The Islamic University of Gaza

Scientific Research& Graduate Studies Affairs

Faculty of Engineering

Electrical Engineering Depart.

الجامعة الإسلامية – غزة

شئون البحث العلمي و الدراسات العليا

كلية الهندسة

قسم الهندسة الكهربائية

Trajectory Tracking Control of A 2-DOF Robot Arm Using Neural Networks

Mahmoud M. Al Ashi

Advisors

Dr. Hatem Elaydi

Dr. Iyad Abu Hadrous

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering

February 2014

ABSTRACT

This thesis investigated several control strategies to handle the trajectory tracking problem for a two degree-of-freedom (2-DOF) robotic arm using artificial neural networks (ANNs). Feed-forward two layer neural networks were designed and utilized in both model-based and non-model based control structures to conduct online learning and identification of the inverse dynamics of the robotic manipulator and to compensate for both structured and unstructured uncertainties.

The simulation results obtained proved the superiority of the proposed neural network controllers to dramatically reduce the error between the desired and actual position trajectories even in the presence of uncertainties unlike other conventional methods such as the PD-computed torque method. The neural network-based controllers proposed in this thesis provide solutions to the trajectory tracking problem of robotic manipulators with or without a mathematical model which would make them effective controllers for both planned and unplanned trajectory tracking problems for any degree of freedom robotic manipulator.

The development of the mathematical models for the 2-DOF robotic arm and its joints driving motors as well as their simulation experiments were carried out under the Dynamic Modeling Laboratory (Dymola) environment which uses the Modelica object-oriented multi-domain system modeling language. The simulation results obtained in the thesis were accompanied by three dimensional (3D) figures in order to visualize the results and to help establish a deeper analysis and understanding of these results.

ACKNOWLEDGEMENT

I would like to dedicate the first page of the thesis to express my warmest greetings and gratitude to those without whom this thesis would not have seen the light:

First, All my thanks and gratitude go to our god and the god of all worlds ALLAH whose sincere guidance was being felt through every single step of preparing and writing up this thesis.

The second "thank you" goes to my parents, Moneer and Taghreed, who never hesitated to provide me with all the psychological and materialistic helps in order to achieve success in my life.

A big big "thank you" is presented to the loves of my life, my wife Sarah, and my son Zain who walked me safely through the road of sacrifice and patience until my graduation and the successful submission of this thesis.

Finally, I would like to deeply thank my supervisors Dr. Iyad Abu Hadrous and Dr. Hatem El-Aydi for their patience and sincere support, guidance, and advices without which this thesis would not have been submitted in the present manner.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

θ_i, d_i, α_i, a_i	Denavit-Hartenberg (DH) parameters
T_i^j	Homogeneous transformation matrix from frame i to frame j
x_2, y_2, z_2	Cartesian coordinates of the manipulator end-effector
θ_i	Angular position of revolute joint i
$\dot{\theta}_i$	Angular velocity of revolute joint i
v	Total linear velocity at the manipulator end-effector
ω	Total angular velocity at the manipulator end-effector
J_v	Jacobian matrix of the end-effector linear velocity
J_ω	Jacobian matrices of the end-effector angular velocity
O_i	Origin of frame i
L	Lagrangian of the robotic manipulator
K	Total kinetic energy
P	Total potential energy
τ_j	Torque or force applied on the j^{th} joint.
q_j	j^{th}Joint variable
m_i	Mass concentrated at the center of link i
v_i	Linear velocity at the center of mass of link i
ω_i	Angular velocity at the center of mass of link i
I_i	Inertia tensor matrix at the center of mass of link i
J_{vij}	Component of the linear velocity v_i due to the angular velocity of joint j
I_{ci}	Inertia tensor matrix of link i with respect to its center frame
$I_{xxi}, I_{yyi}, I_{zzi}$	Principal moments of inertia intertia tensor matrix of link i
ρ	Link mass density
R_{ci}	Rotational matrix of the center frame of link i with respect to the inertial frame
g	Gravity vector expressed in the inertial frame
K_i	Kinetic energy of link i
P_i	Potential energy of link i
r_{ci}	Position vector of the center of mass of link i with respect to the inertial frame
R	Armature resistance
$V(t)$	Input voltage signal
i_a	Armature current
τ_m	DC-Motor torque
θ_m	DC-Motor angular position
V_b	Back electromotive force
K_m, K_b	DC- Motor constants
$r: 1$	Gear reduction ratio
J_m	DC-motor inertia
B_m	DC-motor damping coefficient
τ_L	Load torque
θ_{mi}	Angular position of the DC-motor driving the joint i
$L(s)$	Open-loop transfer function
$G_{cl}(s)$	Closed-loop transfer function
t_s	Settling time

$P.O.$	Peak overshoot
ω_n	Natural undamped frequency
ρ_d	Damping ratio
S_D	Design closed loop system pole
K_P	Proportional gain
K_D	Derivative gain
$D(\theta)$	Inertia matrix of the robotic manipulator
$C(\theta,\dot{\theta})$	Vector of centrifugal and coriolis forces
$g(\theta)$	Vector of gravitational forces
$h(\theta,\dot{\theta})$	Vector of centrifugal, coriolis, and gravitational forces
$\widehat{D}(\theta)$	Estimated inertia matrix of the robotic manipulator
$\hat{h}(\theta,\dot{\theta})$	Vector of estimated centrifugal, coriolis, and gravitational forces
d_{ij}	Element ij of the inertia matrix of the robotic manipulator
h_i	The i^{th} element of the vector $h(\theta,\dot{\theta})$
$C(s)$	Controller transfer function
$R(s)$	Laplace transform of the reference input signal
$G(s)$	Plant transfer function
$F(s)$	Transfer function of the feed-forward controller
$E(s)$	Laplace transform of the error signal $e(t)$
τ_{di}	Disturbance torque applied on joint i
a	Neuron's output
$\boldsymbol{W}$	Weight matrix of neural network
$\boldsymbol{b}$	Bias vector of neural network
n	Neuron's net input
$\boldsymbol{p}$	Input vector of neural network
$f(n)$	Neuron activation function
$\boldsymbol{f}$	Matrix of activation functions
$\boldsymbol{a}$	Output vector of neural network
$\boldsymbol{W^i}$	Weight matrix of the i^{th} layer of neural network
$\boldsymbol{a^i}$	Output vector of the i^{th} layer of neural network
$\boldsymbol{b^i}$	Bias vector of the i^{th} layer of neural network
E	Error performance index function
$E(k)$	Value of error performance index function at iteration k
$\mathbf{e}(k)$	Output error vector of neural network at iteration k
$\boldsymbol{a_d}(k)$	Desired output vector of neural network at iteration k
$w_{ij}^m(k)$	The ij element of the weight matrix of the m^{th}layer at iteration k
α	Learning rate
n_i^m	The i^{th} element of the net input vector of the m^{th}layer
S^m	Sensitivity vector associated with the m^{th} layer
S^M	Sensitivity vector associated with the output layer
$\dot{\mathbf{F}}^m(n^m)$	Derivative matrix of the activation functions of the m^{th}layer with respect to their net inputs

ABBREVIATIONS

DOF	Degree of Freedom
ANN	Artificial Neural Network
PD	Proportional Derivative
Dymola	Dynamic Modeling Laboratory
PID	Proportional Integral Derivative
CAD	Computer Aided Design
DC	Direct Current
RNNC	Recurrent Neural Network Controller
NFC	Neuro Fuzzy Controller
CMAC	Cerebellar Model Articulation Controller
SCARA	Self Compliant Articulated Robotic Arm
DH	Denavit Hartenberg
CTC	Computed Torque Control
FFC	Feed Forward Controller
ADALINE	Adaptive Linear
LM	Levenberg Marquardt
SDA	Steepest Descent Algorithm
OTC	Online Torque Compensator
FEL	Feedback Error Learning
3D	Three Dimensional

CHAPTER 1

Introduction

1.1 Research motivation

Robotic Manipulators are widely used in different fields of industry. They are used for the purpose of saving time, effort, and sometimes life. This made robot manipulators play key roles in fields like car manufacturing, space exploration, search and rescues, waste treatment in nuclear plants, in addition to their different applications in medical surgery. For these reasons and due to the vast applications of robotic manipulators, the design of controllers to optimize the tracking and speed performance of robots has become a necessity and an important research area.

In order to design a controller to control the motion of a manipulator, an accurate mathematical model for the robot must be first developed. This requires accurate determination of the manipulator parameters such as masses, inertias, and geometrical properties of the links, and the friction between the gearboxes of the robot joints. The masses and inertias of the links are usually determined from Computer Aided Design (CAD) models, but the friction between the gearboxes depend on the positions and velocities of the joints and this cannot be determined without experimentation which would be very difficult in high speed operation [1]. After developing a mathematical model for the robot, the inverse kinematics and dynamics problems must be solved in order to determine the desired position, velocity, and acceleration of each robot joint, as well as the necessary torques and forces to be applied to enforce these joints to follow their desired positions and velocities. These quantities are important to generate the right control signal for each robot joint.

1.2 Statement of the problem

When designing a controller to control the trajectory tracking performance of a manipulator, some of the problems encountered during the controller design process are as follows:

1- The inverse kinematics and dynamics problems require the solution of complicated highly nonlinear equations which would take much time and processing power when solved offline using a computer software. Moreover, the complexity of such equations increases with the number of Degree of Freedom (DOF) of the manipulator.

2- Due to the structured and unstructured uncertainties in the values of the link and joint parameters, there is always a difference between the conventional mathematical model used for the manipulator and the real robot which would generate a considerable error between the desired and actual trajectories. Therefore, a conventional model-based controller is ineffective in controlling the robot in real time. This problem can be alleviated using adaptive control approaches to compensate for the modeling errors. One commonly-used adaptive control approach is the computed torque method [1-2]. However, the design of a conventional adaptive controller for a manipulator trajectory tracking application requires the solution of highly nonlinear equations that describe the dynamic behavior of the manipulator which is a time consuming task especially for manipulators with a high DOF. In addition, if an adaptive controller works to compensate for the structured uncertainties such as the joint friction, it might not necessarily be capable of compensating for the unstructured uncertainties whose dynamics are not considered in the conventional model.

3- The control methodologies which depend on controlling each joint of the manipulator independently, such as the PID-based control of each joint, are not effective. This is because such methodologies do not count for the coupled interaction between the joints which result in the generation of coupling disturbance torques between the different joints of the manipulator [2].

1.3 Thesis contribution

This thesis aims to investigate the possibility of designing neural network-based controllers to enhance the trajectory tracking performance of a robotic manipulator of unknown dynamics. This idea was motivated by the learning capabilities of neural networks to approximate and identify nonlinear systems [3]. The controller is capable of generating the required torques to enforce the manipulator joints to follow their desired position trajectories with an acceptable precision. The main contribution in this thesis is that the proposed controller is able to emulate the manipulator dynamic behavior without the need to have a complex nonlinear mathematical model for the robot. In addition, the proposed neural network controller is able to conduct online updating of its weights to compensate for any structured and unstructured uncertainties in the model such as joint friction forces or sudden changes in the load.

1.4 Research methodologies

In order to explore the capabilities of neural network-based controllers in effectively controlling the trajectory tracking performance of a 2-DOF robotic manipulator and to show their superiority over other conventional control techniques, the following tasks are achieved throughout the thesis:

1- In the beginning of the research, a comprehensive review is established of the literature and state-of-the-art related to the theory and applications of neural networks in the design of manipulator trajectory tracking controllers. Such a review provides a detailed exposure to the modern adaptive control approaches, different neural network architectures, as well as the commonly-used learning algorithms for training neural networks. The literature review also allows for determining the disadvantages of the different neural network control schemes proposed so far which enables the development of novel neural control strategies.

2- In order to design a controller for a specific system, a model for the system must be first developed. Therefore, a detailed description of the manipulator structure and its parameters must be determined in order to facilitate the derivation of the kinematics and inverse dynamics equations of the manipulator. The kinematics equations are necessary for the determination of the desired angular positions of the joints from a planned trajectory given in the Cartesian space, whereas the dynamics equations are necessary to develop model-based trajectory tracking control strategies.

3- Since the joints of a robotic manipulator are driven by motors, the use of permanent magnet DC motors is investigated in driving the two joints of the robotic arm. This requires the development of a mathematical model of the permanent magnet DC motor and then connecting such a model with the model of the robotic arm derived in step 2.

4- One of the important steps in the process of designing a manipulator trajectory tracking controller is the design of a linear PD controller in order to achieve the stability of the motors angular positions before using them to drive the manipulator joints.

5- When connecting a PD-controlled motor to each joint of the robotic arm through a gear reducer, the disturbance torques generated due to the nonlinear dynamics of the manipulator will dramatically affect the time response of the motor system which reflects the poor performance of the PD controller alone to reject such disturbances. This requires adding another adaptive controller in order to remove the effect of such disturbance torques. One of the adaptive controllers used in the literature to handle the disturbance rejection problem is the computed torque method. This method primarily depends on the availability of a mathematical model to estimate the inverse dynamics of the robotic manipulator. Therefore, it is considered to be a highly complicated approach for a high DOF manipulator. In addition, later simulation results show its poor efficiency to compensate for un-modeled dynamics such as joint friction forces. However, for the reasons of later comparison with the adaptive neural network-based control techniques, the computed torque method is adopted and applied as a disturbance torque rejection method for the 2-DOF robotic arm.

6- In order to avoid the complexity of the computed torque-based controller design and to handle its poor performance in compensating for un-modeled dynamics, intelligent control mechanisms must be adopted. There are various intelligent control techniques available in the literature such as Fuzzy logic-based PID control [4], genetic algorithm-based PID control [5], and neuro-fuzzy control [6]. One of the intelligent adaptive controllers used in manipulator trajectory tracking applications is neural network-based controllers [7]. The universal approximation capabilities of neural networks which make them effective candidates for approximating any complex nonlinear function with a very simple structure have attracted researchers to employ them for identifying the highly nonlinear inverse dynamics of robotic manipulators [8]. In this thesis, model-based and non-model based neural network controllers are designed and utilized to improve the trajectory tracking performance of the robotic arm with and without a mathematical model.

7- As mentioned in step 6, a model-based neural network controller must be used in the presence of a computed torque disturbance rejection controller. Despite the fact that the model-based neural network would have to learn only the unknown factors of the system such as parameter inaccuracies and un-modeled dynamics, this model-based control strategy would be highly complicated and time-consuming for controlling a manipulator of higher DOF. In order to solve this problem, a non-model-based neural network training algorithm must be used so that the neural network can work both as an online identifier of the manipulator inverse dynamics and as a compensator for structured and unstructured uncertainties.

1.5 Literature review

Neural networks are considered one of the modern intelligent tools that are being utilized in position trajectory tracking applications of robotic manipulators. This is due to their simple structure and model as well as their universal complex function approximation capabilities gained through simple training algorithms. Neural networks used for manipulator trajectory tracking applications have been designed and used in different control configurations some of which are listed as follows:

Tomochika *et al.* [8] provided a Neural Network-based control strategy based on the idea of the computed torque method. The designed controller consisted of two separate three-layered neural networks, one of which was used to compute an estimated manipulator mass matrix, and the other network was trained to compute an estimated centrifugal and coriolis torque vector. The simulation results obtained by the authors proved the capability of the proposed controller to learn the nonlinear dynamics of a 2-DOF manipulator and it was able to enforce the end-effector to follow its desired position trajectory in the XY-plane. However, the speed of the model learning process of the controller cannot be considered to be fast since it took about 600 seconds to follow the specified trajectory of a small circle on the XY plane with an acceptable precision. This means that the proposed controller would be inefficient in more complex and larger trajectory tracking tasks. In addition, the presence of a mathematical model for the controlled manipulator is still necessary to generate the training data set for each of the two neural networks. In order to speed up the model learning process of the proposed controller, a huge amount of training data sets must be provided and they should be uniformly distributed over the entire work space of the manipulator which would be hardly achievable and time consuming for wider workspaces especially with a manipulator with a larger number of DOF.

Refaat *et al.* [7] proposed a robot trajectory tracking controller design which consisted of feed-forward and PD feedback components. The feed-forward control was performed by a three-layered Neural Network learned by a modified backpropagation algorithm to emulate the inverse dynamics equations of the manipulator. The proposed trained network was required to provide the necessary torque for each joint according to a given set of desired positions, velocities, and accelerations. The PD feedback component was used as an online learning signal to adjust the weights of the network in order to minimize the error in the generated torque due to any variations in the manipulator parameters or external disturbances. Despite the good performance shown by the proposed Neural Network-based controller to follow the desired joint position and speed trajectories with an acceptable error even in disturbance conditions, the structure of the network which contained a large number of neurons in the hidden layer (35 neurons) makes the training and weight adjustment process take a long time to reach the optimal weight matrices for generating the minimum output error. This large structure of the network even increases with the DOF of the controlled manipulator.

