14

An Improved Adaptive Kinematics Jacobian Trajectory Tracking of a Serial Robot Passing Through Singular Configurations

Ali T. Hasan[1], Hayder M.A.A. Al-Assadi[2] and Ahmad Azlan Mat Isa[2]
[1]*Mechanical & Manufacturing Engineering Department, Faculty of Engineering, University Putra Malaysia 43400 UPM Serdang, Selangor*
[2]*Faculty of Mechanical Engineering, University Technology MARA Shah Alam, 40450 Malaysia*

1. Introduction

In real time applications, the trajectory which has to be followed and the task that has to be performed during motion planning of multi-axis non-linear mechanical systems, such as robot manipulators are of great importance. Due to the non-linear transformation between the task space and the joint space coordinates, singularities and uncertainties in the arm configuration occur, the unplanned occurrence of such problems drive the end-effector out of the desired path which may cause collision of the robot arm with objects located in its work cell (Köker, 2005; Antonelli et al., 2003).

Depending on different tasks operation requirements and circumstances, motion control algorithms can be developed either at the *kinematics* level or at the *dynamic* level (Graca & Gu, 1993; Karilk & Aydin, 2000). To develop a dynamic control algorithm, torque limits of the joint actuators are to be handled, two typical approaches were introduced which are the Computed-torque and Resolved-acceleration approach, both approaches are based on the inverse dynamic model of the robot system (Asada & Soltin,1986; Sopng & Vinyasagar,1998; Faiz & Agrawal ,2000). A problem with these algorithms is the remarkable computational load required to handle the dynamics of a full-sized manipulator, which is seldom affordable by current industrial control units. In addition, implementation of torque-based control laws requires replacement of the low-level joint servos typically available in industrial robots with custom control loops.

Aimed at overcoming the above drawbacks, a different approach to path tracking based on the kinematics control was proposed. In detail, kinematics control consists in an inverse kinematics transformation which sends to the joint servos the reference values corresponding to an assigned end-effector trajectory; as a first advantage, this allows simple interfacing with the standard control architecture of industrial robots. In the framework of kinematics-based methods for path tracking, the counterpart of the physically meaning joint torque limits is played by acceleration constraints and the use of full dynamic models can be avoided; this typically leads to computationally light algorithms that allow real-time implementation on standard numerical hardware even for robot arms of many Degrees of

Freedom (DOF). A further advantage of kinematics control methods is the possibility of exploiting the presence of redundant (DOF) (Antonelli et al., 2003).
A considerable research effort has been devoted to solve the Inverse Kinematics problem in past years (Yang, 1969; Duffy & Rooney, 1975; Albala & Angeles, 1979; Tsai & Morgan, 1985; Daniel & Raul, 2003). Even though, Closed-form analytical solutions can only be found for manipulators having simple geometric structures (Antonelli et al., 2003; Karilk & Aydin, 2000). A number of algorithmic techniques mainly based on inversion of the mapping established between the joint space and the task space of the manipulator's Jacobian matrix have been proposed for those structures that cannot be solved in closed form.
The Resolved Motion Rate-Control technique was the first work in this field (Whitney,1969), in this technique the pseudoinverse of the Jacobian matrix is used to obtain the joint velocities corresponding to a given end-effector velocity, a major drawback of this method was the singularity problem. The use of a damped least-squares inverse of the Jacobian matrix has been later proposed in lieu of the pseudoinverse to overcome the problem of kinematics singularities (Nakamura & Hanafusa, 1986; Wampler, 1986).
Since in the above algorithmic methods the joint angles are obtained by numerical integration of the joint velocities, these and other related techniques suffer from errors due to both long-term numerical integration drift and incorrect initial joint angles.
To alleviate the difficulty, algorithms based on the feedback error correction are introduced (Balestrino et al., 1984; Wampler & Leifer, 1988). However, it is assumed that the exact model of manipulator Jacobian matrix of the mapping from joint coordinate to Cartesian coordinate is exactly known. It is also not sure to what extent the uncertainty could be allowed. Therefore, most research on robot control has assumed that the exact kinematics and Jacobian matrix of the manipulator from joint space to Cartesian space are known. This assumption leads to several open problems in the development of robot control laws today (Antonelli et al., 2003).
A new direction making control systems able to attribute more intelligence and high degrees of autonomy was proposed. With proper development, intelligent control systems may have great potential for solving today's and tomorrow's more complex control problems. The common objective associated with an intelligent control system can be identified to reduce accurate crisp model dependence and increase intelligent abilities of the control system.
Owing to this motivation, there have been increasing research interest of ANNs and a number of realistic control approaches have been proposed and justified for their feasible applications to robotic systems (D'Souza et al., 2001; Ogawa et al., 2005; Köker, 2005; Hasan et al., 2006; Al-Assadi et al., 2007). Artificial neural network (ANN) uses data sets to obtain the models of systems in fields such as robotics, factory automation and autonomous vehicles. Their ability to learn by example makes artificial neural networks very flexible and powerful. Therefore, neural networks have been intensively used for solving regression and classification problems in many fields. In short, neural networks are nonlinear processes that perform learning and classification. Recently neural networks have been used in many areas that require computational techniques such as pattern recognition, optical character recognition, outcome prediction and problem classification. The current focuses in learning research lies on increasingly more sophisticated algorithms for the off-line analysis of finite data sets, without severe constraints on the computational complexity of the algorithms (Bingual et al., 2005).

Kuroe and colleges (Kuroe et al., 1994) have proposed a learning method of a neural network such that the network represents the relations of both the positions and velocities from the Cartesian coordinate to the joint space coordinate. They've derived a learning algorithm for arbitrary connected recurrent networks by introducing adjoint neural networks for the original neural networks (Network inversion method). On-line training has been performed for a 2 DOF robot.

It was essentially an on-line learning process (Graca & Gu, 1993) have developed a Fuzzy Learning Control algorithm. Based on the robotic differential motion procedure, the Jacobian inverse has treated as a fuzzy matrix and has learned through the fuzzy regression process. It was significant that the fuzzy learning control algorithm neither requires an exact kinematics model of a robotic manipulator, nor a fuzzy inference engine as is typically done in conventional fuzzy control. Despite the fact that unlike most learning control algorithms, multiple trials are not necessary for the robot to "learn" the desired trajectory. A major drawback was that it only remembers the most recent data points introduced, the researchers have recommended neural networks so that it would remember the trajectories as it traversed them.

Studying the trajectory tracking of a serial manipulator by using ANNs has two problems, one of these is the selection of the appropriate type of network and the other is the generating of suitable training data set (Funahashi, 1998; Hasan et al, 2007). Researchers have applied different methods for gathering training data, while some of them have used the kinematics equations (Karilk & Aydin, 2000; Bingual et al., 2005), others have used the network inversion method (Kuroe et al., 1994); Köker, 2005), while the cubic trajectory planning was also used (Köker et al., 2004), a simulation program has also been used for this purpose (Driscoll, 2000). However, there are always kinematics uncertainties presence in the real world such as ill-defined linkage parameters, links flexibility and backlashes in gear train.

The proposed solution of the kinematics Jacobian in this approach, involves the determination of the end-effectors coordinates and their rate of change as a function of given positions and speed of the axes of motion, although this is very difficult in practice (Hornic, 1991), training data were recorded experimentally from sensors fixed on each joint and the Euler (RPY) representation was used to represent the orientation (as was recommended by Karilk and Aydin (Karilk & Aydin, 2000), as they have used the robot model to get the training data and used the homogeneous transformation matrix representation to represent the orientation). On the other hand, two different network's configurations were trained and compared to examine the effect of the orientation on the Inverse Kinematics solution of serial robots. Finally, the obtained results from the testing phase of the best network were verified experimentally using a six DOF serial robot manipulator.

2. Kinematics of serial robots

For serial robot manipulators, the Cartesian space coordinates x of a robot manipulator is related to the joint coordinates q by:

$$x = f(q) \tag{1}$$

where $f(\cdot)$ is a non-linear differential function.

If the Cartesian coordinates x were given, joint coordinates q can be obtained as:

$$q = f^{-1}(x) \tag{2}$$

If a Cartesian linear velocity is denoted by V, the joint velocity vector $\dot{q}$ has the following relation:

$$V = J\dot{q} \tag{3}$$

Where J is the Jacobian matrix.

If V, is a desired Cartesian velocity which represents the linear velocity of the desired trajectory to be followed. Then, the joint velocity vector $\dot{q}$ can be resolved by:

$$\dot{q} = J^{-1}V \tag{4}$$

In differential motion control, the desired trajectory is subdivided into sampling points separated by a time interval Δt between two terminal points of the path. Assuming that at time t_i the joint positions take on the value $q(t_i)$, the required q at time $(t_i + \Delta t)$ is conventionally updated by using:

$$q(t_i + \Delta t) = q(t_i) + \dot{q}\,\Delta t \tag{5}$$

Substituting Eqs. (2) and (4) into (5) yields:

$$q(t_i + \Delta t) = f^{-1}(x)(t_i) + J^{-1}V\Delta t \tag{6}$$

Equation (6) is a kinematics control law used to update the joint position q and is evaluated on each sampling interval. The resulting $q(t_i + \Delta t)$ is then sent to the individual joint motor servo-controllers, each of which will independently drive the motor so that the robotic manipulator can be maneuvered to follow the desired trajectory (Graca & Gu, 1993).

