T2 (sampling time = 4 ms).

Fig. 30. Trajectory tracking error for Joint T2 (sampling time = 4 ms).

Fig. 31. Trajectory tracking for Joint T1.
at a sampling time of 1 ms. This test started off without the D-FNNs controller by disabling FNN B, i.e. having only the PD controller controlling the robot and FNN A learning the inverse model of the SCARA. As a result, tracking errors of Joint T1 and Joint T2 have a maximum value of 0.04 rad. When FNNB is enabled at $t = 35$ s, online weight training is used to tune the learned D-FNNs and compensate for tracking errors. Subsequently, the robot’s joint angles converge to their desired set points quickly once the FNNB is enabled.

It is evident from the preceding results that the D-FNNs learning algorithm is able to learn the inverse model quickly and track the reference trajectories accurately. In addition, the D-FNNs controller is robust in the presence of disturbances.

7. Conclusions

A D-FNNs Controller suitable for real-time industrial applications has been successfully designed, developed and implemented in this paper. The approach of RPP was adopted to shorten the development time, thus reducing the cost of development. Experimental results show that the proposed controller is able to learn the inverse dynamics of the SCARA quickly and reduce the tracking error to zero. The performance of the D-FNNs controller was found to be very good and robust in the presence of external disturbances and it matches favourably with the simulation results. Besides, parameters can be modified in real time and actual trajectories can be monitored via the GUI. This facilitates testing under different input conditions. We are confident that the proposed D-FNNs controller will be very useful in many other real-time industrial applications.

References


**Meng Joo Er** received the BE and ME degrees in Electrical Engineering from the National University of Singapore, and the PhD degree in Systems Engineering from the Australian National University, Canberra in 1985, 1988 and 1992, respectively. He is currently an Associate Professor in the School of Electrical and Electronic Engineering (EEE), Nanyang Technological University (NTU). He has authored numerous published works in his research areas of interest, which include control theory and applications, robotics and automation, fuzzy logic and neural networks, artificial intelligence, biomedical engineering, parallel computing, power electronics and drives and digital signal processors applications. Dr Er was the recipient of the Institution of Engineers, Singapore (IES) Prestigious Publication (Application) Award in 1996 and the recipient of IES Prestigious Publication (Theory) Award in 2001. He received the Teacher of the Year Award for the School of EEE for his excellent performance in teaching in 1999. He was also a recipient of a Commonwealth Fellowship tenable at University of Strathclyde from 14 February to 11 October, 2000.

**Chang Boon Low** is currently an undergraduate (majoring in Control) in the School of Electrical and Electronics Engineering, Nanyang Technological University (NTU). He received his Diploma (with Merit) in Electronics and Computer Engineering from Singapore Polytechnic in 1997. Low was the 1st and 2nd runner up winner in Singapore Robotics Games 97. He was also awarded the Motorola Award for meritorious academic performance in Singapore Polytechnic in 1997. During his academic days in SP and NTU, his projects were selected to participate in the Commex 96/97 exhibition held in SP, the Technopreneurship Competition 2000 and the College of Engineering Technology Exhibition 2002 held in NTU, respectively.

**Khuan Holm Nah** is currently an undergraduate (majoring in Communications) in the School of Electrical and Electronic Engineering, Nanyang Technological University (NTU). He received the Diploma in Electronics, Computer and Communication Engineering from the Singapore Polytechnic (SP) in 1996. During his academic days in SP and NTU, his projects were selected to participate in the Commex 95/96 exhibition, held in SP, the Technopreneurship Competition 2000 and the College of Engineering Technology Exhibition 2002 held in NTU, respectively.

**Moo Heng Lim** received the BE degree in Electrical and Electronic Engineering with Second Class Lower Division from Nanyang Technological University (NTU) in 2001. He is currently an Air Force Officer in the Republic of Singapore Air Force.

**Shee Yong Ng** received the BE degree in Electrical and Electronic Engineering with Second Class Lower Division from Nanyang Technological University (NTU) in 2001. He is currently a Software Engineer in the Systems Services Division of Singapore Engineering Software Pte Ltd.