Kuo *et. al.* [11] introduced an online learning control method in which a feed-forward compensator is proposed to learn and compensate for the unknown dynamic torques of the manipulator joints. The proposed feed-forward compensator is composed of a PD controller and a cerebellar model articulation controller (CMAC). The proposed CMAC controller was designed using a feed-forward single-layer neural network whose output is the sum of selected vectors of its weight matrix unlike the conventional law of a traditional neural perceptron which involves all the vectors of the weight matrix. The CMAC controller selects the weight vectors based on an associative memory index vector corresponding to a given input state. Despite the approved capability of the proposed controller in effectively learning the unknown dynamics and hence dominating the control of the 2-DOF robotic arm, the number of its neurons is proportional to the number of samples taken from the input space which would complicate the network structure for larger workspaces despite the fact that only a few number of these neurons would be active for a given input reference state. The other majority of the inactive neurons can be a useless overhead on the system.

In [12], Zhao proposed a new hybrid non-model based trajectory tracking control strategy using a linear feedback controller in parallel with a feed-forward multiple-layer neural network controller. The linear feedback controller was used to regulate the joint position error and to generate actuating torques to help the joints to track their dynamic desired trajectories during the initial stage of the learning process of the neural controller. An error backpropagation algorithm was used to train the neural network controller online to learn and compensate for the uncertain inverse dynamics of the manipulator joints. The proposed hybrid controller was used to control the trajectory tracking performance of a SCARA AdeptOne XL industrial manipulator. Both simulation and experimental results proved the ability of the neural network to effectively learn and compensate for the unknown dynamics of the manipulator. The results also indicated that the neural controller was able to dominate the full control of the trajectory tracking of the manipulator as approved by the very small control inputs supplied by the linear feedback controller.

Kamel *et al.* [10] applied the idea of the Generalized Linear Prediction control to design a nonlinear predictive controller for the tracking performance of a 2-DOF manipulator using a neuronal model of the nonlinear system. The proposed controller was designed to predict the future values of the manipulator joint velocities over a finite time horizon. Those predicted values of the joint velocities were used to compute the future errors which enabled the generation of incremental changes in the torque control signal. The simulation results showed that the designed neuronal-model was effective in predicting the manipulator joint velocities but the prediction error never reached an acceptable steady value even after a high number of training samples. Moreover, the second joint velocity prediction error showed large values of about 10 rad/sec difference at the 800th sample which is an unacceptable value. Such a velocity prediction error caused the occurrence of a continuously changing error in the tracking of the desired velocity of the second joint. In addition, no simulation experiments were done to test the effectiveness of the predictive neuro controller in the presence of structured and unstructured uncertainties and disturbances.

Jafar *et al.* [6] proposed a Neuro-Fuzzy strategy for manipulator trajectory control by utilizing the advantages of both fuzzy logic based control and the learning capabilities of neural networks. The proposed controller incorporated a neural network which was trained offline to optimize the tuning of the parameters of the input and output membership functions of a designed fuzzy logic controller. The main contribution of this paper is that the rule base of the fuzzy controller inference system was determined by the used neural network. The neural network was trained using a hybrid learning algorithm which consisted of both the least squares method and the back-propagation method. The network training data set was obtained from the inputs and outputs of PID controllers. The simulation results obtained by the author showed the capability of the proposed NFC to track the desired position trajectories of a 2 DOF elbow manipulator even in the presence of uncertain frictional forces on each joint. The author claimed that the proposed NFC is effective to compensate for unstructured uncertainties and external disturbances but no simulation or experimental methods were used to verify their claim.

Joel *et al.* [9] used a new manipulator trajectory tracking control methodology using a recurrent neural network. Unlike the other used neural network-based controllers which required the presence of a large data set for the training of the used networks, the recurrent neural network-based controller (RNNC) proposed in this paper was trained online to minimize the error between the desired and actual trajectories of the manipulator. This means that no mathematical model of the controlled manipulator was necessary to be obtained. Despite the proved capability of the proposed RNNC to ensure the stability and trajectory tracking accuracy of a 2 DOF elbow manipulator, no simulation experiments were done to test the learning speed of network (e.g. the number of weight adjustment iterations the network took for the manipulator output to converge to an acceptable error was not shown). In addition, no experiments or simulations were done to test the effectiveness of the proposed RNNC to compensate unstructured uncertainties and external disturbances such as joint friction forces and payload changes.

1.6 Thesis Organization

This thesis is composed of five chapters organized in the following manner:

- Chapter 2 explains the detailed process of deriving a mathematical model for the 2-DOF robotic arm based on given structural and geometrical characteristics. This model constitutes the kinematics and inverse dynamics equations associated with the arm. A Dymola model based on Modelica language is developed in this chapter for the 2-DOF robotic arm using the built-in multi-body library contained in Dymola software. In the same chapter, mathematical and Dymola models are developed for the permanent magnet DC motor consisting of an armature circuit driving a mechanical load. The developed model is validated through carrying out simulations of the motor angular position with and without connecting a mechanical load.

- Chapter 3 explains the process of designing a PD controller for the permanent magnet DC motor system using the root locus method to achieve specified time response requirements. The performance of the designed PD controller is tested through simulating the step response of the motor angular position with and without connecting it to the manipulator joints. The chapter also explains how to handle the poor performance of the PD controller to compensate for the disturbance torques generated due to the inverse dynamics of the arm by using a computed-torque disturbance rejection controller.

- In chapter 4, a model-based intelligent controller is designed using a feed-forward two-layer neural network which is trained online using the steepest descent error back propagation algorithm to learn the uncertain parameters of both the robotic arm and the joint driving motors. The neural controller is used as an online torque compensator along with the computed-torque disturbance rejection controller developed in chapter 3 in order to remove the effect of the joint disturbance torques the driving motor angular positions. In addition, the neural controller is designed to conduct online training to adapt its weights in order to compensate for any structured and unstructured uncertainties. Chapter 4 also introduces a non-model based neural network controller which is used in the absence of a mathematical model for the robotic arm. The performance of both model-based and non-model based neural controllers are simulated and tested on Dymola.

- The thesis ends with a brief conclusion and future work plans in chapter 5.

CHAPTER 2

Two Degree-Of-Freedom Robotic Manipulator

In this chapter, the structure of a two degree-of-freedom robotic arm will be fully described together with the parameters of its links including the mass, inertia tensor, as well as the geometrical dimensions of each link. The main goal of describing the structure of the robotic manipulator is to enable the derivation of the kinematics and dynamics equations that will be used later in the design process of the controllers.

2.1 Geometrical structure

A two degree of freedom elbow manipulator consists primarily of two links which can take the form of a cylinder or a bar. The two links are connected together serially with a revolute joint called "*elbow*". The other revolute joint called the "*shoulder*" is used to connect the first link with the fixed part of the manipulator. A schematic diagram illustrating an upper view of the robotic manipulator is shown in Figure (2.1).

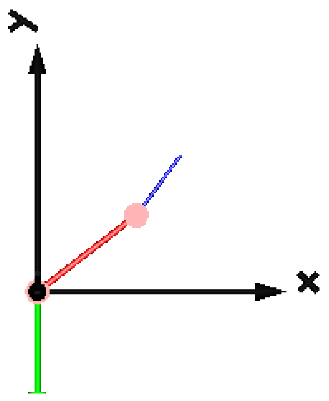


Figure 2.1: A schematic diagram of a 2-DOF robotic arm

Table (2.1) lists the values of the parameters related to the links of the manipulator such as the mass, inertia tensor, and geometrical dimensions of each link.

Table 2.1: Link parameters of the 2-DOF robotic arm

Quantity	1st Link	2nd Link
Length	0.25 m	0.15 m
Mass	1.95 Kg	0.93 Kg
The element I33 of the inertia tensor matrix	0.0980 Kg.m2	0.980 Kg.m2

2.2 Kinematics

The kinematics of a robotic manipulator refer to the mathematical equations that describe the forward and inverse relationships between the joint variables (angular positions of the revolute joints) and the Cartesian position coordinates of the manipulator end-effector [2]. The kinematics equations are very important tools in the process of mapping the manipulator desired trajectories from the Cartesian space to the joint space and vice versa. Such a mapping process will be considered when we come to the point of designing trajectory tracking controllers for the manipulator in Chapter 5.

The kinematics of a robotic manipulator can be divided into forward and inverse kinematics. The forward kinematics describe the Cartesian position coordinates of the end-effector as functions of the joint angular positions. Whereas, the inverse kinematics describe the joint angular positions as functions of the end-effector Cartesian coordinates. The following steps describe a general analytical procedure of deriving the forward and inverse kinematics of a robotic manipulator and will apply such a procedure for deriving the kinematics of the 2-DOF robotic arm shown in Figure (2.1).

2.2.1 Forward kinematics

In order to derive the forward kinematics of a robotic manipulator, the following steps must be followed:

a) Frame assignment

Attaching a frame rigidly with each link of a robotic manipulator enables to describe the motion of each link with respect to the other previous links. The first frame that must be defined is the inertial (world) frame with respect to which any point in the configuration space of the manipulator can be defined. Figure (2.2) shows a schematic diagram of the 2-DOF robotic arm with all necessary frames defined and attached to the links. As shown in figure (2.2), the z-axis of each frame represents the axis of rotation of one revolute joint of the manipulator. z_0 axis is the axis of rotation of the first revolute joint and z_1 is the axis of rotation of the second revolute joint. Since figure (2.2) provides a planar view of the robotic arm, the axes of rotations of both revolute joints are taken to be directed outside the page.

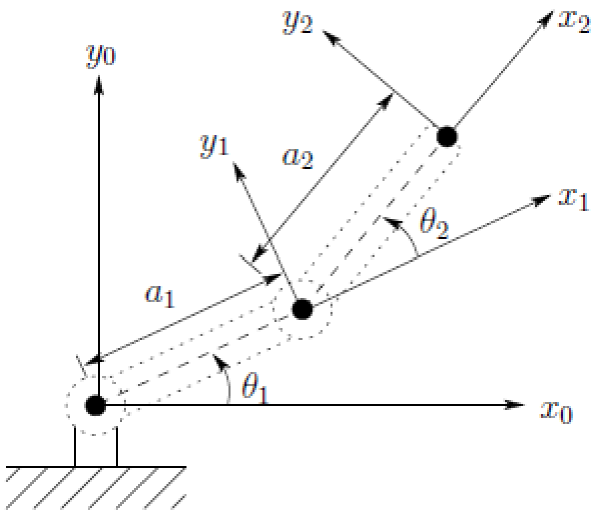


Figure 2.2: Frame assignment for the 2-DOF robotic arm

The direction of the x_0-axis of the inertial frame is chosen arbitrarily. However, the directions of the x-axes of the other frames are determined based on the following two assumptions called Denavit Hartenberg (DH) coordinate frame assumptions [2]:

DH1: The axis x_1 is perpendicular to the axis z_0.
DH2: The axis x_1 intersects the axis z_0.

These assumptions are clearly considered in the choice of the x-axes of the frames assigned for the robotic arm in figure (2.2). The x_1 axis intersects and is perpendicular to the z_0 axis and the x_2-axis intersects and is perpendicular to the $z_1$axis. After determining the directions of the z and x axes of a frame, the direction of the y-axis is determined using the right-hand rule.

b) Derivation of Denavit-Hartenberg (DH) parameters

The second step in the process of deriving the kinematics of a robotic manipulator is the determination of a group of parameters called Denavit-Hartenberg parameters. These parameters are important for deriving the homogenous transformation matrices between the different frames assigned on the manipulator structure in step (a) [2]. The Denavit-Hartenberg parameters for the link i of a robotic manipulator are defined as follows:

Joint angle (θ_i): is defined as the angle from the axis x_{i-1} to the axis x_i about the axis z_{i-1}.

Link offset (d_i): is defined as the perpendicular distance from the origin O_{i-1} to the intersection point of the axis x_i with the axis z_{i-1} along the axis z_{i-1}.

Link twist angle (α_i): denotes the angle from the axis z_{i-1} to the axis z_i measured about the axis x_i.

Link length (a_i): denotes the distance from the axis z_{i-1} to the axis z_i measured along the axis x_i.

Using the above definitions, the DH parameters for the 2-DOF robotic arm shown in figure (2.2) can be determined and listed in table (2.2).

Table 2.2: DH-parameters for the 2-DOF robotic arm

Link (i)	θ_i	d_i	α_i	a_i
1	$\theta_1{}^*$	0	0	a_1
2	$\theta_2{}^*$	0	0	a_2

c) Derivation of the homogenous transformation matrices

In order to express the position and orientation of a frame i with respect to another frame j, a so called homogenous transformation matrix T_i^j from the frame i to the frame j must be derived. This matrix is of 4×4 dimension. The first three rows and columns of the homogenous transformation matrix T_i^j represent the orientation of the frame i with respect to the frame j. The first three elements of the fourth column of the transformation matrix T_i^j represent the position coordinates of the origin of frame i with respect to the frame j.

The homogenous transformation matrices for the 2-DOF robotic arm shown in figure 1.2 are derived as follows:

$$A_1 = T_1^0 = \begin{bmatrix} x_1.x_0 & y_1.x_0 & z_1.x_0 & (O_1 - O_0)_x \\ x_1.y_0 & y_1.y_0 & z_1.y_0 & (O_1 - O_0)_y \\ x_1.z_0 & y_1.z_0 & z_1.z_0 & (O_1 - O_0)_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_1 & -s_1 & 0 & a_1c_1 \\ s_1 & c_1 & 0 & a_1s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} ---------- \quad (2.1)$$

$$A_2 = T_2^1 = \begin{bmatrix} x_2.x_1 & y_2.x_1 & z_2.x_1 & (O_2 - O_1)_x \\ x_2.y_1 & y_2.y_1 & z_2.y_1 & (O_2 - O_1)_y \\ x_2.z_1 & y_2.z_1 & z_2.z_1 & (O_2 - O_1)_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_2 & -s_2 & 0 & a_2c_2 \\ s_2 & c_2 & 0 & a_2s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} ----------- \quad (2.2)$$

Using the equations (2.1) and (2.2), the homogenous transformation matrix T_2^0 can be derived as follows:

$$T_2^0 = A_1A_2 = T_1^0T_2^1 = \begin{bmatrix} c_{12} & -s_{12} & 0 & a_2c_{12} + a_1c_1 \\ s_{12} & c_{12} & 0 & a_2s_{12} + a_1s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} ------ \quad (2.3)$$

The homogenous transformation matrix defined in (2.3) is the one that defines the forward kinematics of the 2-DOF robotic arm shown in figure (2.2). From this matrix, the position coordinates of the manipulator end-effector is given by:

$x_2 = a_2c_{12} + a_1c_1$,
$y_2 = a_2s_{12} + a_1s_1$,
$z_2 = 0$

And the end-effector's orientation matrix is defined by the first three rows and three columns of the transformation matrix (2.3).

2.2.2 Inverse kinematics

The inverse kinematics of a robotic manipulator is concerned by the problem of finding the manipulator joint variables (angular positions of the revolute joints) given the position Cartesian coordinates of the end-effector. The mathematical equations used to solve the inverse kinematics problem can be derived either algebraically or geometrically [2]. The geometrical approach is considered to be much easier for manipulators of high degrees of freedom. In this section, the inverse kinematics equations for the 2-DOF robotic arm shown in figure (2.2) will be derived using the geometrical method as follows:

Figure (2.3) shows a geometrical manipulation on the planar view of the 2-DOF robotic arm. This geometrical manipulation involved projecting the position of the end-effector on the x_0 and y_0 axes of the inertial frame as well as on the x_1 axis of frame $x_1y_1z_1$.