Using ANN to solve relation (2), researchers applied two approaches. In (Ogawa et al., 2005; Hasan et al., 2006; Köker et al.,2004) only the Cartesian coordinates has been inverted, mapping from the joint space to the Cartesian space is uniquely decided when the end effector's position is calculated using direct kinematics, as shown in figure 1(a). However, the transformation from the Cartesian to the joint space is not uniquely decided in the inverse kinematics as shown in figure 1(b).

When coupling of the position and orientation e.g., (Köker,2005; Karilk & Aydin, 2000) Denavit and Hartenberg (Denavit & Hertenberg, 1955) proposed a matrix method of systematically establishing a coordinate system to each link of an articulated chain as shown in figure 2 to describe both translational and rotational relationships between adjacent links (Fu et al., 1987; Köker, 2005).

In this method each of the manipulator links is modelled, this modelling describes the "A" homogeneous transformation matrix, which uses four link parameters. The forward kinematics solution can be obtained as:

$$A_{END-EFFECTOR} = T_6 = A_1.A_2.A_3.A_4.A_5.A_6 = \left[\begin{array}{c|c} \begin{matrix} Rotation \\ matrix \end{matrix} & \begin{matrix} Position \\ vector \end{matrix} \\ \hline \begin{matrix} Perspective \\ transformation \end{matrix} & Scaling \end{array}\right] = \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7)$$

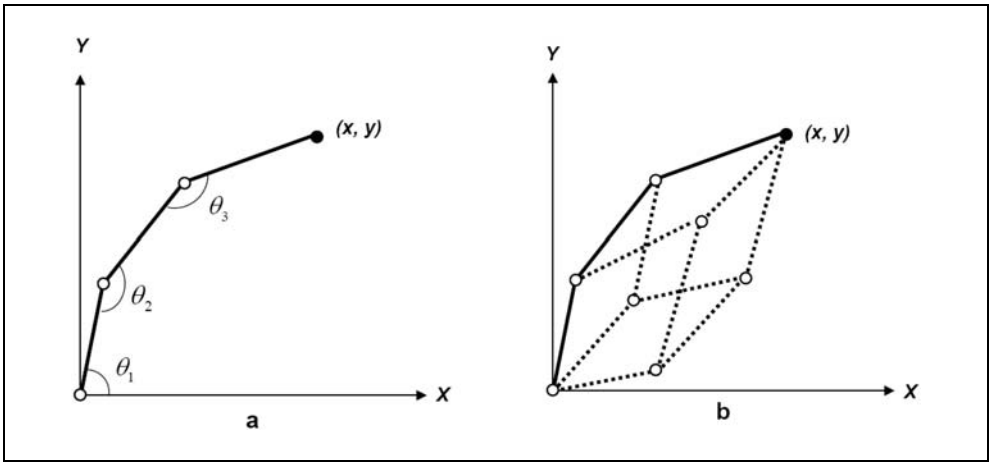

Fig. 1. a) Joint angles and end-effector's coordinates (forward kinematics).
b) Combination of all possible joint angles (Inverse Kinematics).

Where:

n: Normal vector of the hand. Assuming a parallel-jaw hand, it is orthogonal to the fingers of the robot arm.

a: Sliding vector of the hand. It is pointing in the direction of the finger motion as the gripper opens and closes.

a: Approach vector of the hand. It is pointing in the direction normal to the palm of the hand (i.e., normal to the tool mounting plate of the arm).

p: Position vector of the hand. It points from the origin of the base coordinate system to the origin of the hand coordinate system, which is usually located at the center point of the fully closed fingers.

The orientation of the hand is described according to the *RPY* rotation as:

$$RPY(\varphi_x, \varphi_y, \varphi_z) = Rot(Z_w, \varphi_z).Rot(Y_w, \varphi_y).Rot(X_w, \varphi_x) \quad (8)$$

After T_6 matrix is solved:

$$\varphi_z = ATAN2(n_y, n_x) \quad (9)$$

$$\varphi_y = ATAN2(-n_z, n_x \cos\varphi_z + n_y \sin\varphi_z) \quad (10)$$

$$\varphi_x = ATAN2(a_x \sin\varphi_z - a_y \cos\varphi_z, o_y \cos\varphi_z - o_x \sin\varphi_z) \quad (11)$$

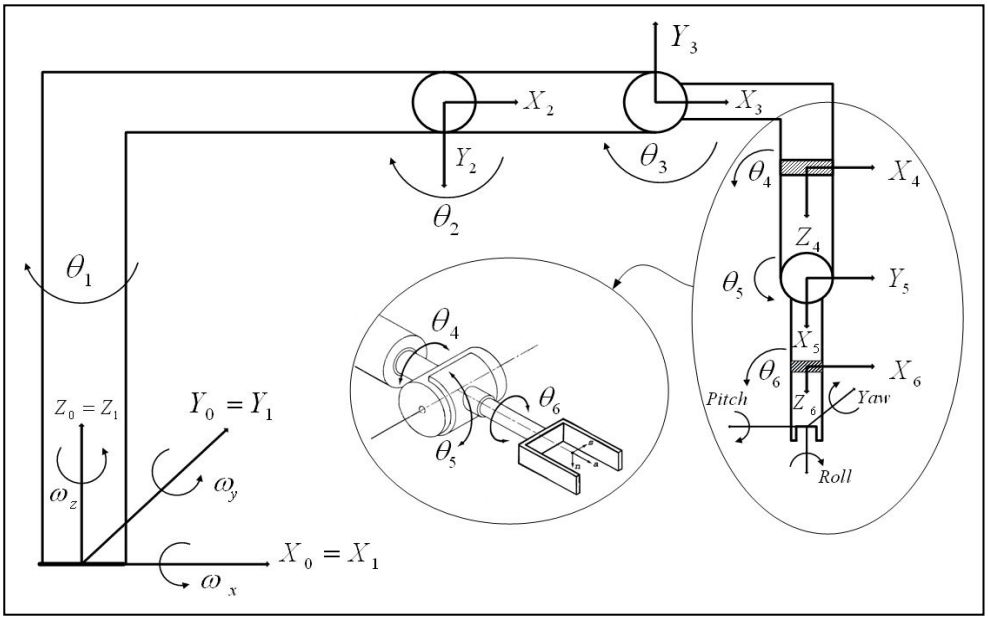

Fig. 2. Schematic diagram for a general 6 DOF serial robot showing the wrist mechanism

These equations describe the orientation according to the RPY representation (Karilk & Aydin, 2000). To find the IK solution, however, joints angels are found according to the manipulator's end position, described with respect to the world coordinate system.
IK solution can be shown as a function:

$$IK(X,Y,Z,\varphi_x,\varphi_y,\varphi_z) = (\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6) \tag{12}$$

Traditional methods for solving the IK problem are inadequate if the structure of the robot is complex, besides; these methods suffer from the fact that the solution does not give a clear indication on how to select an appropriate solution from the several possible solutions for a particular arm configuration, users often needs to rely on their intuition to choose the right answer (Fu et al., 1987; Hasan et al., 2006).
On the other hand, solving Eq. (4) for the joint velocities (Inverting the Jacobian matrix), results in the singularity problem. The manipulator singularity resolution problem has attracted many research interests, and various approaches have been proposed to tackle the problem. Techniques of coping with kinematics singularities can be divided into four groups: avoiding singular configurations, robust inverses, a normal form approach and extended Jacobian techniques.
The first approach to cope with singularities is to keep a current configuration far away from singular configurations. Unfortunately, it causes severe restrictions on the configuration space as well as the workspace because the singular configurations split the configuration space into separate components. To avoid ill conditioning of the Jacobian matrix, robust inverses are used. Instead inverting the original Jacobian matrix at singularity, a disturbed well-conditioned Jacobian matrix is inverted. The main drawback using this approach is that robust inverse methods increase errors in following a desired path.

The normal form technique, with the use of diffeomorphisms in joint and task spaces, expresses original kinematics around singularity in the simplest normal form. Then, a piece of the path to follow corresponding to the singular configuration mapped into the task space is moved from the task to the joint space and trajectory planning is performed there. Far away from singularities the basic Newton algorithm is used to generate a trajectory. Finally, trajectory pieces are joined.
For most singularities the normal form approach enables to detect their types. It provides for a smooth passing through singular configurations. The main disadvantage of the normal form approach is a significant computational load in deriving the diffeomorphisms.
Finally, The extended Jacobian technique, supplements original kinematics with auxiliary functions. Then, extended Jacobian is formulated to be well conditioned.
For nonredundant manipulators with square Jacobian matrices the extended Jacobian forms a non-square matrix and its generalized (Moore-Penrose) inversion is computationally expensive (Dulęba & Sasiadek, 2000).
Therefore, to analyze the singular conditions of a manipulator and develop effective algorithms to resolve the inverse kinematics problem at or in the vicinity of singularities are of great importance.