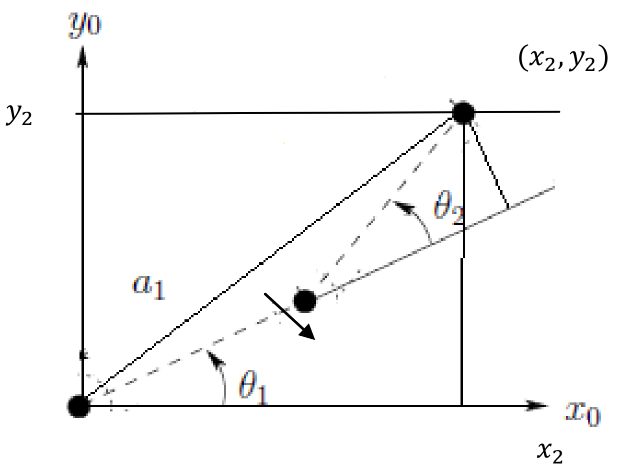


Figure 2.3: Geometrical approach for inverse kinematıcs derivation

From figure (2.3), a mathematical equation for solving the elbow joint angle θ_2 can be derived using Pythagoras theorem as follows:

$$x_2^2 + y_2^2 = a_1^2 + a_2^2 + 2a_1a_2\cos(\theta_2)$$

Therefore,

$$\theta_2 = cos^{-1}\left(\frac{x_2^2 + y_2^2 - a_1^2 - a_2^2}{2a_1a_2}\right) ------------- \quad (2.4)$$

It can also be shown from figure (2.3) that:

$$\theta_1 = tan^{-1}\left(\frac{y_2}{x_2}\right) - tan^{-1}\left(\frac{a_2s_2}{a_1 + a_2c_2}\right) ----------- \quad (2.5)$$

The equations (2.4) and (2.5) are used to solve the inverse kinematics problem for the 2-DOF robotic arm shown in figure (2.2).

2.2.3 Singular configuration

A robotic manipulator is said to be in a singular configuration when it loses one of its degrees of freedom [2,13]. This happens when the robot joint velocities become extremely high [2,13] which is not a favorable case in practice. In order to find the conditions when a singular configuration occurs, the so called Jacobian matrices must be derived. The Jacobian matrices of a robotic manipulator define the relationships between the linear and angular velocities of the manipulator's end-effector and its joints' velocities. The

Jacobian matrices for the 2-DOF robotic arm shown in figure (2.2) are derived as follows:

Let v and ω be the 3×1 vectors of the linear and angular velocities computed at the end-effector of the 2-DOF robotic arm, then:

$$v = J_v\dot{\theta} \;------------- (2.6)$$
$$,\ \omega = J_\omega\dot{\theta} ---------\ ---- (2.7)$$

Where J_v and J_ω represent the Jacobian matrices of the end-effector's linear and angular velocities, respectively and $\dot{\theta}$ represents the 2×1 vector of the robot joint angular velocities. Therefore, the dimension of each Jacobian matrix will be 3×2.

The equation (2.6) can be rewritten as follows:

$$\begin{bmatrix}\dot{x}_2\\ \dot{y}_2\\ \dot{z}_2\end{bmatrix} = [J_{v1} \quad J_{v2}]\begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix}$$

Where J_{v1} and J_{v2} represent the components of the end-effector's linear velocity due to the angular velocity of the first shoulder joint and the angular velocity of the second elbow joint, respectively. These components of the linear velocity can be derived as follows:

$$J_{v1} = Z_0 \times (O_2 - O_0) = \begin{bmatrix} i & j & k\\ 0 & 0 & 1\\ a_2c_{12} + a_1c_1 & a_2s_{12} + a_1s_1 & 0\end{bmatrix}$$

$$= (-a_2s_{12} - a_1s_1)\hat{\imath} + (a_2c_{12} + a_1c_1)\hat{\jmath} + 0\hat{k}$$

$$J_{v2} = Z_1 \times (O_2 - O_1) = \begin{bmatrix} i & j & k\\ 0 & 0 & 1\\ a_2c_{12} & a_2s_{12} & 0\end{bmatrix}$$

$$= -a_2s_{12}\hat{\imath} + a_2c_{12}\hat{\jmath} + 0\hat{k}$$

$$Therefore, \qquad J_v = \begin{bmatrix}(-a_2s_{12} - a_1s_1) & -a_2s_{12}\\ (a_2c_{12} + a_1c_1) & a_2c_{12}\\ 0 & 0\end{bmatrix} --------(2.8)$$

$$, \ J_\omega = [Z_0 \quad Z_1] = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix} ---------(2.9)$$

Equations (2.8) and (2.9) constitute the complete Jacobian matrix at the end-effector of the 2-DOF robotic arm. As mentioned earlier, the singular configurations of a robotic manipulator occur when the joint angular velocities infinite values. This can help to find the conditions of the singular configurations of the 2-DOF robotic arm as follows:

Equation (2.6) implies that:

$$\dot{\theta} = J_v^{-1} v$$

This means that the joint angular velocities become infinite when the determinant of the Jacobian matrix component J_v becomes zero. Therefore, the singular configurations of the 2-DOF robotic arm can be calculated by finding the determinant of J_v in equation (2.8) and equalizing it to zero. However, the dimension of J_v is 3×2 which is not the dimension of a square matrix for which a determinant exists. Therefore, in order to avoid this problem, we have to avoid the last zero row of the matrix J_v. This can be possibly done since the linear velocity of the arm's end-effector does not have a component in the Z-direction. This is because the end-effector moves in a planar motion. Therefore,

$$|J_v| = \begin{vmatrix} (-a_2 s_{12} - a_1 s_1) & -a_2 s_{12} \\ (a_2 c_{12} + a_1 c_1) & a_2 c_{12} \end{vmatrix} = 0$$

Then, $\quad -a_2^2 c_{12} s_{12} - a_1 a_2 s_1 c_{12} + a_2^2 s_{12} c_{12} + a_1 a_2 c_1 s_{12} = 0$

$$Therefore, \ a_1 a_2 \sin(\theta_1 + \theta_2 - \theta_1) = 0$$

This implies that the 2-DOF robotic arm is considered to be in a singular configuration when the angular position of the second elbow joint θ_2 is either 0^0 or 180^0.

2.3 Dynamics

The dynamics of a robotic manipulator plays a key role in forming a foundation of understanding of the robot motion control. In order to build a control mechanism to control the motion of a robotic manipulator, the relationships between the necessary torques and forces that must be applied to the joints and the angular positions, velocities, and accelerations of those joints must be established and thouroughly understood. This is because any robot motion control strategy is based upon the generation of an actuating torque to move the robot joints to a desired space configuration. The problem concerned with establishing the relationships between the joint torques and forces and their angular positions, velocities, and accelerations can be solved using the manipulator dynamics equations [2]. Therefore, this section is devoted to the derivation of the dynamics equations of the 2-DOF robotic arm shown in figure (2.2).

There are two different methods used to derive the forward dynamics equations of a robotic manipulator: Euler-Lagrange method and Newton-Euler method [2]. The Euler-Lagrange method depends on finding the total kinetic and potential energies of the manipulator and then using them to calculate the Lagrangian (L) of the whole robot system which can be used to calculate the torque of each robot joint. The Newton-Euler method depends on the concept of Newton's second law to calculate the coupling torques between the adjacent links of the manipulator.

In the following, the first (Euler-Lagrange) method is utilized for deriving the dynamics equations of the 2-DOF robotic arm.

A closed-form relationship that can be used to calculate the torques and forces applied on the joints of a robotic manipulator is written as follows [2]:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_j} - \frac{\partial L}{\partial q_j} = \tau_j \quad ------------(2.10)$$

Where, L denotes the Lagrangian of the robotic manipulator and is defined as the difference between the total kinetic energy (K) and the total potential energy (P) of the robotic manipulator, q_j denotes the j^{th} joint variable, and τ_j denotes the torque or force applied on the j^{th} joint.

In order to find the Lagrangian of the 2-DOF robotic arm, the total kinetic and potential energies of the arm must be derived. The total kinetic and potential energies are defined as the sum of the kinetic and potential energies of the individual links of the manipulator. The kinetic and potential energy associated with one link are computed at the center of mass of that link. The derivation of the total kinetic and potential energies of the 2-DOF robotic arm is done as follows:

The first link:

The following equation describes the kinetic energy computed at the center of mass of the first link of the arm:

$$K_1 = \frac{1}{2}m_1 v_1^T v_1 + \frac{1}{2}\omega_1^T I_1 \omega_1 \quad ---------- (2.11)$$

Where m_1 denotes the mass concentrated at the center of the first link, v_1 is the vector of the linear velocity at the center of mass of the first link, ω_1 is the vector of the angular velocity at the center of mass of the first link, I_1denotes the inertia tensor matrix computed at the center of mass of the first link with respect to the inertial frame of the robotic arm.

The linear velocity v_1 can be derived using the equation (2.6) as follows:

$$v_1 = J_{v1}\dot{\theta} = [J_{v11} \quad J_{v12}]\begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix} --------- (2.12)$$

Where J_{v11} and J_{v12} denote the components of the linear velocity v_1 due to the angular velocity of the first joint $\dot{\theta}_1$ and the angular velocity of the second joint $\dot{\theta}_2$, respectively.

By considering the center of mass of each link is at the center of the link, the linear velocity components J_{v11} and J_{v12} can be calculated as follows:

$$J_{v11} = Z_0 \times (O_{c1} - O_0) = \begin{bmatrix} i & j & k \\ 0 & 0 & 1 \\ \frac{a_1}{2}c_1 & \frac{a_1}{2}s_1 & 0 \end{bmatrix}$$

$$= -\frac{a_1}{2}s_1\hat{\imath} + \frac{a_1}{2}c_1\hat{\jmath} + 0\hat{k}$$

Since the sole movement of the second elbow joint of the robotic arm does not affect the first link, the component added to the linear velocity at the center of the first link due to the angular velocity of the second elbow joint will be zero. This implies that:

$$J_{v12} = 0\hat{\imath} + 0\hat{\jmath} + 0\hat{k}$$

Therefore, $v_1 = \begin{bmatrix} -\frac{a_1}{2}s_1 & 0 \\ \frac{a_1}{2}c_1 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix} = \begin{bmatrix} -\frac{a_1}{2}s_1\dot{\theta}_1 \\ \frac{a_1}{2}c_1\dot{\theta}_1 \\ 0 \end{bmatrix} ------ (2.13)$

When the first shoulder joint moves with a specific angular velocity $\dot{\theta}_1$, every point on the first link will move with the same angular velocity $\dot{\theta}_1$. However, the sole movement of the second elbow joint does not affect the first link. Therefore, the angular velocity at the center of mass of the first link is given as follows:

$$\omega_1 = J_{\omega 1}\dot{\theta} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix} - - - - - - - - (2.14)$$

The computation of the inertia tensor matrix of a link depends on the geometrical dimensions of that link as well as on the coordinate frame with respect to which the computation is carried out. Therefore, in order to compute the inertia tensor matrix at the center of mass of the first link of the robotic arm, the geometrical dimensions of the first link must be determined and a local coordinate frame must be attached to the center of the link in order to derive the inertia tensor matrix with respect to it. Figure (2.4) shows a schematic diagram illustrating the geometrical shape of the first link. For simplicity of calculation, we will consider both links of the robotic arm to have cubic shapes. Other geometrical shapes may be considered but the calculation of the inertia tensor matrix must differ accordingly.

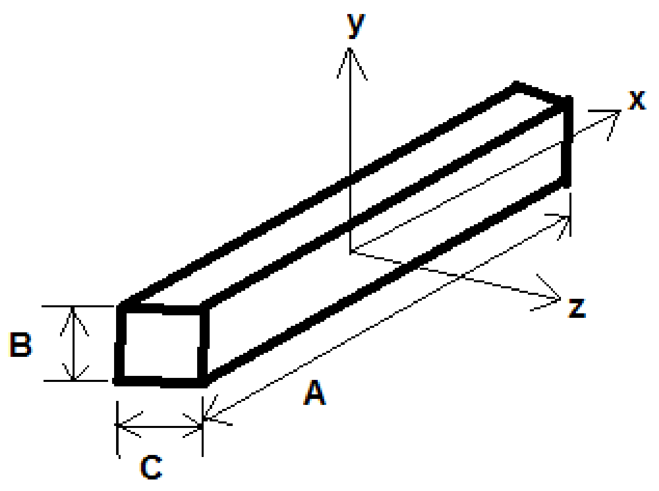


Figure 2.4: A schematic diagram of the first link of the 2-DOF robotic arm

As shown in figure (2.4), a local coordinate frame xyz is attached at the center of the link in order to use it for calculating the inertia tensor matrix. The inertia tensor matrix computed at the center of the first link with respect to the local frame xyz is given as:

$$I_{c1} = \begin{bmatrix} I_{xx1} & I_{xy1} & I_{xz1} \\ I_{yx1} & I_{yy1} & I_{yz1} \\ I_{zx1} & I_{zy1} & I_{zz1} \end{bmatrix} - - - - - - - - -(2.15)$$

Where the diagonal elements I_{xx1} , I_{yy1}, and I_{zz1} are called the principal moments of inertia, and the other elements of the inertia tensor matrix are called the cross products of inertia. Based on the schematic diagram shown in figure (2.4), and given that the mass density of the first link is denoted by ρ, the elements of the inertia tensor matrix I_{c1} can be calculated as follows:

$$I_{xx1} = \int_{-C/2}^{C/2} \int_{-B/2}^{B/2} \int_{-A/2}^{A/2} (y^2 + z^2)\rho dxdydz$$

$$= \int_{-C/2}^{C/2} \int_{-B/2}^{B/2} A\,(y^2 + z^2)\rho\; dydz$$

$$= \int_{-C/2}^{C/2} A\rho \left[\frac{y^3}{3} + z^2 y\right]_{-B/2}^{B/2} dz$$

$$= \int_{-C/2}^{C/2} A\rho [\frac{B^3}{24} + z^2\frac{B}{2} + \frac{B^3}{24} + z^2\frac{B}{2}]\; dz$$

$$= \int_{-C/2}^{-C/2} A\rho \left[\frac{B^3}{12} + z^2 B\right] dz = \left[\frac{B^3}{12} z + \frac{z^3}{3} B\right]_{-C/2}^{C/2}$$

$$= \rho\frac{ABC}{12}(B^2 + C^2)$$

By doing similar calculations, the other elements of the inertia tensor matrix I_{c1} can be calculated to have the following values:

$$I_{yy1} = \int_{-C/2}^{C/2} \int_{-B/2}^{B/2} \int_{-A/2}^{A/2} (x^2 + z^2)\rho dxdydz = \rho\frac{ABC}{12}(C^2 + A^2)$$

$$I_{zz1} = \int_{-C/2}^{C/2} \int_{-B/2}^{B/2} \int_{-A/2}^{A/2} (x^2 + y^2)\rho dxdydz = \rho\frac{ABC}{12}(B^2 + A^2)$$

$$I_{xy1} = I_{yx1} = -\int_{-C/2}^{C/2} \int_{-B/2}^{B/2} \int_{-A/2}^{A/2} xy\rho dxdydz = 0$$

$$I_{xz1} = I_{zx1} = -\int_{-C/2}^{C/2} \int_{-B/2}^{B/2} \int_{-A/2}^{A/2} xz\rho dxdydz = 0$$

$$I_{yz1} = I_{zy1} = -\int_{-C/2}^{C/2} \int_{-B/2}^{B/2} \int_{-A/2}^{A/2} yz\rho dxdydz = 0$$

It is clear from the definition of the cross products of inertia that the inertia tensor matrix of a link is symmetric. The cross products of inertia for the first link are found to be zero due to the geometrical symmetry of the link about the local frame xyz as shown in figure (2.4).

By substituting the elements into the equation (2.15), the inertia tensor matrix of the first link with respect to the local frame xyz is found to be:

$$I_{c1} = \begin{bmatrix} \rho\frac{ABC}{12}(B^2+C^2) & 0 & 0 \\ 0 & \rho\frac{ABC}{12}(C^2+A^2) & 0 \\ 0 & 0 & \rho\frac{ABC}{12}(B^2+A^2) \end{bmatrix} ----(2.16)$$

In order to compute the kinetic energy defined in (2.11), the inertia tensor matrix at the center of mass of the first link must be computed with respect to the inertial frame of the robotic arm shown in figure (2.2). This requires the similarity transformation of the locally-defined inertia matrix (2.16) which can be done as follows:

$$I_1 = R_{c1} I_{c1} {R_{c1}}^T$$

Where, R_{c1} represents the rotational matrix of the local frame xyz attached at the center of the first link with respect to the inertial frame of the 2-DOF robotic arm and is given by:

$$R_{c1} = \begin{bmatrix} c_1 & -s_1 & 0 \\ s_1 & c_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore,

$$I_1 = \begin{bmatrix} \rho\frac{ABC}{12}[(B^2+C^2){c_1}^2 + (A^2+C^2){s_1}^2] & \rho\frac{ABC}{12}(B^2-A^2)s_1c_1 & 0 \\ \rho\frac{ABC}{12}(B^2-A^2)s_1c_1 & \rho\frac{ABC}{12}[(B^2+C^2){s_1}^2 + (A^2+C^2){c_1}^2] & 0 \\ 0 & 0 & \rho\frac{ABC}{12}(B^2+A^2) \end{bmatrix} --------(2.17)$$

Substituting the equations (2.13), (2.14), and (2.17) into the equation (2.11) gives the kinetic energy at the center of mass of the first link of the 2-DOF robotic arm as follows:

$$K_1 = \frac{m_1}{2}\frac{{a_1}^2}{4}{\dot{\theta}_1}^2 + \rho\frac{ABC}{24}(B^2+A^2){\dot{\theta}_1}^2 ------(2.18)$$

The following equation is used to compute the potential energy at the center of mass of the first link of the arm [2]:

$P_1 = g^T r_{c1} m_1$

Where g is the vector of gravity in the inertial frame of the robotic arm, $r_{c1}$is the position vector of the center of mass of the first link which was considered earlier to be at the center of the link, and m_1 is the mass of the first link.

Therefore,

$$P_1 = \begin{bmatrix} 0 & g & 0 \end{bmatrix} \begin{bmatrix} \frac{a_1}{2} c_1 \\ \frac{a_1}{2} s_1 \\ 0 \end{bmatrix} m_1 = m_1 g \frac{a_1}{2} s_1 ------(2.19)$$

<u>The second link:</u>

Following the same steps of computing the kinetic and potential energies of the first link, the kinetic and potential energies at the center of mass of the second link are computed as follows:

$$K_2 = \tfrac{1}{2} m_2 v_2^T v_2 + \tfrac{1}{2} \omega_2^T I_2 \omega_2 -----------(2.20)$$

Where m_2 denotes the mass concentrated at the center of the second link, v_2 is the vector of the linear velocity at the center of mass of the second link, ω_2 is the vector of the angular velocity at the center of mass of the second link, I_2denotes the inertia tensor matrix computed at the center of mass of the second link with respect to the inertial frame of the robotic arm.

$$v_2 = J_{v2}\dot{\theta} = \begin{bmatrix} J_{v21} & J_{v22} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} \frac{-a_2}{2} s_{12} - a_1 s_1 & \frac{-a_2}{2} s_{12} \\ \frac{a_2}{2} c_{12} + a_1 c_1 & \frac{a_2}{2} c_{12} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

Therefore,

$$v_2 = \begin{bmatrix} \left(\frac{-a_2}{2} s_{12} - a_1 s_1\right) \dot{\theta}_1 - \frac{a_2}{2} s_{12} \dot{\theta}_2 \\ \left(\frac{a_2}{2} c_{12} + a_1 c_1\right) \dot{\theta}_1 + \frac{a_2}{2} c_{12} \dot{\theta}_2 \\ 0 \end{bmatrix} ------(2.21)$$

It is clear from the equation (2.21) that the angular velocities of both joints of the 2-DOF robotic arm have effects on the linear velocity computed at the center of mass of the second link.

$$\omega_2 = J_{\omega_2}\dot{\theta} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 + \dot{\theta}_2 \end{bmatrix} - - - - - - - - (2.22)$$

By considering that both links of the robotic arm have the same geometrical shape and dimensions, it can be shown that both links will have the same inertia tensor matrix relative to the local coordinate frames attached at the centers of the links. This implies that the inertia tensor matrix at the center of the second link I_{c1} is the same as the one derived in (2.16). Therefore,

$$I_{c2} = \begin{bmatrix} \rho\frac{ABC}{12}(B^2 + C^2) & 0 & 0 \\ 0 & \rho\frac{ABC}{12}(C^2 + A^2) & 0 \\ 0 & 0 & \rho\frac{ABC}{12}(B^2 + A^2) \end{bmatrix}$$

By doing the similarity transformation of I_{c2}, the inertia tensor matrix at the center of the second link with respect to the inertial frame of the arm I_2 is found to be as follows:

$$I_2 = R_{c2}I_{c2}{R_{c2}}^T$$

Where, R_{c2} represents the rotational matrix of the local frame xyz attached at the center of the second link with respect to the inertial frame of the 2-DOF robotic arm and is given by:

$$R_{c2} = \begin{bmatrix} c_{12} & -s_{12} & 0 \\ s_{12} & c_{12} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore,

$$I_2 =$$

$$\begin{bmatrix} \rho\frac{ABC}{12}[(B^2 + C^2){c_{12}}^2 + (A^2 + C^2){s_{12}}^2] & \rho\frac{ABC}{12}(B^2 - A^2)s_{12}c_{12} & 0 \\ \rho\frac{ABC}{12}(B^2 - A^2)s_{12}c_{12} & \rho\frac{ABC}{12}[(B^2 + C^2){s_{12}}^2 + (A^2 + C^2){c_{12}}^2] & 0 \\ 0 & 0 & \rho\frac{ABC}{12}(B^2 + A^2) \end{bmatrix}$$

$$- - - - - - - - - - (2.23)$$

Substituting the equations (2.21), (2.22), and (2.23) into the equation (2.20) gives the kinetic energy at the center of mass of the second link of the 2-DOF robotic arm as follows:

$$K_2 = \frac{m_2}{2}\left[\left(-\left(\frac{a_2}{2}s_{12} + a_1 s_1\right)\dot{\theta}_1 - \frac{a_2}{2}s_{12}\dot{\theta}_2\right)^2 + \left(\left(\frac{a_2}{2}c_{12} + a_1 c_1\right)\dot{\theta}_1 + \frac{a_2}{2}c_{12}\dot{\theta}_2\right)^2\right] + \rho\frac{ABC}{24}(A^2 + B^2)\left(\dot{\theta}_1 + \dot{\theta}_2\right)^2 ----(2.24)$$

The potential energy at the center of mass of the second link of the 2-DOF robotic arm is computed as follows:

$$P_2 = g^T r_{c2} m_2$$

Where g is the vector of gravity in the inertial frame of the robotic arm, r_{c2} is the position vector of the center of mass of the second link, and m_2 is the mass of the second link.

Therefore,

$$P_2 = \begin{bmatrix} 0 & g & 0 \end{bmatrix}\begin{bmatrix} \frac{a_2}{2}c_{12} + a_1 c_1 \\ \frac{a_2}{2}s_{12} + a_1 s_1 \\ 0 \end{bmatrix} m_2 = m_2 g\left(\frac{a_2}{2}s_{12} + a_1 s_1\right) ----(2.25)$$

By adding the equations (2.18) and (2.24), the total kinetic energy of the 2-DOF robotic arm is found to be:

$$K = K_1 + K_2 = \frac{m_1}{2}\frac{{a_1}^2}{4}{\dot{\theta}_1}^2 + \rho\frac{ABC}{24}(B^2 + A^2){\dot{\theta}_1}^2 + \frac{m_2}{2}\left[\left(-\left(\frac{a_2}{2}s_{12} + a_1 s_1\right)\dot{\theta}_1 - \frac{a_2}{2}s_{12}\dot{\theta}_2\right)^2 + \left(\left(\frac{a_2}{2}c_{12} + a_1 c_1\right)\dot{\theta}_1 + \frac{a_2}{2}c_{12}\dot{\theta}_2\right)^2\right] + \rho\frac{ABC}{24}(A^2 + B^2)\left(\dot{\theta}_1 + \dot{\theta}_2\right)^2 ----- (2.26)$$

By adding the equations (2.19) and (2.25), the total potential energy of the 2-DOF robotic arm is found to be:

$$P = P_1 + P_2 = m_1 g\frac{a_1}{2}s_1 + m_2 g\left(\frac{a_2}{2}s_{12} + a_1 s_1\right) ------(2.27)$$

After computing the total kinetic and potential energies of the 2-DOF robotic arm, the Lagrangian of the arm can be computed as follows:

$$L = K - P = \frac{m_1}{2}\frac{a_1{}^2}{4}\dot{\theta}_1{}^2 + \rho\frac{ABC}{24}(B^2 + A^2)\dot{\theta}_1{}^2$$
$$+ \frac{m_2}{2}\left[\left(-\left(\frac{a_2}{2}s_{12} + a_1 s_1\right)\dot{\theta}_1 - \frac{a_2}{2}s_{12}\dot{\theta}_2\right)^2 + \left(\left(\frac{a_2}{2}c_{12} + a_1 c_1\right)\dot{\theta}_1\right.\right.$$
$$\left.\left. + \frac{a_2}{2}c_{12}\dot{\theta}_2\right)^2\right] + \rho\frac{ABC}{24}(A^2 + B^2)(\dot{\theta}_1 + \dot{\theta}_2)^2$$
$$- \left(m_1 g\frac{a_1}{2}s_1 + m_2 g\left(\frac{a_2}{2}s_{12} + a_1 s_1\right)\right)$$

By using the Euler-Lagrange equations defined in (2.10), the torques applied to the revolute joints of the 2-DOF robotic arm for specified angular positions θ, angular velocities $\dot{\theta}$, and angular accelerations $\ddot{\theta}$ can be computed and found to be:

$$\tau_1 = \left[m_1\frac{a_1{}^2}{4} + m_2\left(\frac{a_2{}^2}{4} + a_1{}^2 + a_1 a_2 c_2\right) + \rho\frac{ABC}{6}(A^2 + B^2)\right]\ddot{\theta}_1$$
$$+ \left[m_2\left(\frac{a_2{}^2}{4} + \frac{a_1 a_2}{2}c_2\right) + \rho\frac{ABC}{12}(A^2 + B^2)\right]\ddot{\theta}_2 + [-m_2 a_1 a_2 s_2]\dot{\theta}_1\dot{\theta}_2$$
$$+ \left[-m_2 a_1\frac{a_2}{2}s_2\right]\dot{\theta}_2{}^2 + \left[m_1 g\frac{a_1}{2}c_1 + m_2 g\left(\frac{a_2}{2}c_{12} + a_1 c_1\right)\right] - -(2.28)$$

$$\tau_2 = \left[m_2\left(\frac{a_2{}^2}{4} + \frac{a_1 a_2}{2}c_2\right) + \rho\frac{ABC}{12}(A^2 + B^2)\right]\ddot{\theta}_1 + \left[\frac{a_2{}^2}{4}m_2 + \rho\frac{ABC}{12}(A^2 + B^2)\right]\ddot{\theta}_2$$
$$+ \left[m_2\frac{a_1 a_2}{2}s_2\right]\dot{\theta}_1{}^2 + \left[m_2 g\frac{a_2}{2}c_{12}\right] - - - - - - - - - - (2.29)$$

2.4 Modeling of the 2-DOF robotic arm on Dymola

In this section, a model is developed for the 2-DOF robotic arm shown in figure (2.2) under the Dymola simulation environment [14]. The reason behind choosing Dymola for simulation in this thesis rather than MATLAB is because of its dependence on the object-oriented modeling language *Modelica* developed by Hilding Elmqvist in 1978 [15]. This language is useful for modeling any complex system by decomposing the system into individual simple component models (*objects*) and then connecting such models with suitable connectors to develop a model for the whole system. This object-oriented modeling methodology does not require solving highly complicated nonlinear equations in order to derive a closed-form mathematical model (such as a transfer function) for the system as MATLAB requires [16]. In addition, using simple differential algebraic equations (DAEs) to develop a modelica-model allows for modeling complex systems containing components from different domains of engineering [14,15].

The development of a modelica-model for the 2-DOF robotic arm on Dymola requires the use of some of the component models packaged within the *Multibody* library which is one of the several built-in libraries available in Dymola [14]. The main components contained within the *Multibody* library are described as follows:

- **World**
  The world component is used to model the inertial (world) frame of a multi-body mechanical system such as a robotic manipulator. The world model defines parameters for the gravity vector, as well as the properties needed for developing an animation for the whole system. Therefore, the presence of this component at the top level of the model of a multi-body system is a necessity and its absence causes the translator (compiler) to generate an error.

- **BodyShape**
  This component is used to model a rigid body characterized by a mass and an inertia tensor such as a link of a robotic manipulator. The component model defines parameters that allow for choosing a desired geometrical shape with its dimensions as well as defining the coordinates of the center of mass of the body relative to the frame attached at its starting terminal. The frames attached at the ends of the BodyShape component allow for determining the position and orientation of the body with respect to one of the two frames. The beauty of using the BodyShape model appears when a number of BodyShape components are connected. In this case, the kinematics problem of the whole system is automatically solved. This component will be used later for the modeling of the two links of the robotic arm.

- **Revolute**
  The Revolute model is one of the models available within the *Joints* library. This model, as its name implies, is used for the modeling of a revolute joint. The parameters defined in the Revolute model include the angular position of the revolute joint, the geometrical shape used to visualize the revolute joint such as a small cylinder. The Revolute model will be used to define the revolute joints of the 2-DOF robotic arm.

There are many other built-in libraries and components defined within the Multibody library of Dymola. Information regarding the definitions and purposes of use of such components can be obtained from [14]. Figure (2.5) shows a diagram illustrating the use of the above components to build the model of the 2-DOF robotic arm on Dymola.

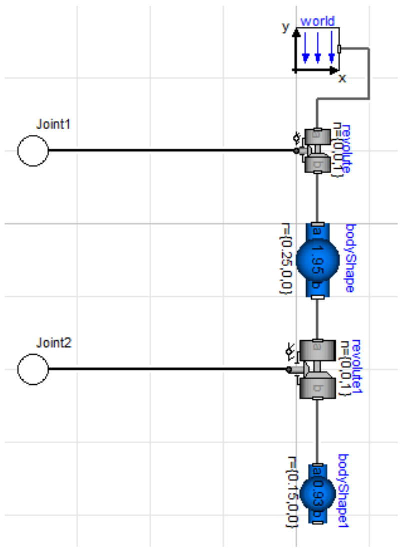


Figure 2.5: Dymola model for the 2-DOF robotic arm

2.5 Permanent magnet DC- motor

This section is devoted to the study and analysis of the dynamic behavior of the permanent-magnet DC-motor which is considered to be the driving force of the 2-DOF robotic manipulator. In order to facilitate the process of understanding how a DC-motor works, a mathematical model for the DC-motor is first developed. After developing a mathematical model, the stability and other characteristics of the time response of the DC-motor system can be easily observed and analyzed for different inputs and load values.

The conventional model used to describe the dynamics of a permanent-magnet DC motor is shown in figure (2.6) [2].

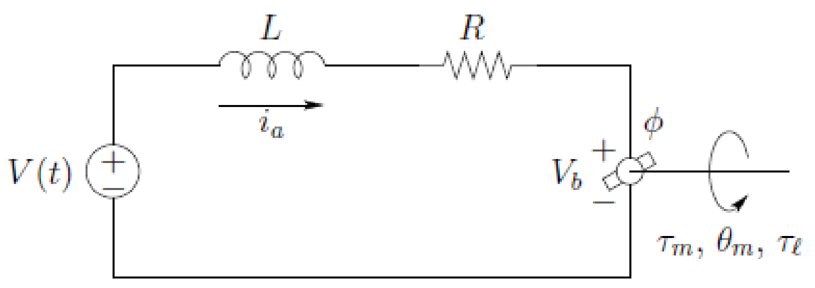


Figure 2.6: A circuit diagram for armature-controlled DC motor

It is clear from figure (2.6) that the model of a permanent-magnet DC motor consists of two parts: an electrical part represented by the armature circuit and a mechanical part. The components of the armature circuit shown in figure (2.6) include an armature inductance L, an armature resistance R, and an input voltage signal source $V(t)$. i_a represents the armature current.

The mechanical part of the DC-motor model is shown in figure (2.7). It is shown in this figure that the mechanical part of the DC-motor model consists of the motor inertia J_m, the motor damper B_m, and the load inertia which is connected to the motor inertia through a gear train with gear reduction ratio $(r{:}\,1)$. Both the motor inertia and the motor damper affect the amount of torque needed for the motor to operate.