3. Artificial neural networks

The possibility of developing a machine that would "think" has intrigued human beings since ancient times, Machinery can outperform humans physically. Similarly, computers can outperform mental functions in limited areas, notably in the speed of mathematical calculations. For example, the fastest computers developed are able to perform roughly 10 billion calculations per second. But making more powerful computers will probably not be the way to create a machine capable of thinking. Computer programs operate according to set procedures, or logic steps, called algorithms. In addition, most computers do serial processing such as operations of recognition and computations are performed one at a time. The brain works in a manner called parallel processing, performing a number of operations simultaneously. To achieve simulated parallel processing, artificial neural networks (ANNs) are collections of small individual interconnected processing units. Information is passed between these units along interconnections. An incoming connection has two values associated with it, an input value and a weight. The output of the unit is a function of the summed value. ANNs while implemented on computers are not programmed to perform specific tasks. Instead, they are trained with respect to data sets until they learn the patterns presented to them. Once they are trained, new patterns may be presented to them for prediction or classification (Kalogirou, 2001).
The elementary nerve cell called a neuron, which is the fundamental building block of the biological neural network. Its schematic diagram is shown in Figure 3.
A typical cell has three major regions: the cell body, which is also called the soma, the axon, and the dendrites. Dendrites form a dendritic tree, which is a very fine bush of thin fibbers around the neuron's body. Dendrites receive information from neurons through axons-Long fibbers that serve as transmission lines. An axon is a long cylindrical connection that carries impulses from the neuron. The end part of an axon splits into a fine arborization. Each branch of it terminates in a small end bulb almost touching the dendrites of neighbouring neurons. The axon-dendrite contact organ is called a *synapse*. The synapse is where the neuron introduces its signal to the neighbouring neuron (Zurada, 1992; Hasan et al., 2006),

to stimulate some important aspects of the real biological neuron. An ANN is a group of interconnected artificial neurons usually referred to as "node" interacting with one another in a concerted manner; Figure 4 illustrates how information is processed through a single node. The node receives weighted activation of other nodes through its incoming connections. First, these are added up (summation). The result is then passed through an activation function and the outcome is the activation of the node. The activation function can be a threshold function that passes information only if the combined activity level reaches a certain value, or it could be a continues function of the combined input, the most common to use is a sigmoid function for this purpose. For each of the outgoing connections, this activation value is multiplied by the specific weight and transferred to the next node (Kalogirou, 2001; Hasan, 2006).

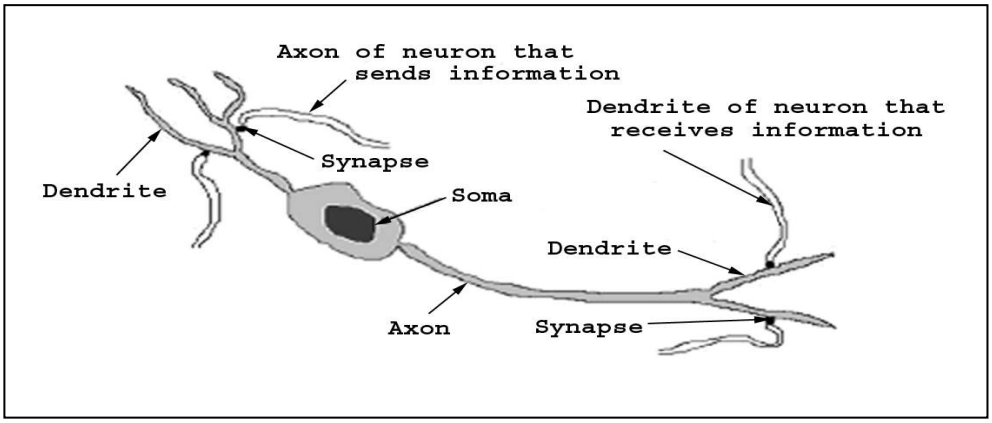

Fig. 3. Schematic diagram for the biological neuron

An artificial neural network consists of many nods joined together usually organized in groups called 'layers', a typical network consists of a sequence of layers with full or random connections between successive layers as Figure 5 shows. There are typically two layers with connection to the outside world; an input buffer where data is presented to the network, and an output buffer which holds the response of the network to a given input pattern, layers distinct from the input and output buffers called 'hidden layer', in principle there could be more than one hidden layer, In such a system, excitation is applied to the input layer of the network.

Following some suitable operation, it results in a desired output. Knowledge is usually stored as a set of connecting weights (presumably corresponding to synapse efficiency in biological neural system) (Santosh et al., 1993). A neural network is a massively parallel-distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the human brain in two respects; the knowledge is acquired by the network through a learning process, and interneuron connection strengths known as synaptic weights are used to store the knowledge (Haykin, 1994).

Training is the process of modifying the connection weights in some orderly fashion using a suitable learning method. The network uses a learning mode, in which an input is presented to the network along with the desired output and the weights are adjusted so that the network attempts to produce the desired output. Weights after training contain meaningful information whereas before training they are random and have no meaning (Kalogirou, 2001).

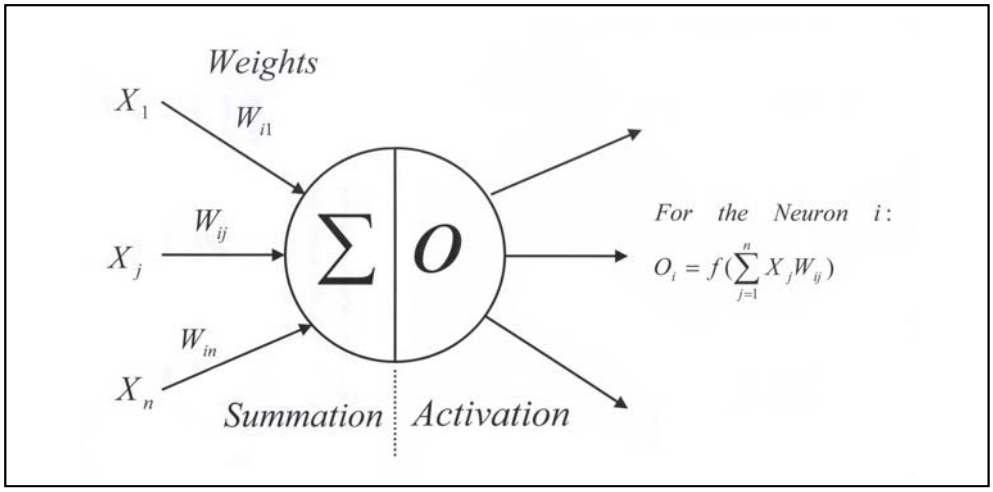

Fig. 4. Information processing in the neural unit

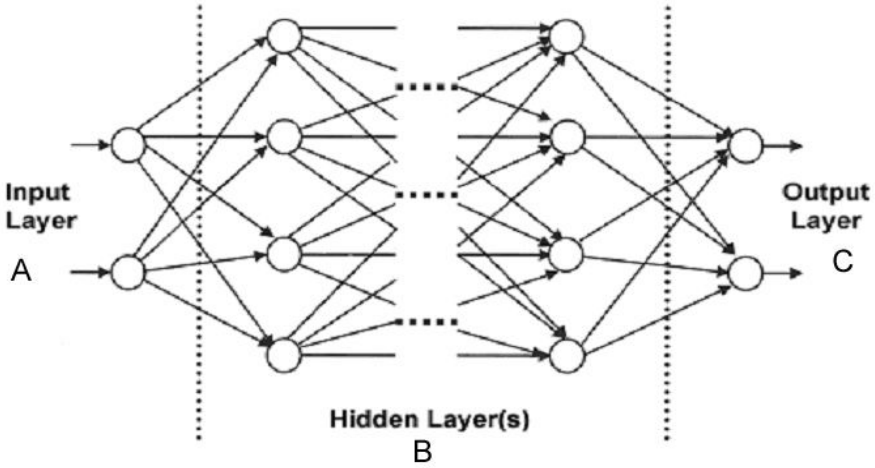

Fig. 5. Schematic diagram of a multilayer feedforward neural network

Two different types of learning can be distinguished: supervised and unsupervised learning, in supervised learning it is assumed that at each instant of time when the input is applied, the desired response d of the system is provided by the teacher. This is illustrated in Figure 6-a. The distance ρ [d,o] between the actual and the desired response serves as an error measure and is used to correct network parameters externally. Since adjustable weights are assumed, the teacher may implement a reward-and-punishment scheme to adopt the network's weight. For instance, in learning classifications of input patterns or situations with known responses, the error can be used to modify weights so that the error decreases. This mode of learning is very pervasive.