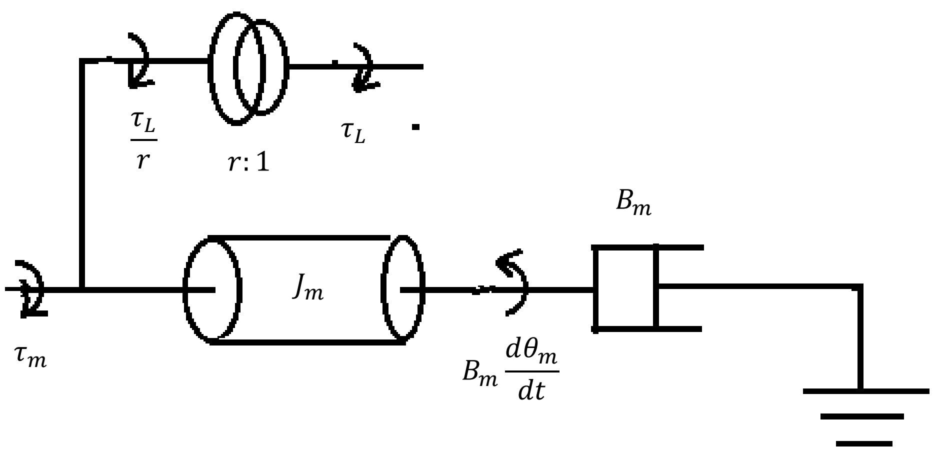


Figure 2.7: Mechanical part of the DC-motor system

The operation of a DC-motor is based on the effect of a constant magnetic field ϕ (passing through the motor stator) on a current-carrying conductor (motor rotor). Such effect is represented by the generation of torque τ_m to enforce the motor inertia to rotate with a certain angular velocity ω_m. The angular motion of the motor inertia within the constant magnetic field induces a voltage drop called *back-electromotive force* V_b across the terminals of the moving rotor as shown in figure (2.6).

The open-loop transfer function relating the angular position of the motor θ_m to the input armature voltage $V(t)$ is given by the following:

$$\frac{\theta_m(s)}{V(s)} = \frac{K_m}{(LS+R)(J_mS^2+B_mS)+K_bK_mS} \quad -------- \; (2.30)$$

Where K_m and K_b are the motor constants.

The mathematical derivation of the above transfer function is explained in detail in Appendix A. It is important to remind here that the derivation of a transfer function is not necessary for simulating the behavior of a DC motor model on Dymola because the simulation of a system on Dymola is based on the object-oriented and acausal modeling methodology. However, the importance of such a transfer function will appear in the process of designing a proportional-derivative (PD) controller for the motor angular position in the next chapter.

2.6 Dymola Modeling and simulation of the DC-motor

In this section, the DC-motor conventional model shown in figures (2.6) and (2.7) will be used to develop an object-oriented model for the motor on Dymola. The Dymola model will then be simulated to analyze the step response of the DC-motor in the presence and absence of a driven load. This simulation aims to investigate the stability of the motor system without a controller as well as the impact of driving a certain load on the motor angular position and velocity. Figure (2.8) shows the component-oriented model developed in Dymola for the DC-motor system.

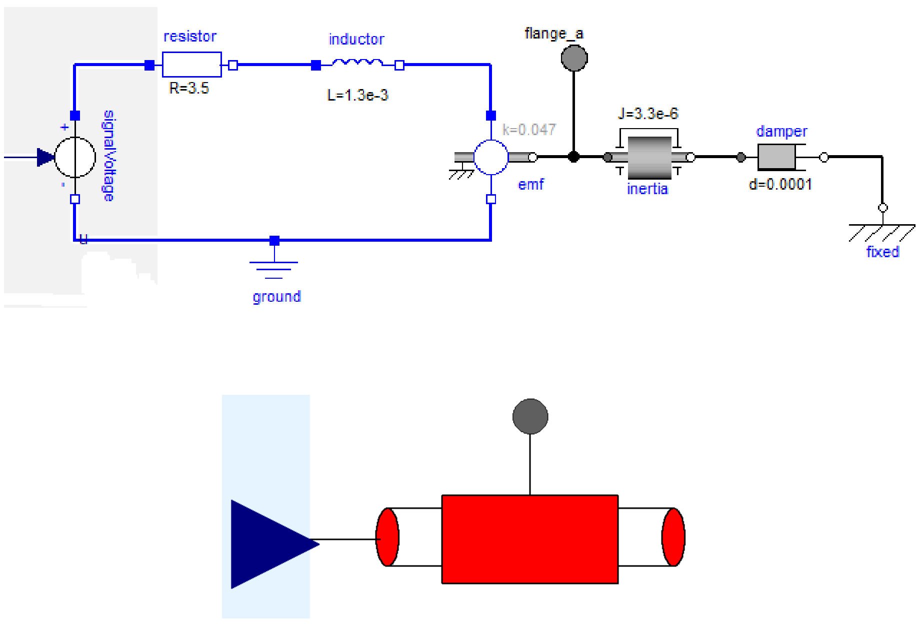


Figure 2.8: Dymola model for the DC-motor system

Figure (2.8) shows two diagrams. The upper one is the motor armature and mechanical circuits similar to those shown in figures (2.6) and (2.7). The lower diagram shows a simplified icon which represents an abstract model for the upper motor circuit diagram. The motor icon was developed in order to use it later in a more complex system structure which would contain the controllers and the 2-DOF robotic manipulator. The motor circuit diagram shown in figure (3.3) also shows the motor parameter values used in simulation which are $R = 3.5\Omega$, $L = 1.3\ mH$, $K_b = K_m = K = 0.047$, $J_m = 3.3 \times 10^{-6}\ Kg.m^2$, $B_m = 0.0001\ N.m.s/rad$.

In order to observe the effect of connecting a constant load torque on the step response of the motor angular position and velocity, the following simulation is conducted: In the beginning of the simulation time, no load is connected to the motor. During this time, the simulation will show the natural step response of the motor with no load (when $\tau_L = 0$). At time $t = 0.5\ sec$, a load torque ($\tau_L = 0.5\ N.m.$) is applied to the motor inertia. The simulation from this time on will show the effect of connecting the load on the motor step response. Figures (2.9)-(2.12) show the step responses of different characteristics of the DC-motor.

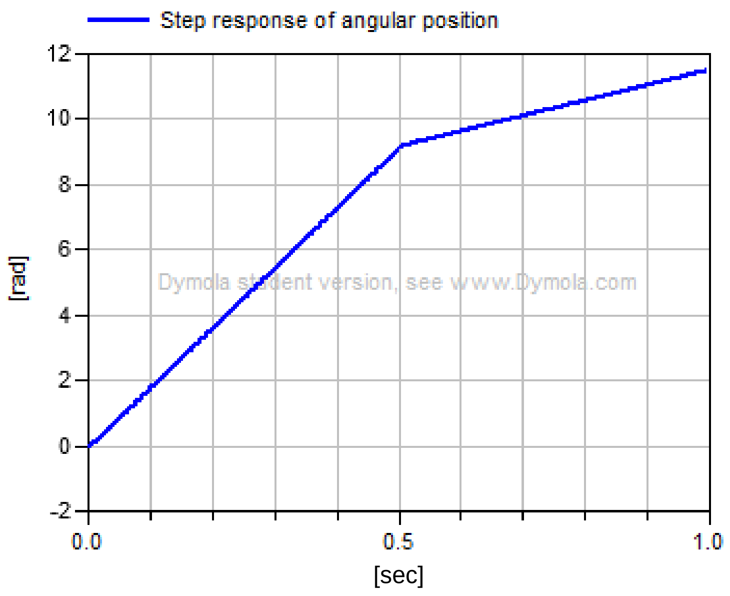


Figure 2.9: Step response of the DC motor angular position

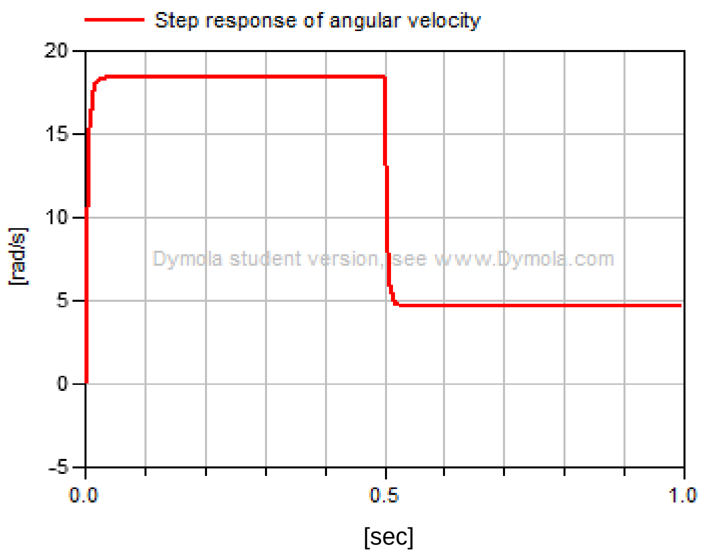


Figure 2.10: Step response of the DC motor angular velocity

Figure (2.9) shows that for time $0.03 < t \leq 0.5$, the motor angular position $\theta_m$starts increasing linearly with a slope of about 18.3 rad/sec which is the value of the motor angular velocity during the same time interval as shown in figure (2.10). For time $t > 0.5sec$ when the load is connected to the motor flange, the motor angular position $\theta_m$starts increasing linearly with a slower slope of about 4.7 rad/sec which is the value of the slower angular velocity for the same time interval as shown in figure (2.10).

Figure (2.11) shows that the armature current when no load is connected to the motor is about 0.04 A. However, when the load is connected to the motor at $t = 0.5sec$, the armature current starts increasing to the value of 0.22 A.

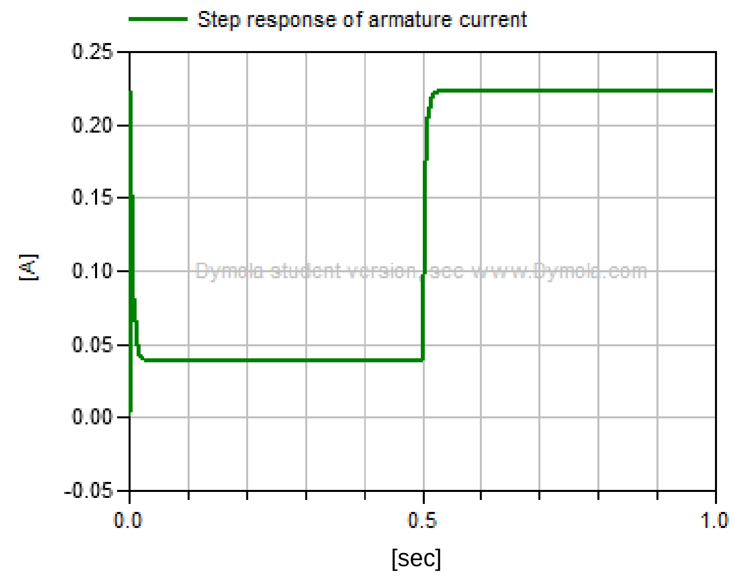


Figure 2.11: Step response of the DC motor armature current

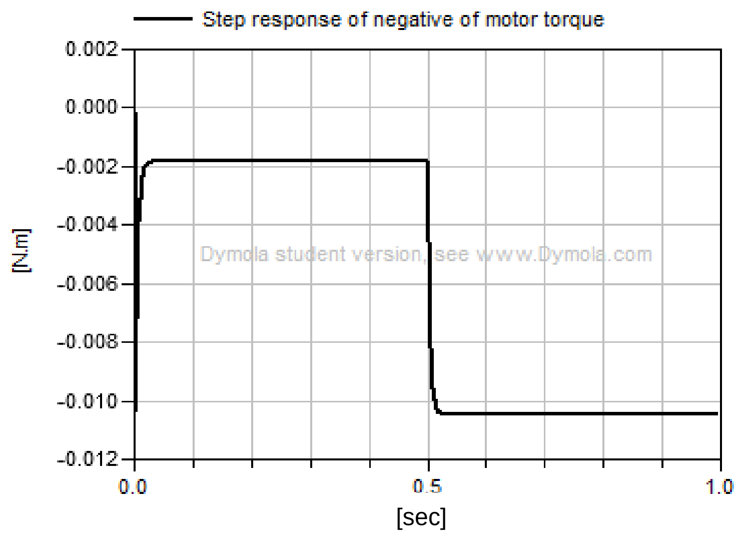


Figure 2.12: Step response of the negative of DC motor torque

Such an increase in the armature current is logically necessary in order to generate a larger torque for driving both the motor and the load inertias together. Since the motor torque is directly proportional to the armature current through the motor constant K_m, the increase of the armature current at time $t = 0.5sec$ causes a corresponding increase in the motor torque $\tau_m$as shown in figure (2.12) in which the torque increases from about 0.001 $N.m.$ to about 0.01 $N.m.$

One important indication of figure (2.9) is the unstable behavior of the motor angular position which keeps increasing linearly and never stabilizes at a certain value which is not practically desired.

In order to observe the effect of the unstable behavior of the DC motor on the motion of the 2-DOF robotic arm, the following simulation is conducted: DC-motors are connected to the joints of the 2-DOF robotic manipulator modeled in figure (2.5) through gear trains with $r_1 = 90$ and $r_2 = 220$. The step responses of the angular positions and torques of both joints of the arm are shown in figures (2.13) – (2.16).

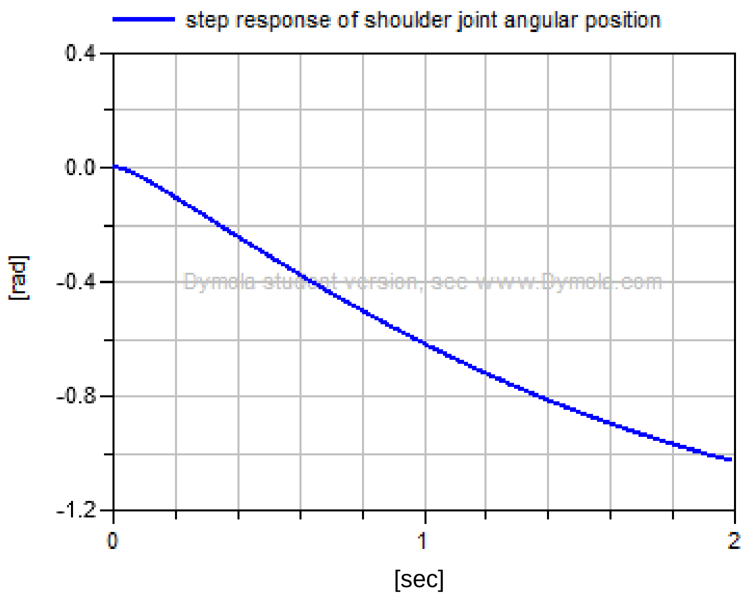


Figure 2.13: Step response of uncontrolled shoulder joint angular position

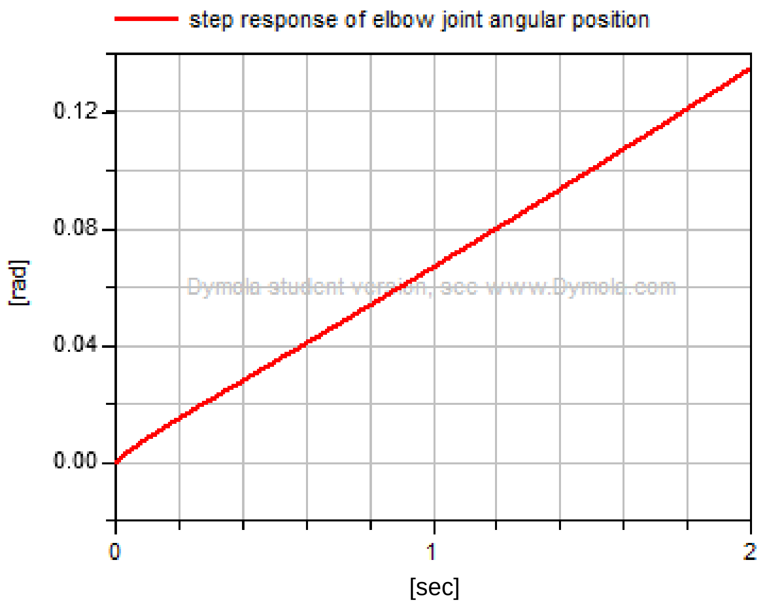


Figure 2.14: Step response of uncontrolled elbow joint angular position

It is clear from figures (2.13) and (2.14) that the angular positions of both joints of the arm θ_1 and θ_2 are unstable, as expected, due to the unstable angular positions of their driving motors θ_{m1} and θ_{m2}. It is also clear that the angular position of the shoulder joint θ_1 is more affected by its nonlinear dynamic torque τ_1 than the angular position of the elbow joint θ_2. This is because the shoulder joint exerts a greater torque in order to be able to drive both links of the manipulator. The elbow joint, however, needs to exert a lower torque because it drives only one link of the manipulator.