Also, it is used in many situations of learning. A set of input and output patterns called a training set is required for this learning mode. Figure 6-b shows the block diagram of unsupervised learning. In unsupervised learning, the desired response is not known; thus, explicit error information cannot be used to improve network's behaviour. Since no information is available as to correctness or incorrectness of responses, learning must somehow be accomplished based on observations of responses to inputs that we have marginal or no knowledge about (Zurada, 1992).
The fundamental idea underlying the design of a network is that the information entering the input layer is mapped as an internal representation in the units of the hidden layer(s) and the outputs are generated by this internal representation rather than by the input vector. Given that there are enough hidden neurons, input vectors can always be encoded in a form so that the appropriate output vector can be generated from any input vector (Santosh et al., 1993).

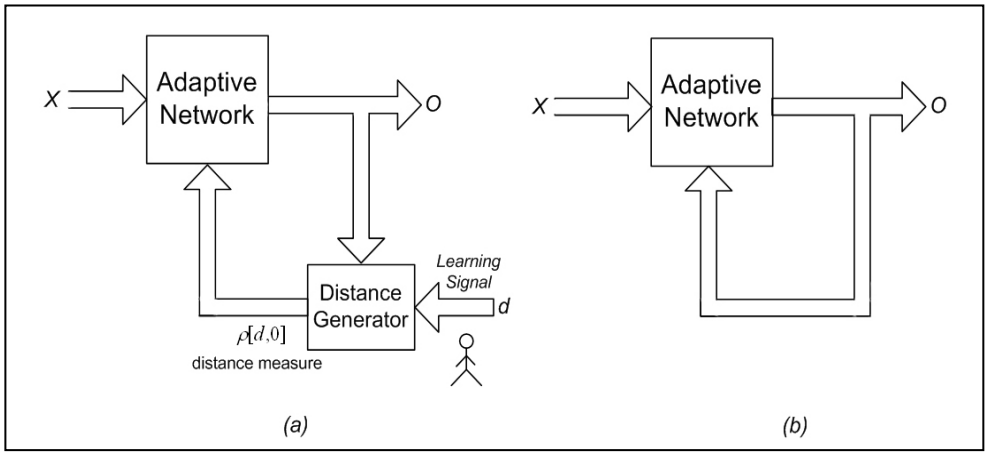

Fig. 6. Basic learning modes

As it can be seen in figure 5, the output of the units in layer ***A*** (Input Layer) are multiplied by appropriate weights W_{ij} and these are fed as inputs to the hidden layer. Hence if O_i are the output of units in layer ***A,*** then the total input to the hidden layer, i.e., layer ***B*** is:

$$Sum_B = \sum_i O_i W_{ij} \tag{13}$$

And the output O_j of a unit in layer ***B*** is:

$$O_j = f(sum_B) \tag{14}$$

Where f is the non-linear activation function, it is a common practice to choose the sigmoid function given by:

$$f(O_j) = \frac{1}{1+e^{-O_j}} \tag{15}$$

as the nonlinear activation function. However, any input-output function that possesses a bounded derivative can be used in place of the sigmoid function. If there is a fixed, finite set

of input-output pairs, the total error in the performance of the network with a particular set of weights can be computed by comparing the actual and the desired output vectors for each presentation of an input vector. The error at any output unit e_K in the layer **C** can be calculated by:

$$e_K = d_K - O_K \tag{16}$$

Where d_K is the desired output for that unit in layer **C** and O_K is the actual output produced by the network .the total error E at the output can be calculated by:

$$E = \frac{1}{2}\sum_K (d_K - O_K)^2 \tag{17}$$

Learning comprises changing weights so as to minimize the error function and to minimize E by the gradient descent method. It is necessary to compute the partial derivative of E with respect to each weight in the network. Equations (13) and (14) describe the forward pass through the network where units in each layer have there states determined by the inputs they received from units of lower layer. The backward pass through the network that involves "back propagation " of weight error derivatives from the output layer back to the input layer is more complicated. For the sigmoid activation function given in equation (15), the so-called delta-rule for iterative convergence towards a solution maybe stated in general as:

$$\Delta W_{JK} = \eta \delta_K O_J \tag{18}$$

Where η is the learning rate parameter, and the error δ_K at an output layer unit K is given by:

$$\delta_K = O_K(1 - O_K)(d_K - O_K) \tag{19}$$

And the error δ_J at a hidden layer unit is given by:

$$\delta_J = O_J(1 - O_J)\sum_K \delta_K W_{JK} \tag{20}$$

Using the generalize delta rule to adjust weights leading to the hidden units is back propagating the error-adjustment, which allows for adjustment of weights leading to the hidden layer neurons in addition to the usual adjustments to the weights leading to the output layer neurons. A back propagation network trains with two step procedures as it is shown in figure 7, the activity from the input pattern flows forward through the network and the error signal flows backwards to adjust the weights using the following equations:

$$W_{IJ} = W_{IJ} + \eta \delta_J O_I \tag{21}$$

$$W_{JK} = W_{JK} + \eta \delta_K O_J \tag{22}$$

Until for each input vector the output vector produced by the network is the same as (or sufficiently close to) the desired output vector (Santosh et al., 1993).

ANNs while implemented on computers are not programmed to perform specific tasks. Instead, they are trained with respect to data sets until they learn the patterns presented to them. Once they are trained, new patterns may be presented to them for prediction or classification (Kalogirou, 2001).

4. Experiment design

Trajectory planning was performed for every 1-second interval using cubic trajectory planning method to generate the angular position and velocity for each joint, and then these generated data were fed to the robot's controller to generate the corresponding Cartesian position, orientation and linear velocity of the end-effector, which were recorded experimentally from sensors fixed on the robot joints.

In trajectory planning of a manipulator, it is interested in getting the robot from an initial position to a target position with free of obstacles path. Cubic trajectory planning method has been used in order to find a function for each joint between the initial position, θ_0, and final position, θ_f of each joint.

It is necessary to have at least four-limit value on the $\theta(t)$ function that belongs to each joint, where $\theta(t)$ denotes the angular position at time t.

Two limit values of the function are the initial and final position of the joint, where:

$$\theta(0) = \theta_0 \tag{23}$$

$$\theta(t_f) = \theta_f \tag{24}$$

Additional two limit values, the angular velocity will be zero at the beginning and the target position of the joint, where:

$$\dot{\theta}(0) = 0 \tag{25}$$

$$\dot{\theta}(t_f) = 0 \tag{26}$$

Based on the constrains of typical joint trajectory listed above, a third order polynomial function can be used to satisfy these four conditions; since a cubic polynomial has four coefficients.

These conditions can determine the cubic path, where a cubic trajectory equation can be written as:

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{27}$$

The angular velocity and acceleration can be found by differentiation, as follows:

$$\dot{\theta}(t) = a_1 + 2a_2 t + 3a_3 t^2 \tag{28}$$

$$\ddot{\theta}(t) = 2a_2 + 6a_3 t \tag{29}$$

Substituting the constrain conditions in the above equations results in four equations with four unknowns:

$$\begin{aligned} \theta 0 &= a_0, \\ \theta_f &= a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3, \\ 0 &= a_0, \\ 0 &= a_1 + 2a_2 t_f + 3a_3 t_f^2 \end{aligned} \tag{30}$$

The coefficients are found by solving the above equations.

$$\begin{aligned} a_0 &= \theta_0, \\ a_1 &= 0, \\ a_2 &= \frac{3}{t_f^2}(\theta_f - \theta_0), \\ a_3 &= \frac{-2}{t_f^3}(\theta_f - \theta_0) \end{aligned} \tag{31}$$

Angular position and velocity can be calculated by substituting the coefficients driven in Eq. (31) into the cubic trajectory Equations (27) and (28) respectively (Köker et al.,2004), which yield:

$$\theta_i(t) = \theta_{i0} + \frac{3}{t_f^2}(\theta_{if} - \theta_{i0})t^2 - \frac{2}{t_f^3}(\theta_{if} - \theta_{i0})t^3, \tag{32}$$

$$\dot{\theta}_i(t) = \frac{6}{t_f^2}(\theta_{if} - \theta_{i0})t - \frac{6}{t_f^3}(\theta_{if} - \theta_{i0})t^2 \tag{33}$$

$i = 1, 2, ..., n$ Where n is the joint number

Joint angles generated ranged from amongst all the possible joint angles that do not exceed the physical limits of each joint; Table 1 shows the range of angles for each joint used in this study.

	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6
Range of angles	$0^\circ - 160^\circ$	$0^\circ - 60^\circ$	$0^0 - 150^\circ$	$0^0 - 150^\circ$	$0^\circ - 120^\circ$	$0^\circ - 160^\circ$

Table 1. The range of angles for each joint used

Trajectory used for the training process has meant to be random trajectory rather than a common trajectory performed by the robot in order to cover as most space as possible of the robot's working cell. The interval of 1 second was used between a trajectory segment and another where the final position for one segment is going to be the initial position for the next segment and so on for every joint of the six joints of the robot.

After generating the joint angles and their corresponding angular velocities, these data are fed to the robot controller, which is provided with a sensor system that can detect the angular position and velocity on one hand and the Cartesian position, orientation and the linear velocity of the end-effector on the other hand; which are recorded to be used for the networks' training and testing process later.

5. ANN implementation

To avoid modeling kinematics and the determination of the inverse of the Jacobian matrix, the ANN technique has been used.
Two different configurations of supervised feed-forward ANNs were designed using C programming language, each of which consists of input, output, and one hidden layer. Every neuron in each network was fully connected with each other. Sigmoid transfer function was chosen to be the activation function, and the generalized backpropagation GDR algorithm was used in the training process.
Off-line training was implemented, every input and output values are usually scaled individually such that overall variance in the data set is maximized, this is necessary as it leads to faster learning, all the vectors were scaled to reflect continuous values ranges from -1 to 1.
FANUC M-710i robot was used in this study, which is a serial robot manipulator consisting of axes and arms driven by servomotors. The place at which arm is connected is a joint, or an axis. This type of robot has three main axes; the basic configuration of the robot depends on whether each main axis functions as a linear axis or rotation axis. The wrist axes are used to move an end effecter (tool) mounted on the wrist flange. The wrist itself can be wagged about one wrist axis and the end effecter rotated about the other wrist axis, this highly non-linear structure makes this robot very useful in typical industrial applications such as the material handling, assembly of parts and painting.

5.1 Training stage

In order to overcome the uncertainties in arm configuration and singularities that result from applying the robot system model, and to make sure that for a certain trajectory the angular position and velocity of each joint will be the same as desired when planning the trajectory for the robot; the ANN technique has been utilized where learning is only based on observation of the input–output relationship.
In back-propagation networks, the number of hidden neurons determines how well a problem can be learned. If too many are used, the network will tend to try to memorize the problem and thus not generalize well later; if too few are used, the network will generalize well but may not have enough power to learn the patterns well. Obtaining the correct number of hidden neurons is a matter of trial and error.

5.1.1 Networks' topologies

In this chapter, two different configurations were used in the training process to determine which configuration is better to be used corresponding to Eq. (2) previously discussed in section 2.

5.1.1.1 First Configuration (4 – 12 Network Configuration)

As can be seen in Figure 7, the input layer consists of 4 neurons the first three of them represent the Cartesian position of the X, Y and Z positions along the world coordinate system of the robot while the fourth neuron represents the linear velocity of the end-effector. The output layer consists of 12 neurons; the first 6 of them represent the angular position of the robot joints while the last 6 of them represent the angular velocity of each joint respectively. Number of neurons in the hidden layer was set to 77 with a constant learning factor of 0.9 by trail and error.

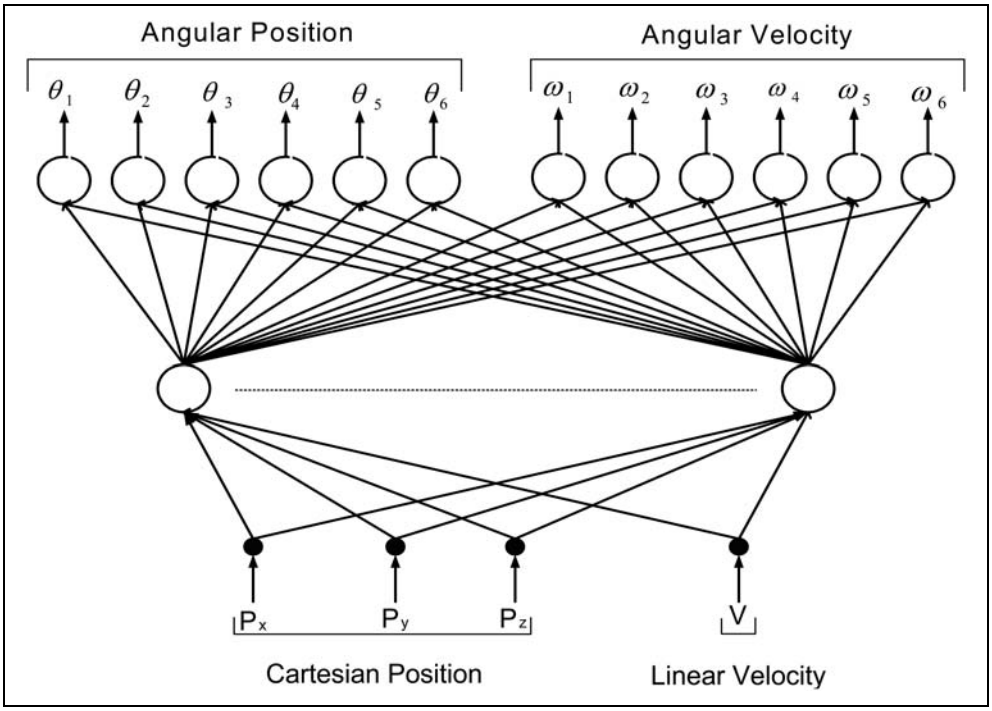

Fig. 7. The Topology of the First Network (4 –12 Network Configuration)

5.1.1.2 Second Configuration (7 – 12 Network Configuration)

In this configuration, the input layer has 7 neurons; the first three of them represent the X, Y and Z coordinates of the robot along the world coordinates system, the next three represent the orientation of the tool mounted on the last joint of the robot according to the RPY (Roll, Pitch, Yaw) representation, while the last neuron represents the linear velocity of the end-effector; as can be seen in figure 8.

Same as the first configuration, the output layer consists of 12 neurons; the first 6 of them represent the angular position of the robot joints while the last 6 of them represent the angular velocity of each joint respectively.

Number of neurons in the hidden layer was set to 55 with a constant learning factor of 0.9 by trail and error.

5.1.2 Networks' performance

The success of the ANN approach is measured according to the training error (the difference between the desired and actual system outputs). In the Generalized Delta learning Rule GDR the system is modified following each iteration, which leads to the learning curves a sample of which is shown in Figure 9 of each network configuration compared to the other (the rest of the curves have a similar behavior), as this curve shows; error is reduced in subsequent trials.

Table 2, shows the error percentages of each of the six joints compared for each other in both network configurations after the training has finished after 150 000 iteration.

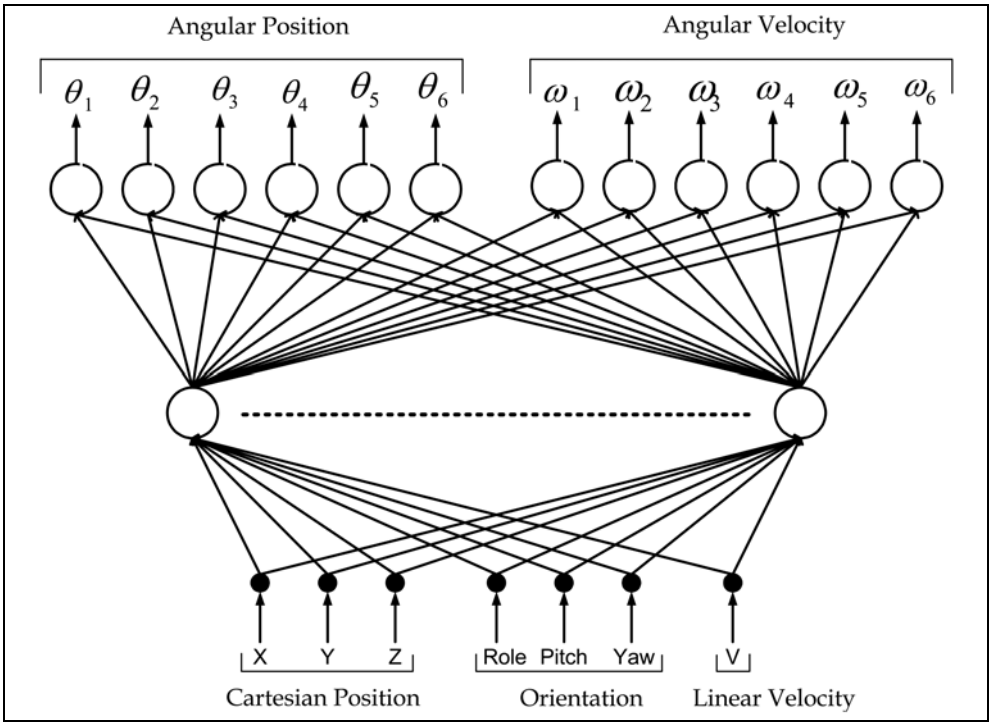

Fig. 8. The Topology of the Second Network (7 –12 Network Configuration)