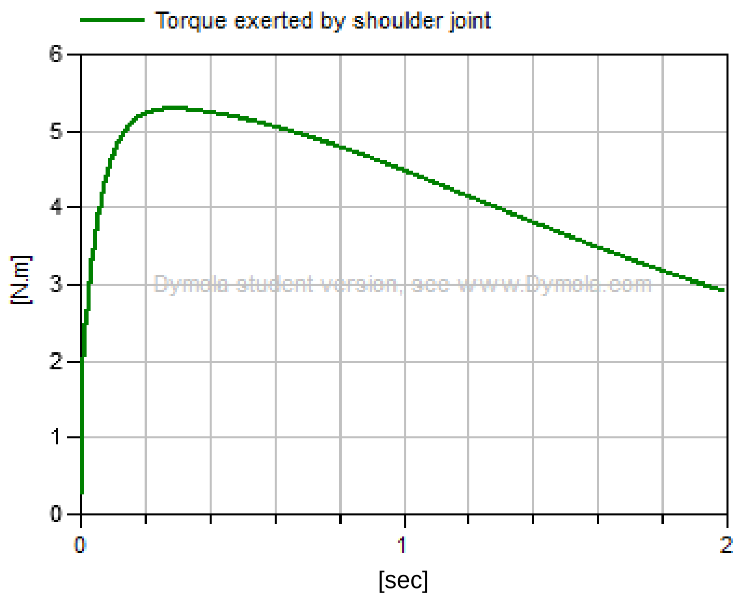


Figure 2.15: Step response of uncontrolled shoulder joint torque

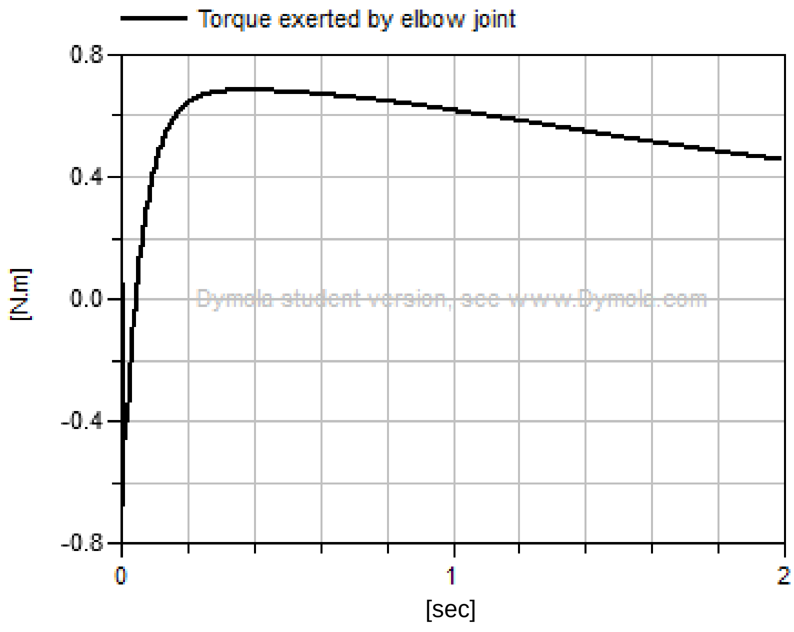


Figure 2.16: Step response of uncontrolled elbow joint torque

To see this in numbers, the value of the torque generated by the shoulder joint at time $t = 0.5\, sec$ is $\tau_1 = 5.2\, N.m$. This would mean that the disturbance load torque applied on the inertia of the shoulder joint driving motor is $\frac{\tau_1}{r_1} = \frac{5.2}{90} = 0.06\ N.m$. From figure (2.12), the motor torque generated when no load is connected is found to be about $0.002\ N.m$. Therefore, the net torque applied on the inertia of the shoulder joint driving motor is $0.002 - 0.06 = -0.058\ N.m$. Such a negative torque is the reason why the shoulder joint angular position is always negative as shown in figure (2.13).

On the other hand, the elbow joint exerts a lower torque of about $\tau_2 = 0.68\ N.m$. This implies that the disturbance load torque applied on the inertia of the elbow joint driving motor is $\frac{\tau_2}{r_2} = \frac{0.68}{220} = 0.003\ N.m$. Therefore, the net torque applied on the inertia of the elbow joint driving motor is $0.002 - 0.003 = -0.001\ N.m$. Despite the negative value of the net torque, its magnitude is very small that is not sufficient to enforce the motor angular position to have negative values. Moreover, the very small magnitude of the applied torque causes the elbow joint angular position to have very small values as shown in figure (2.14).

Figure (2.17) shows a (3D) visualization of the unstable behavior of the uncontrolled joint driving motors.

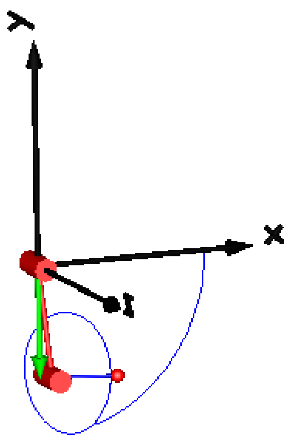


Figure 2.17: 3D visualization of the trajectory of the uncontrolled arm

Figure (2.17) shows the trajectory followed by the end-effector during a $200\ sec$ period. It is clearly seen that the angular positions of the joint driving motors never stabilize. At the outset, the shoulder joint starts rotating with negative angular positions, as indicated by figure (2.13), until it reaches the maximum position of $-83.52\ deg$ at the time instant $t = 22\ sec$. After that, the shoulder joint keeps oscillating between the positions of $-66.2\ deg$ and $-83.52\ deg$ for the rest of the simulation time. The angular position of the elbow joint, however, keeps increasing with positive values making an elliptical trajectory as shown in figure (2.17).

CHAPTER 3

PD-Computed Torque Control

It was shown in the previous chapter that the open-loop step response of the permanent-magnet DC motor is unstable. Such instability must therefore be corrected before using the motor to drive a load. In this chapter, the problem of stabilizing the angular position of the DC-motor is handled through designing a (PD) controller which is done using the root locus method. The computed torque methodology is then adopted in order to cancel the effects of the joint disturbance torques on the responses of the PD-controlled motors.

3.1 PD-control

There are different methods used for designing a PD controller for a specified system. One of these methods is based on a mathematical analysis which involves the derivation of the system transfer function and then using the transfer function of the PD-controller to achieve required time response characteristics [18].

Although this method seems to be simple for second-order systems, it becomes highly complicated for higher order systems for which the derivation of the closed-loop transfer function is complicated and a time consuming process. In order to facilitate the process of designing a PD controller for higher order systems, the root locus method was proposed as a technique that requires only the open-loop poles of the system [17,18]. The root-locus method depends on tracking the movement of the open-loop poles of the system on the complex number plane as the closed-loop system gain increases from 1 to infinity. The resultant trajectory of the poles on the complex number plane is called the root locus of the system. So, the root-locus based controller design process is mainly based on making the root locus of the system pass through a specified design point that achieves the required time response characteristics.

The general process of designing a PD controller for a system using the root locus method is explained in detail in Appendix B.

As mentioned earlier, the primary goal of designing a PD controller for the DC motor system is to achieve the stability of the motor angular position. The secondary goal of using a PD controller is to achieve specific requirements for the motor step response. These requirements are given as follows:

The settling time t_s is taken to be approximately $20\,mSec$ and the peak overshoot $P.O.$ is to be about 0.0001.

For these time response requirements, the design point S_D that must be located on the root locus of the DC-motor system is found to be the following:

$$S_D = -200 + j65.74 - - - - - - - - (3.1)$$

The computation of the above design point is found in Appendix B.

Using the equation (2.30) and the motor parameter values shown in figure (2.8), the open-loop transfer function $L(S)$ of the DC-motor can be calculated as follows:

$$L(S) = \frac{0.047}{(429 \times 10^{-11})S^3 + (1168 \times 10^{-8})S^2 + 0.002559S} - - - - (3.2)$$

Using the tf2zpk MATLAB function, the open-loop poles of the DC-motor system can be computed and are given as follows:

$$s_1 = 0 \quad , \quad s_2 = -0.2403 \times 10^3 \quad , \quad s_3 = 2.4823 \times 10^3$$

The general transfer function of a PD controller is given as $G_{PD} = K_P + K_D S$, where K_P and K_D are the PD controller gains. This implies that the design of a PD controller for a specific system involves inserting an additional zero into the open-loop transfer function of the system such that the root locus of the resulting open-loop system passes through the design point S_D in order to achieve the required time response characteristics. Applying such an implication is described in the following steps:

1- Let $(S + a)$ be the zero term to be added to the DC-motor open-loop transfer function $L(S)$ in (4.9). Therefore, the resulting open-loop transfer function would be $(S + a)L(S)$.

2- In order for the design pole S_D in (4.9) to be located on the root locus of the resulting open-loop transfer function in step 1, the magnitude and phase angle requirements derived in (4.2) must be satisfied. This enables to calculate the corresponding location of the additional zero on the real axis (a).

Applying steps 1 and 2 reveal that the additional zero $(S + a) = (S + 273.14)$.

Figure (3.1) shows the root locus of the resulting open-loop transfer function $(S + 273.14)L(S)$ which was obtained using the rlocus MATLAB function. It is clear from figure (4.1) that the root locus of the resulting open-loop transfer function did pass through the design pole S_D given in (3.1). The figure also shows that both the required damping ratio and un-damped frequency calculated in Appendix B have been reached. It is also important to note that the value of the closed loop gain at which the root locus passes through the design pole is $K = 0.0345$.

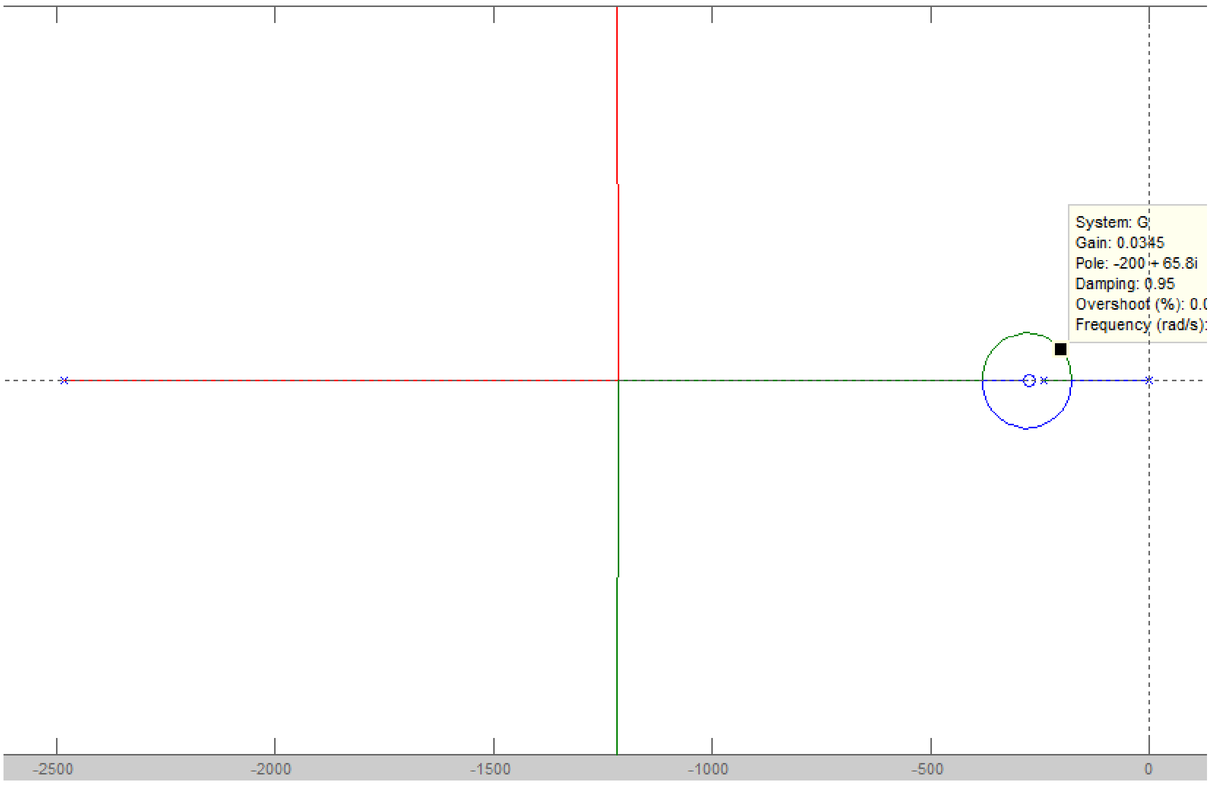


Figure 3.1: Root locus of the open-loop transfer function (S+273.14) L(S)

Therefore, the designed PD controller for the DC-motor system is $0.0345(S+273.14) = 9.42 + 0.0345\ S$ with the controller gains $K_P = 9.42$ and $K_D = 0.0345$.

In order to test the efficiency of the designed PD controller, a simulation is carried out to see its effect on the step response of the DC-motor angular position. Figure (3.2) shows the closed-loop DC-motor system including the designed PD controller and the DC-motor system model represented by the icon shown in figure (2.8). Figure (3.3) shows the step response of the PD-controlled motor system shown in figure (3.2).

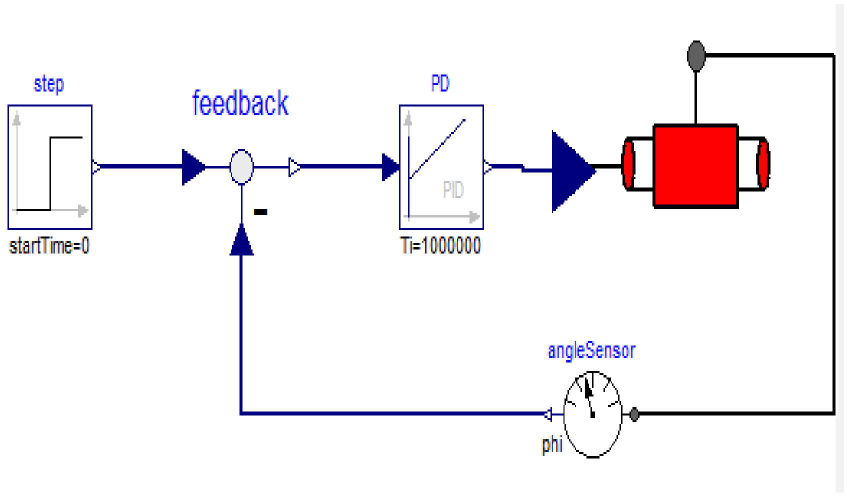


Figure 3.2: Block diagram of PD-controlled DC-motor system

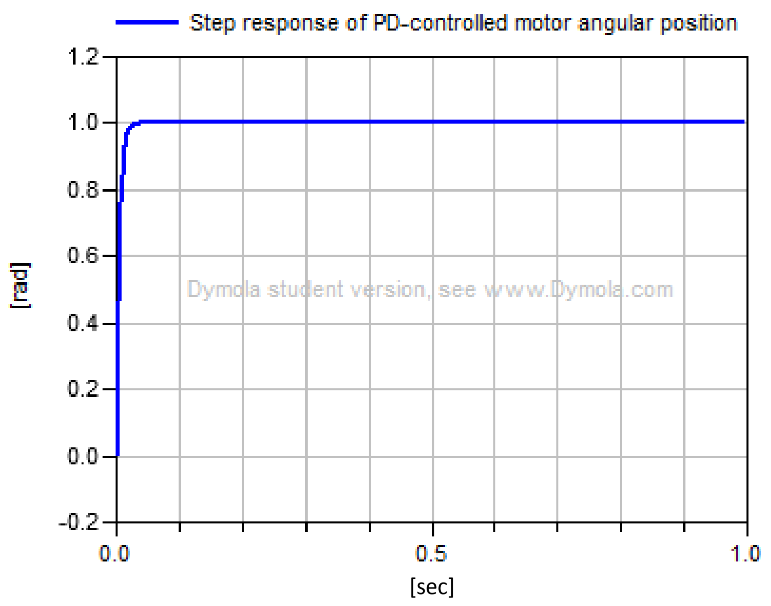


Figure 3.3: Step response of PD-controlled DC-motor

It can be seen from figure (3.3) that the designed PD-controller is capable of stabilizing the motor angular position θ_m and achieving the required settling time and overshoot specifications since $t_s \approx 0.02\ sec$ and the figure shows no overshoot above the steady state value.

It is also noted that the steady state error is zero despite using only a PD controller in the system. This is because the open-loop transfer function of the DC-motor system in (3.2) has a system type I, which is responsible for eliminating the steady state error of the system.