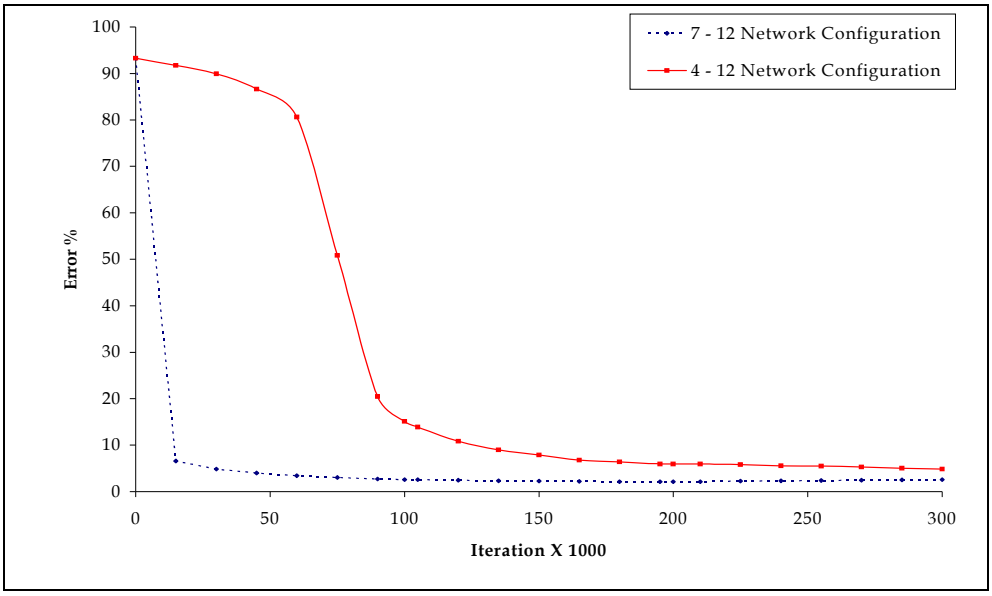

Fig. 9. The learning curve for the angular position of Joint 1 as a sample

		Network Configuration	
		4 - 12	7 - 12
Joint 1	θ	7.898%	2.27%
	ω	8.67%	1.8%
Joint 2	θ	12.432%	0.907%
	ω	39.75%	2.183%
Joint 3	θ	2.607%	1.033%
	ω	5.03%	1.775%
Joint 4	θ	9.82%	2.015%
	ω	10.4%	2.342%
Joint 5	θ	8.47%	4.435%
	ω	19.94%	1.558%
Joint 6	θ	10.86%	1.143%
	ω	5.735%	1.528%

Table 2. Training error percentages of each of the six joints compared for each other in both network configurations

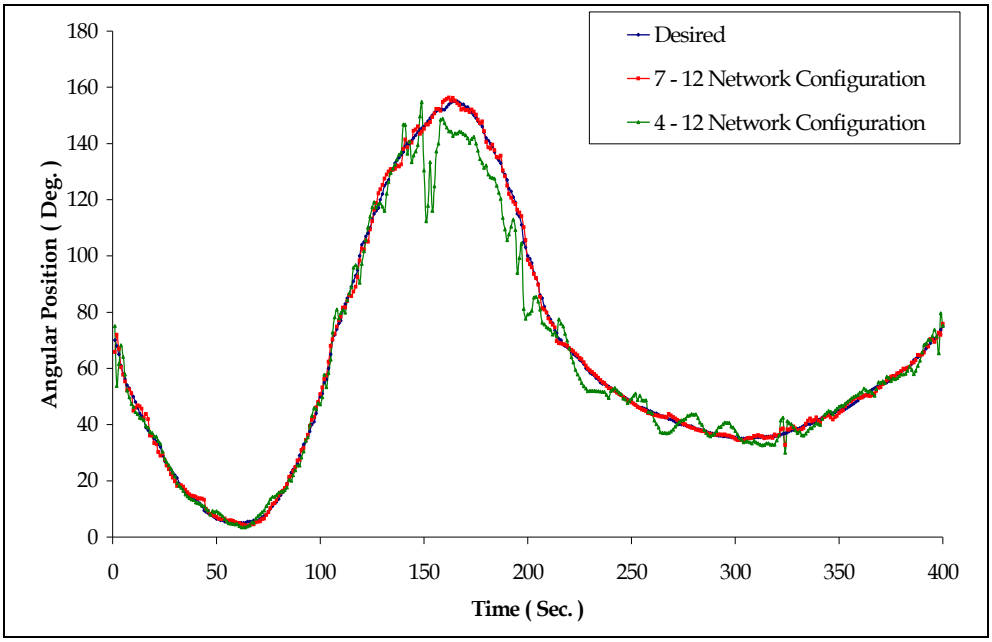

Fig. 10. The response of both network configurations compared to each other during training (The angular position of the firs joint as an example)

As a result for the training stage, 7-12 network configuration has shown a better response than the 4-12 network configuration, in terms of precision and iteration (as can be seen through Table 2). Therefore, it has been chosen to apply the testing data.

To drive the robot to follow a desired path, it will be necessary to divide this path into small segments, and to move the robot through all intermediate points. To accomplish this task, at each intermediate location, the robot's trajectory equations are solved, a set of joint variables

is calculated, and the controller is directed to drive the robot to the next segment. When all segments are completed, the robot will be at the end point as desired. Figure 10 shows a sample of angular position and velocity for each joint during training (other joints have a similar behavior).

5.2 Testing phase

New data that has never been introduced to the network before have been fed to the trained network in order to test its ability to make prediction and generalization to any set of data later overcoming the singularity and uncertainty in the arm configuration resulting from applying the robot model.

Testing data were meant to pass nearby and through the singular configurations (Fourth and Fifth joints), these configurations have been determined by setting the determinant of the Jacobian matrix to zero.

Table 3 shows the percentages of error for the testing data set for each joint.

In order to verify the testing results, experiment has been performed to make sure that the output is the same or sufficiently close to the desired trajectory, and to show the combined effect of error, Figures 11 to 16 show the tracking of the Cartesian paths for the X, Y, and Z coordinates with the Roll, Pitch and Yaw orientation angles respectively.

The locus of which robot is passing through singular configurations are also shown. The error percentages in the experimental data are shown in Table 4.

	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6
Angular Position	0.06%	0.029%	0.039%	5.865%	5.065%	1.495%
Angular Velocity	3.79%	4.285%	3.745%	3.085%	4.97%	2.1%

Table 3. Error percentages for the testing data set for each joint

Cartesian Position			Orientation		
P_x	P_y	P_z	Roll	Pitch	Yaw
5.645%	1.09%	3.93%	5.95%	9.24%	5.338%

Table 4. Error percentages in the experimental data

6. Conclusions

In this approach, ANN technique has been used. The Jacobian inverse is now learned through training the network based only on observation of the input–output relationship unlike most other control schemes, which depends on the robot system model

The proposed technique does not require any prior knowledge of the kinematics model of the system being controlled, the basic idea of this concept is the use of the ANN to learn the characteristics of the robot system rather than to specify explicit robot system model.

Two different ANN configurations were used in this study. Training results have shown a better response (in terms of precision and iteration) for the configuration where the orientation of the tool is considered as an input to the network, which makes it useful in applications where a relatively accurate, minimally complex, and cheaper configuration is required.

As a conclusion, this study shows that ANNs are applicable to the Kinematics Jacobian solution of serial robots. Since one of the most important issues in using ANNs is the selection of the appropriate type of network, for future research, we suggest that different

types of networks (different topology, different activation function, different learning mode) to be used in order to get, if possible, more accurate trajectory tracking.