After designing a PD controller to stabilize the motor angular position and to achieve specific time response requirements, the question now is whether the designed PD controller will be able to preserve its efficiency if the DC-motor is used to drive the joints of our 2-DOF robotic manipulator. This question can be answered by conducting the following simulation on Dymola:

Two PD-controlled DC-motors as the one shown in figure (3.2) are used to drive the joints of the 2-DOF robotic arm. The motors are connected to the arm joints through gear trains with gear reducers $r_1 = 90$ and $r_2 = 220$ for the shoulder joint and the elbow joint respectively. Figure (3.4) shows a block diagram for the 2-DOF robotic arm driven by the PD-controlled DC-motors. The input voltage signal source connected to both motors is a model developed on Dymola for a pick-and-place function. Figure (3.5) shows the time responses of the angular positions of the two driving motors.

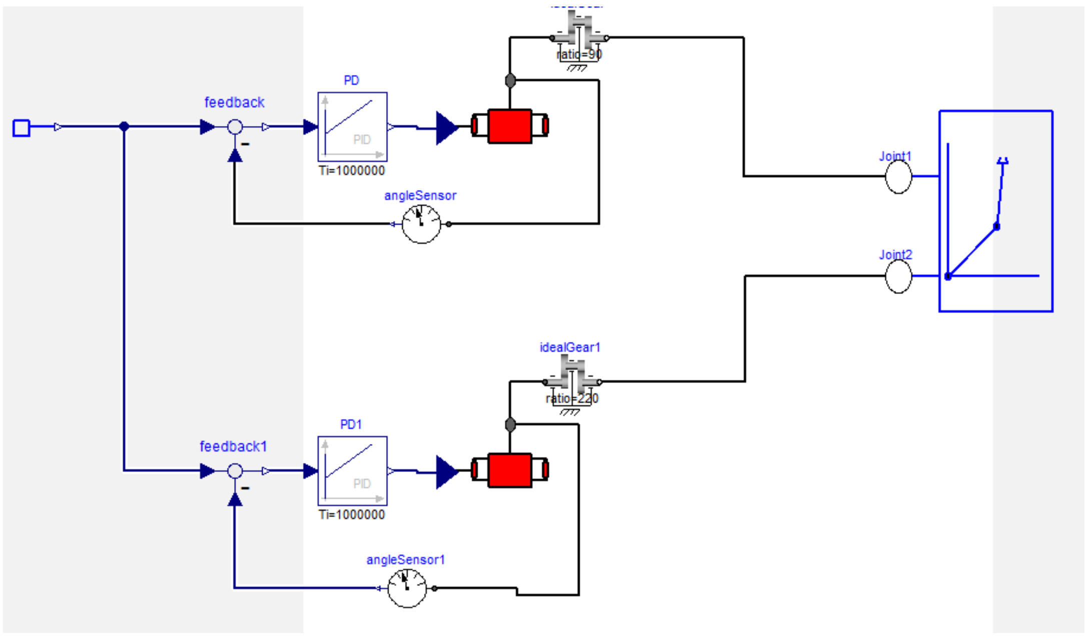


Figure (3.4): Dymola model for the robotic arm driven by PD-controlled DC-motors

From figure (3.5), it is clear that when the PD-controlled DC-motors are connected to the joints of the robotic arm, the designed PD-controller lost its capability of preserving the zero steady state error in the time response of the DC-motor angular position. This can be clearly seen from figure (3.5) where the error in the time response is about $0.5\ rad$ for the shoulder joint motor angular position and about $0.03\ rad$ for the elbow joint motor angular position.

Such a result was actually expected since the PD-controller was designed for the DC-motor without being connected to a load (i.e. the open-loop transfer function of the DC-motor (3.2) was computed based on the assumption that the load torque $\tau_L = 0$). However, in this simulation when the DC-motors are connected to the joints of the robotic arm, the inertias of the driving motors began experiencing some amount of disturbance load torques which are the torques exerted by the dynamics of the 2-DOF robotic arm.

This implies that τ_L now has a value and therefore the designed PD controller is no longer capable of achieving the required time response characteristics. Therefore, in order to recover the efficiency of the designed PD controller, a disturbance torque rejection controller must be developed in order to cancel the effect of the disturbance dynamic torques of the robotic arm joints which will be the topic of the next section.

The difference between the effects of the disturbance joint torques on the angular positions of their driving motors can be clearly observed in figure (3.5) where the disturbance torque of the shoulder joint τ_1 has a greater effect on the angular position of its driving motor θ_{m1} than the effect of the disturbance torque of the elbow joint τ_2 on the motor angular position θ_{m2}.

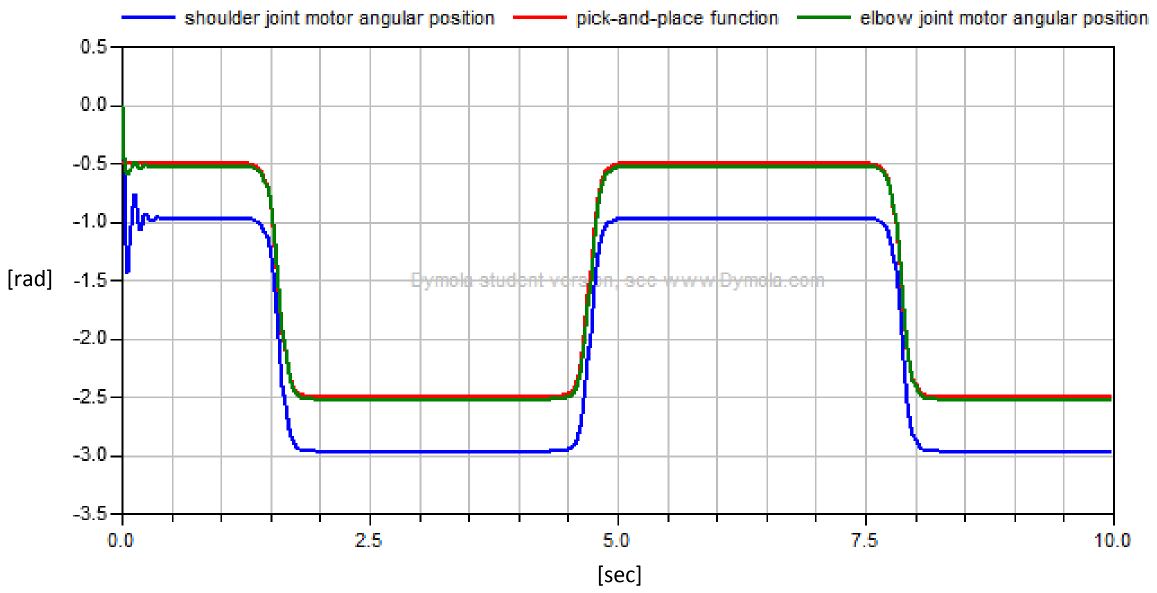


Figure (3.5): Angular positions of the DC-motors with PD-control

In order to see the poor performance of the PD controller in the presence of the disturbance torques of the manipulator joints more clearly, figure (3.6) shows a 3D visualization of the end-effector's trajectory for a period of $10\ secs$. It is shown in figure (3.6) that the manipulator end-effector deviated from the desired trajectory from $t = 0.02\ sec$ to $t = 0.4\ sec$. Such a deviation was caused by the large overshooting of the angular positions of the joint driving motors during that period as shown in figure (3.5).

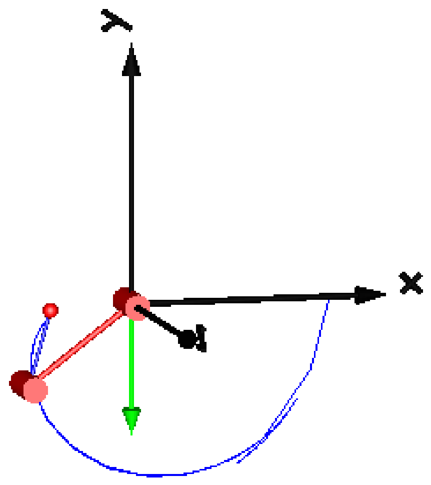


Figure 3.6: 3D visualization of the trajectory of PD-controlled arm

3.2 PD-computed torque control

As shown in section 3.1, using the designed PD controller alone is not effective in achieving the desired trajectory for the 2-DOF robotic manipulator due to the presence of the dynamic disturbance torques of the manipulator joints. In order to improve the performance of the PD controller in the presence of the disturbance torques, a disturbance rejection control strategy must be developed in order to cancel the effect of the disturbance torques of the joints on the angular positions of their driving DC-motors. This section aims to develop a disturbance rejection controller using the well-known computed torque method to be applied on the PD-controlled robotic arm.

The computed torque method was used extensively in the literature as a disturbance torque rejection method to improve the performance of the trajectory tracking control of robotic manipulators [6-8]. This method is based on the idea of computing estimated values for the joint torques using the dynamics equations of the robotic manipulator. The computed torque values can then be added to the actual negative torques of the joints to cancel them out as shall be seen later in this section.

The following discussion shows mathematically the effectiveness of the computed torque method in eliminating the joint disturbance torques for a general robotic manipulator of n DOF:

The mechanical part of the DC-motor model is described as:

$$J_m \frac{d^2\theta_m}{dt^2} + B_m \frac{d\theta_m}{dt} = \tau_m - \frac{\tau_L}{r} - - - - - - - - - (3.3)$$

Where J_m denotes the motor inertia, B_m denotes the motor damping coefficient, τ_m is the motor torque and τ_L denotes the torque generated by the load which in our case describes the disturbance torques due to the dynamics of the driven manipulator joints. These disturbance torques of the joints can be described in matrix form as follows [13]:

$$D(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + g(\theta) = \tau_L - - - - - - (3.4)$$

Where $D(\theta)$ is a symmetric square matrix of dimension $n \times n$ and represents the inertia matrix of the n-DOF robotic manipulator, $C(\theta, \dot{\theta})$ represents the vector of the centrifugal and coriolis forces applied on the manipulator joints, $g(\theta)$ represents the $n \times 1$ vector of gravitational forces applied on the manipulator joints, θ denotes the joint angle, and τ_L represents the torques necessary for driving the joints of the manipulator.

Substituting (3.4) into (3.3) reveals the following equation:

$$J_m \frac{d^2\theta_m}{dt^2} + B_m \frac{d\theta_m}{dt} = \tau_m - \frac{1}{r}[D(\theta)\ddot{\theta} + C(\theta,\dot{\theta})\dot{\theta} + g(\theta)]$$

For simplification purposes, let $C(\theta,\dot{\theta})\dot{\theta} + g(\theta) = h(\theta,\dot{\theta})$, then:

$$J_m \frac{d^2\theta_m}{dt^2} + B_m \frac{d\theta_m}{dt} = \tau_m - \frac{1}{r}[D(\theta)\ddot{\theta} + h(\theta,\dot{\theta})] - - - - - - - - - \quad (3.5)$$

The last term on the right hand side of equation (3.5) represents the disturbance torques that are responsible for the deviation of the PD-controlled motor positions from their desired trajectories as shown in figure (3.5). In order to remove the effect of such disturbance torques of the joints and recover the trajectory tracking efficiency of the PD controller, an additional term must be inserted into equation (3.5) as follows:

$$J_m \frac{d^2\theta_m}{dt^2} + B_m \frac{d\theta_m}{dt} = \tau_m - \frac{1}{r}[D(\theta)\ddot{\theta} + h(\theta,\dot{\theta})] + [\widehat{D}(\theta)\ddot{\theta} + \hat{h}(\theta,\dot{\theta})] - - - - (3.6)$$

Where $\widehat{D}(\theta)$ and $\hat{h}(\theta,\dot{\theta})$ represent the estimated inertia matrix and the estimated centrifugal and coriolis forces, respectively.

In order for the last two terms on the right hand side of equation (3.6) to cancel out, the following equations must be satisfied:

$$\widehat{D}(\theta) = \frac{1}{r}D(\theta) \text{ and } \hat{h}(\theta,\dot{\theta}) = \frac{1}{r}h(\theta,\dot{\theta}) - - - - - - (3.7)$$

The equation (3.7) constitutes the mathematical representation of the computed torque-based disturbance rejection method for an n-DOF robotic manipulator. It is important to note here that the effectiveness of the computed torque method in eliminating the effects of the disturbance torques of the joints depends mainly on the accuracy of computing the actual mass matrix and centrifugal and coriolis forces of the driven manipulator. In the following discussion, the use of equations (3.7) to develop a computed torque-based disturbance rejection control for the 2-DOF robotic arm is investigated:

Expanding the equation (3.4) for the 2-DOF robotic arm reveals the following:

$$\begin{aligned} &d_{11}\ddot{\theta}_1 + d_{12}\ddot{\theta}_2 + h_1(\theta,\dot{\theta}) = \tau_1 \\ &, \\ &d_{21}\ddot{\theta}_1 + d_{22}\ddot{\theta}_2 + h_2(\theta,\dot{\theta}) = \tau_2 \end{aligned} \qquad - - - - - - - - - - - - - - - - \quad (3.8)$$

By equating the equations of (3.8) to the dynamics equations of the robotic arm derived in (2.28) and (2.29), the corresponding elements of the 2×2 symmetric inertia matrix and the corresponding elements of the vector $h(\theta,\dot{\theta})$ of the robotic arm are found to be:

$$d_{11} = \left[m_1\frac{{a_1}^2}{4} + m_2\left(\frac{{a_2}^2}{4} + {a_1}^2 + a_1a_2c_2\right) + \rho\frac{ABC}{6}(A^2 + B^2)\right],$$

$$d_{12} = \left[m_2\left(\frac{{a_2}^2}{4} + \frac{a_1a_2}{2}c_2\right) + \rho\frac{ABC}{12}(A^2 + B^2)\right],$$

$$d_{21} = \left[m_2\left(\frac{{a_2}^2}{4} + \frac{a_1a_2}{2}c_2\right) + \rho\frac{ABC}{12}(A^2 + B^2)\right],$$

$$d_{22} = \left[\frac{{a_2}^2}{4}m_2 + \rho\frac{ABC}{12}(A^2 + B^2)\right],$$

$$h_1 = [-m_2a_1a_2s_2]\dot{\theta}_1\dot{\theta}_2 + \left[-m_2a_1\frac{a_2}{2}s_2\right]{\dot{\theta}_2}^2 + \left[m_1g\frac{a_1}{2}c_1 + m_2g\left(\frac{a_2}{2}c_{12} + a_1c_1\right)\right],$$

$$h_2 = \left[m_2\frac{a_1a_2}{2}s_2\right]{\dot{\theta}_1}^2 + \left[m_2g\frac{a_2}{2}c_{12}\right]$$

Using the equations in (3.8), the estimated inertia matrix $\widehat{D}(\theta)$ and the estimated vector $\hat{h}(\theta,\dot{\theta})$ for the 2-DOF robotic arm are computed using the following relations:

$$\widehat{d_{11}} = \frac{1}{r_1}d_{11}\,,\quad \widehat{d_{12}} = \frac{1}{r_1}d_{12}\,,\quad \widehat{d_{21}} = \frac{1}{r_2}d_{21}\quad,\quad \widehat{d_{22}} = \frac{1}{r_2}d_{22}$$

$$\widehat{h_1} = \frac{1}{r_1}h_1\quad,\quad \widehat{h_2} = \frac{1}{r_2}h_2 \qquad ----------(3.9)$$

The equations (3.9) constitute the computed torque-based disturbance rejection control strategy for the 2-DOF robotic arm. $(1{:}r_1)$ and $(1{:}r_2)$ represent the gear reduction ratios of the gear trains connected to the shoulder joint and the elbow joint, respectively.

In order to test the effectiveness of the computed torque-based disturbance rejection controller to eliminate the effects of the disturbance torques generated by the joint dynamics of the 2-DOF robotic arm, Modelica-based models are developed for both the estimated inertia matrix $\widehat{D}(\theta)$ and the estimated vector of centrifugal and coriolis forces $\hat{h}(\theta,\dot{\theta})$ in order to use them later in the Dymola simulation of the whole system. These models can be found in Appendix C.

Figure (3.7) shows the Dymola model of the arm system with the designed PD-computed torque controller. Figure (3.8) shows the simulation results of the arm control system shown in figure (3.7).

Figure (3.8) shows that with the addition of the designed computed torque disturbance rejection controller, the PD controllers recovered their efficiency of enforcing the joint driving motors to follow their desired pick-and-place trajectory. This shows that the developed computed torque disturbance rejection controller is capable of totally eliminating the effects of the disturbance joint torques on the angular positions of their driving motors.