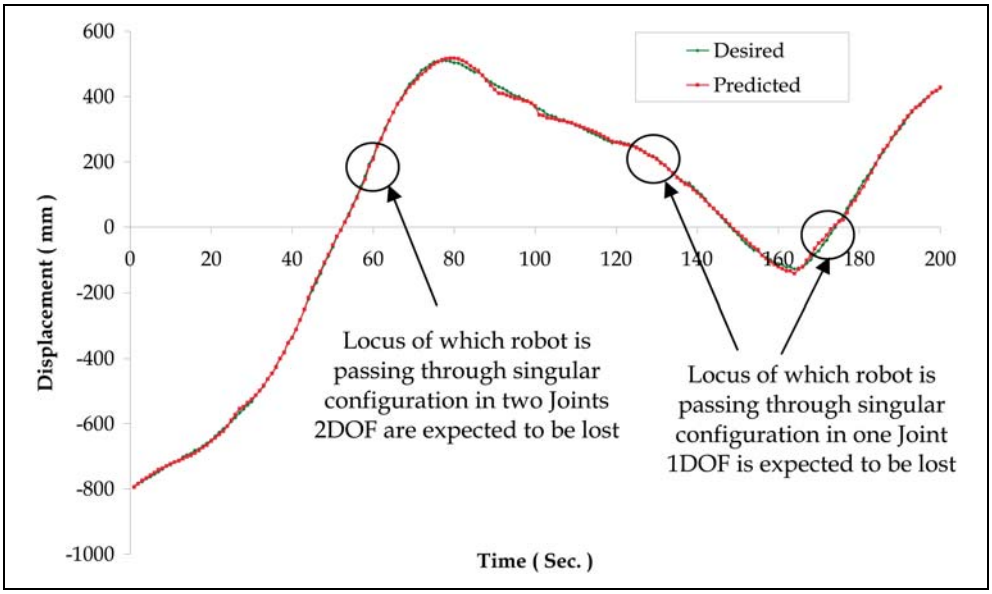

Fig. 11. Experimental trajectory tracking for the predicted X coordinate

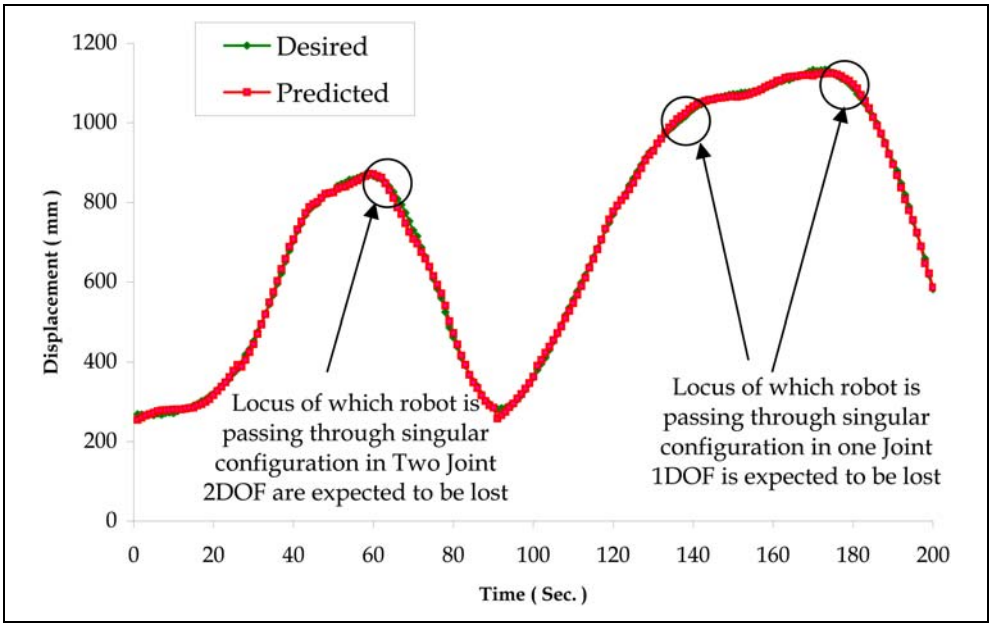

Fig. 12. Experimental trajectory tracking for the predicted Y coordinate

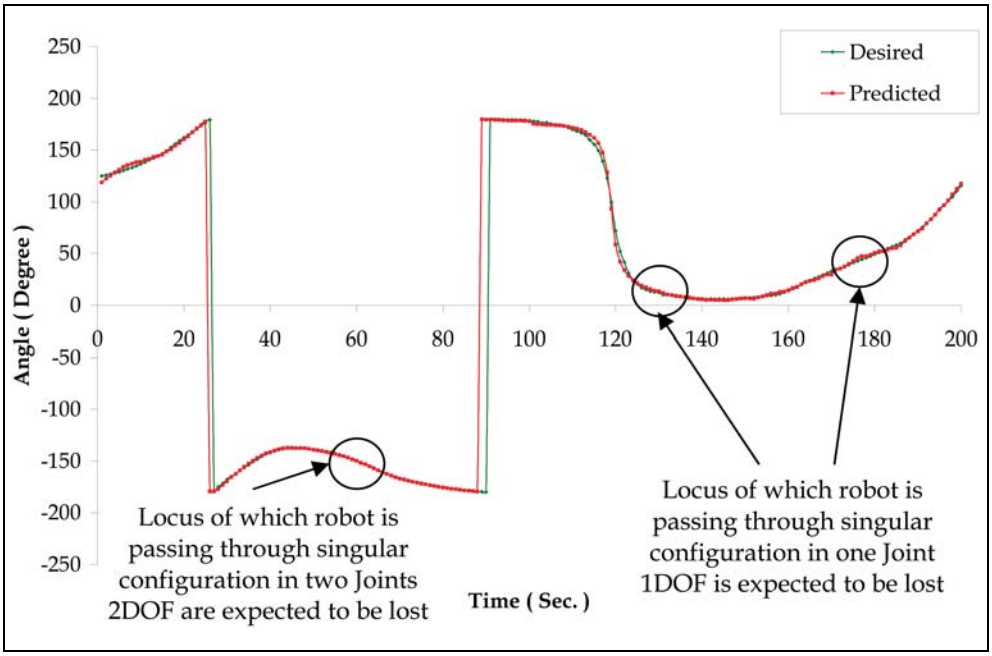

Fig. 13. Experimental trajectory tracking for the predicted Z coordinate

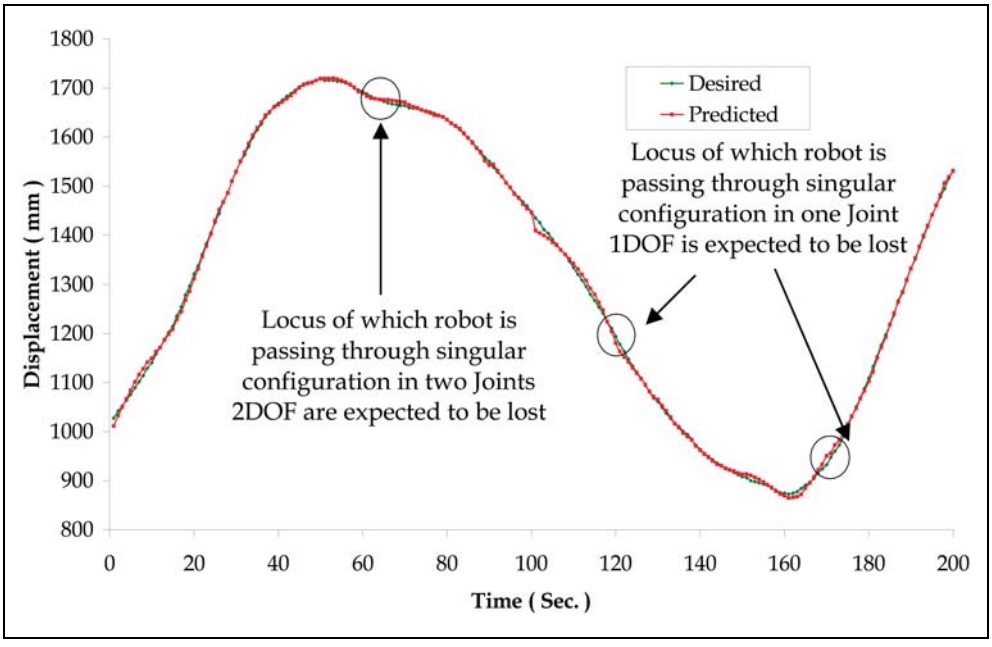

Fig. 14. Experimental trajectory tracking for the Roll orientation angle

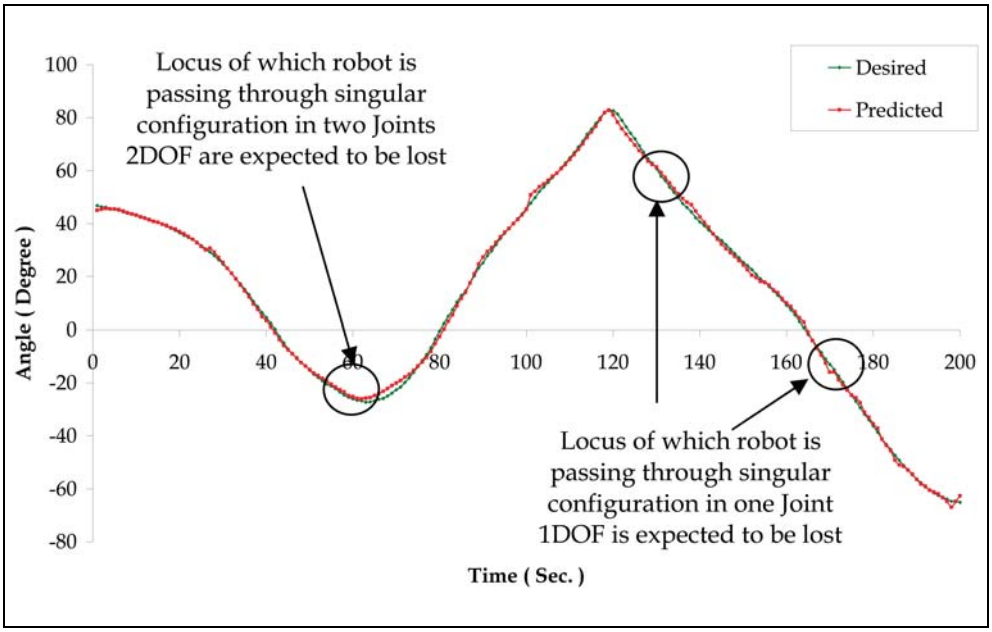

Fig. 15. Experimental trajectory tracking for the Pitch orientation angle

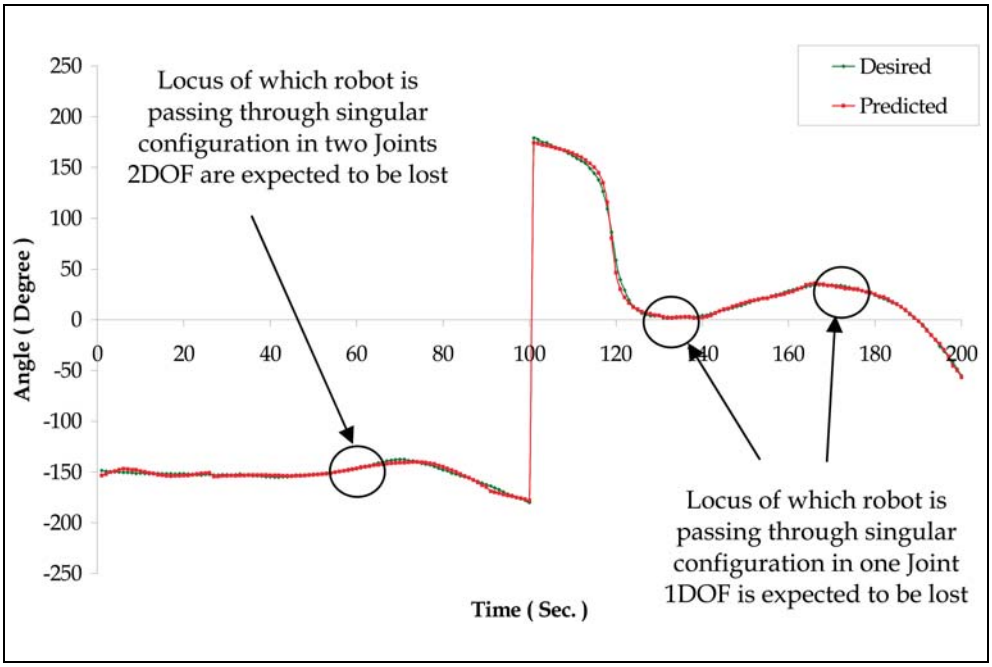

Fig. 16. Experimental trajectory tracking for the Yaw orientation angle

7. References

Al-Assadi, H.M.A.A.; Hamouda, A.M.S.; Ismail, N. and Aris, I. (2007) .An adaptive learning algorithm for controlling a two-degree-of-freedom serial ball-and-socket actuator. *Proceedings of the I MECH E Part I Journal of Systems & Control Engineering*, Vol.221, No. 7, pp.1001-1006.

Albala, H. & Angeles, J. (1979). Numerical solution to the input–output displacement equation of the general 7R spatial mechanism. *Proceedings of the Fifth world congress on theory of machines and mechanisms*, pp 8–11.

Antonelli, G.; Chiaverini, S. and Fusco, G. (2003) .A new on-line algorithm for inverse kinematics of robot manipulators ensuring path-tracking capability under joint limits. *IEEE Transaction on Robotics and Automation*, Vol.19, No.1, pp. 162-167.

Asada, H. & Soltin, J-J. E., Robot analysis and control. John Wiley and Sons Inc., New York. 1986.

Balestrino, A., De Maria, G. and Sciavicco, L., Robust control of robotic manipulators. International Proceedings of the 9th IFAC World Congr. Budapest, Hungary .1984; 6:80-85.

Bingual, Z., Ertunc, H.M. and Oysu, C., Comparison of Inverse Kinematics Solutions Using Neural Network for 6R Robot Manipulator with Offset. 2005 ICSC congress on Computational Intelligence.

Daniel, M. and Raul, G., Hierarchical Kinematics analysis of robots. Mech Mach Theory. 2003; 33: 497–518.

Denavit, J., and Hertenberg, R.S.A. Kinematics Notation for lower Pair Mechanism Based on Matrices. Applied mechanics 1955; 77: 215-221.

Driscoll, J.A., Comparison of neural network architectures for the modeling of robot inverse kinematics. Proceedings of the IEEE, south astcon. 2000:44-51.

D'Souza, A., Vijayakumar, S., and Schaal, S. (2001) 'Learning Inverse Kinematics'. *Proceedings of the 2001 IEEE/ RSJ International Conference on Intelligent Robots and Systems Maui, Haw- USA*, pp.298-303.

Duffy, J. and Rooney, J., A foundation for a unified theory of analysis of spatial mechanism. Eng Ind. 1975; 97(4): 1159–64.

Dulęba, I., and Sasiadek, J.Z. (2000) 'Modified Jacobian method of transversal passing through singularities of nonredundant manipulators'. *Proceedings of the American Control Conference Chicago, Illinois June*, pp.2839-2843.

Faiz, N. and Agrawal, S. K., Trajectory planning of robots with dynamics and inequalities. In Proc. IEEE Int. Conf. Robotics and Automation, San Francisco, CA.2000: 3977–3983.

Fu, K.S., Gonzalez, R.C. and Lee, C.S.G. Robotics Control, Vision, and Intelligence. McGraw-Hill book Co. Singapore, international edition. 1987.

Funahashi, K.I., On the approximate realization of continuous mapping by neural networks. Neural Networks.1998; 2(3): 183-192.

Graca, R.A. and Gu, Y., A fuzzy learning algorithm for kinematics control of a robotic system. Proceeding of the 32nd conference on decision and control. San Antonio, Texas. December 1993:1274-1279.

Hasan, A.T., Hamouda, A.M.S., Ismail, N., and Al-Assadi, H.M.A.A. (2006) 'An adaptive-learning algorithm to solve the inverse kinematics problem of a 6 D.O.F serial robot manipulator'. *Int. J. Advances in Engineering Software*, Vol.37, pp. 432-438.

Hasan, A.T., Hamouda, A.M.S., Ismail, N., and Al-Assadi, H.M.A.A., A new adaptive learning algorithm for robot manipulator control. Proceeding of the I Mech E, Part I: Journal of System and Control Engineering .2007; 221(4): 663-672.

Haykin S. Neural Networks. A Comprehensive Foundation. New York: Macmillan, 1994.

Hornik, K., Approximation capabilities of multi-layer feed forward networks. IEEE Trans. Neural Networks. 1991; 4(2): 251-257.

Karilk, B., Aydin, S., An improved approach to the solution of inverse kinematics problems for robot manipulators. Journal of Engineering applications of artificial intelligence. 2000; 13: 159-164.

Köker, R. (2005) 'Reliability-based approach to the inverse kinematics solution of robots using Elman's networks'. *Int. J. Engineering Applications of Artificial Inttelegence*, Vol.18, pp. 685-693.

Köker, R., Öz, C., Çakar.T. and Ekiz, H., A study of neural network based inverse kinematics solution for a three-joint robot. Robotics and Autonomous Systems. 2004; 49: 227–234.

Kuroe, Y., Nakai, Y. and Mori, T., A new Neural Network Learning on Inverse Kinematics of Robot Manipulators. International Conference on Neural Networks, IEEE world congress on computational Intelligence. 1994; 5: 2819-2824.

Nakamura, Y. and Hanafusa, H., Inverse kinematic solutions with singularity robustness for robot manipulator control. Journal of Dynamic Systems Measurements Control.1986; 108: 163–171.

Ogawa, T., Matsuura, H., and Kanada, H. (2005) 'A Solution of Inverse Kinematics of Robot Arm Using Network Inversion'. *Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation.*

Santosh, A. Devendra P. Garg. Training back propagation and CMAC neural networks for control of a SCARA robot. Engineering Applications of Artificial Intelligence. Vol.6.No.2. pp.105-115. 1993.

Sopng, M.W., and Vinyasagar, M., Robot Dynamics and control. John Wiley and Sons Inc., New York. 1998.

Soteris, A.Kalogirou. Artificial Neural Networks In Renewable Energy Systems Applications: a review. Renewable and Sustainable Energy Reviews. Vol. 5:pp.373-401. 2001.

Tsai, L.W. and Morgan, A., Solving the Kinematics of the most general six and five degree of freedom manipulators by continuation methods. Mech Transm Autom Des 1985; 107:189–200.

Wampler, C. W. and Leifer, L. J., Applications of damped least-squares methods to resolved-rate and resolved-acceleration control of manipulators, Journal of Dynamic Systems Measurements Control.1988; 110: 31–38.

Wampler, C. W., Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods. IEEE Transaction Syst., Man, Cybernetics. 1986; 16: 93–101.

Whitney. E., Resolved motion rate control of manipulators and human prostheses, IEEE Transaction Man–Mach. Systems.1969; 10:47–53.

Yang, A.T., Displacement analysis of spatial five-link mechanism using (3×3) matrices with dual-number element. Eng Ind. 1969; 9(1): 152–7.

Zurda, M. J. (1992). *Introduction to Artificial Neural System Network*. West Publishing Companies, ISBN 0-314-93397-3, St. Paul, MN, USA.