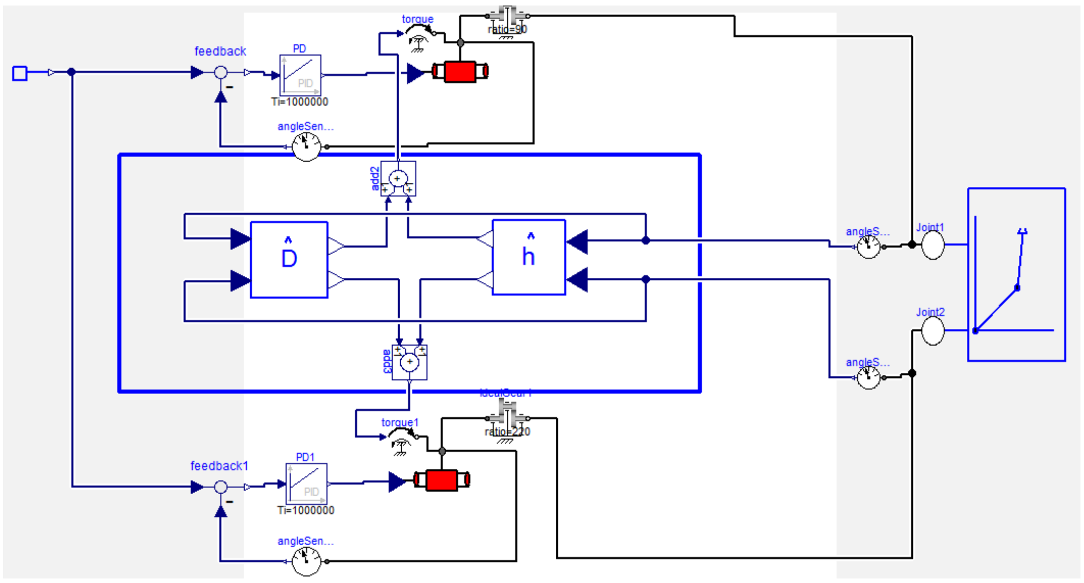


Figure 3.7: Dymola model for the robotic system with computed torque controller

It can be also observed from figure (3.8) that the steady state error in the horizontal areas of the response is equal to zero. However, in the vertical areas of the response, there appears a steady state error of about $0.05\ rad$ between the desired trajectory and the actual angular positions of the motors.

Figure (3.9) shows a 3D visualization of the effect of adding the designed computed torque disturbance rejection controller on improving the trajectory followed by the manipulator end-effector.

Comparing the trajectory followed by the end-effector in figure (3.9) with that shown in figure (3.6), it can be clearly observed that with the addition of the designed computed torque disturbance rejection controller, the large deviation of the end-effector from the desired trajectory was effectively handled as shown in figure (3.9).

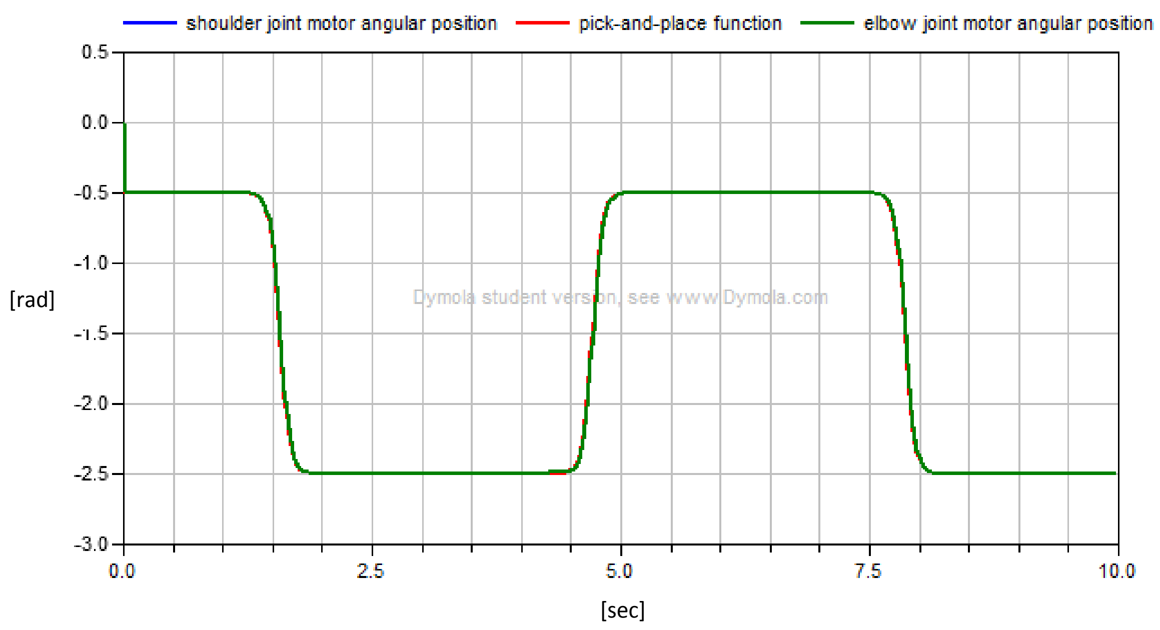


Figure 3.8: Angular positions of motors with computed torque controller

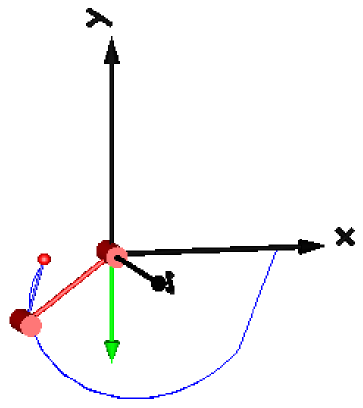


Figure 3.9: 3D visualization of the arm trajectory with computed torque controller

This is supported by figure (3.8) which shows no overshooting for the actual angular position of either driving motor.

The non-zero error shown in the vertical areas of the response in figure (3.8) is caused by the time varying nature of the applied pick-and-place function which cannot be handled by the linear PD controller. Therefore, the feed-forward control technique explained in [2] is used to eliminate this error. The feed-forward error elimination technique depends on adding a feed-forward path in the closed loop control system of each motor such that the transfer function of the feed-forward path is the reciprocal of the open-loop transfer function of the DC-motor system. The following mathematical investigation proves that this feed-forward control technique can totally eliminate the error shown in figure (3.8):

Figure (3.10) shows a symbolic block diagram representing the PD-controlled DC motor system where $C(s)$ is the controller transfer function, $G(s)$ is the open-loop system transfer function, and $F(s)$ is the feed-forward transfer function to be added to the system:

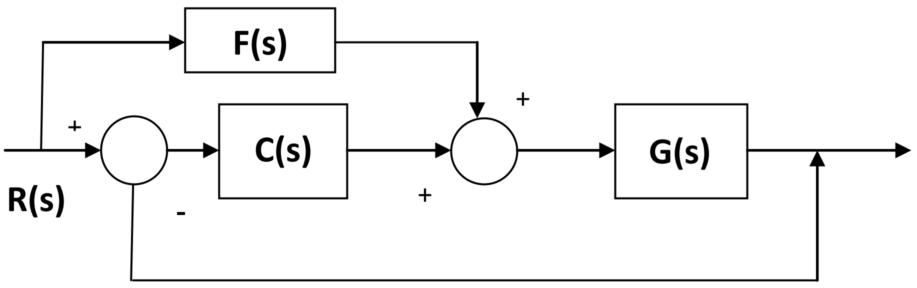


Figure 3.10: Block diagram of control system with feed-forward path

From figure (3.10), it can be shown that

$$E(s) = R(s) - \left[G(s)\left(F(s)R(s) + C(s)E(s)\right)\right]$$

Therefore,

$$E(s)[1 + G(s)C(s)] = R(s)[1 - G(s)F(s)]$$

Which reveals that,

$$E(s) = \frac{R(s)[1-G(s)F(s)]}{[1+G(s)C(s)]} - - - - - - - - (3.10)$$

Equation (3.10) shows that in order for the error $E(s)$ to be equal to zero, the feed-forward transfer function $F(s)$ must be equal to the reciprocal of the open-loop system transfer function of the DC-motor.

Figure (3.11) shows a model developed on Dymola for the feed-forward controller (FFC). Figure (3.12) shows the inclusion of the feed-forward controller (FFC) to each of the PD-controlled driving motors of the arm. Figure (3.13) shows the simulation results of the control system shown in figure (3.12).

It is clear from figure (3.13) that the error due to the time varying nature of the desired pick-and-place trajectory is totally eliminated by the designed feed-forward controller.

As mentioned earlier, the efficiency of the computed torque disturbance rejection control strategy is determined by the accuracy of the torques computed using the relations in (3.9). This accuracy depends mainly on our knowledge of the accurate parameters of the joint driving motors and the controlled manipulator itself.

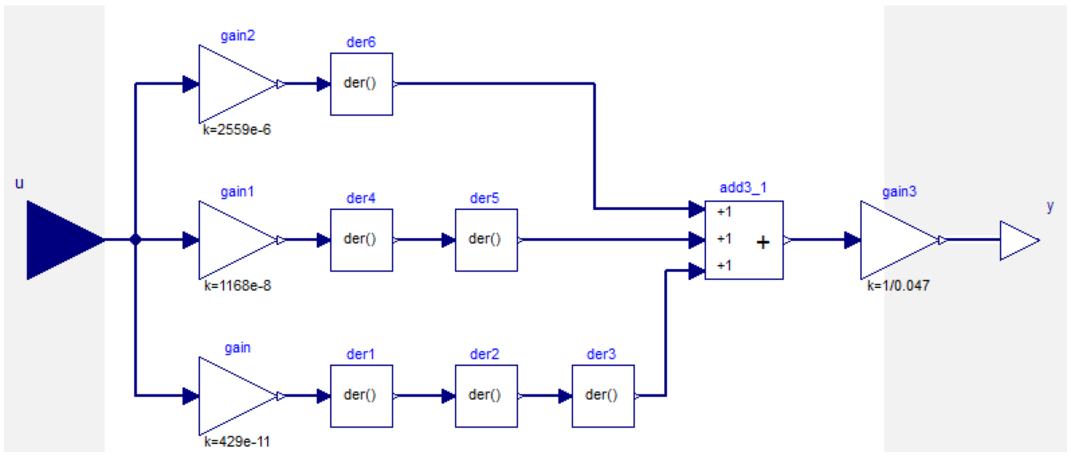


Figure 3.11: Dymola model of the feed-forward controller (FFC)

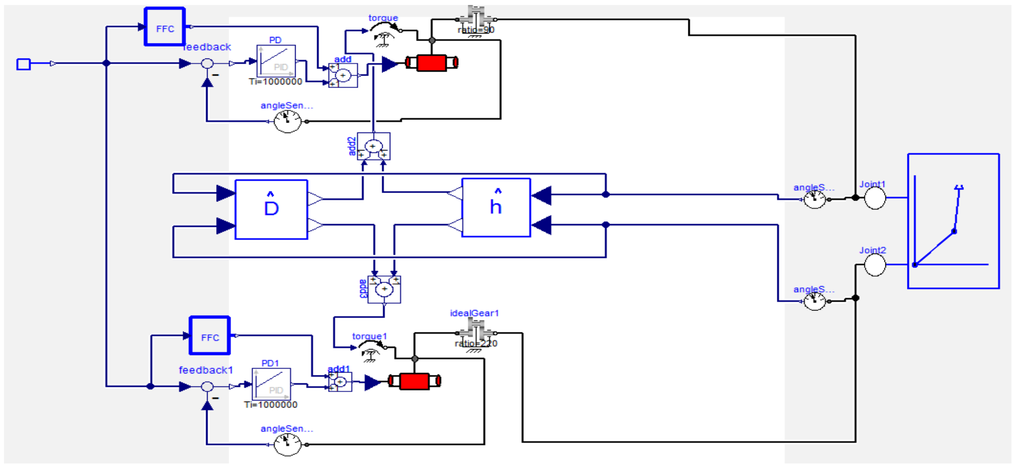


Figure 3.12: Dymola model for the robotic system with computed torque disturbance rejection and feed-forward controller (FFC)

Therefore, the computed torque disturbance rejection controller will fail to achieve its goal of accurately compensating for the joint disturbance torques in either one of the following two conditions:

1- If any of the known parameters of the joint driving DC motors or the controlled robotic arm is not accurate which is the case of a structured uncertainty [7], [8].

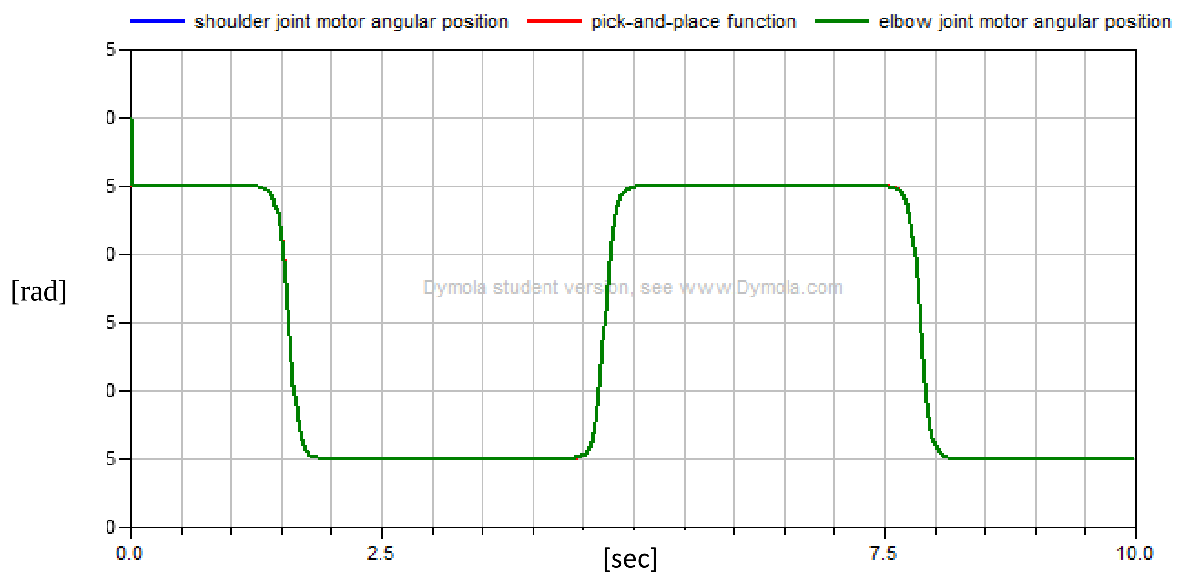


Figure 3.13: Angular positions of the motors with computed torque and feed-forward control (FFC)

2- If the model developed for the robotic arm in (2.28) and (2.29) is not correct in the sense that other un-modeled dynamics of the manipulator exist and were not considered in the development of the manipulator model. This case is known as an unstructured uncertainty. Examples of the un-modeled dynamics that might exist include the coulumb and viscous friction associated with the arm joints, a sudden change in the mass of the payload attached to the end-effector during the online operation of the arm [7,8].

The following simulations aim to test the efficiency of the designed computed torque disturbance rejection controller in the cases of having structured and unstructured uncertainties in the system model:

Simulation in the case of a structured uncertainty:

Table (3.1) lists the accurate and inaccurate parameters of both the joint driving DC-motors and the 2-DOF robotic arm.

Figure (3.14) shows the simulation results of the robotic arm control system shown in figure (3.12) with the parameters of the joint driving motors and the robotic arm changed to their corresponding accurate values listed in table (3.1).

Table 3.1: Accurate and inaccurate parameter values for the driving motors and the robotic arm

DC-motor parameters		Robotic arm parameters	
Inaccurate	**Accurate**	**Inaccurate**	**Accurate**
$R = 3.5\Omega$	$R = 4\Omega$	$a_1 = 0.25m$	$a_1 = 0.35m$
$L = 1.3mH$	$L = 2.4mH$	$a_2 = 0.15m$	$a_2 = 0.30m$
$K_b = K_m = 0.047$	$K_b = K_m = 0.053$	$m_1 = 1.95Kg$	$m_1 = 2.3Kg$
$J_m = 3.3 \times 10^{-6}$	$J_m = 4.2 \times 10^{-6}$	$m_2 = 0.93Kg$	$m_2 = 1.2Kg$
$B_m = 0.0001$	$B_m = 0.001$		

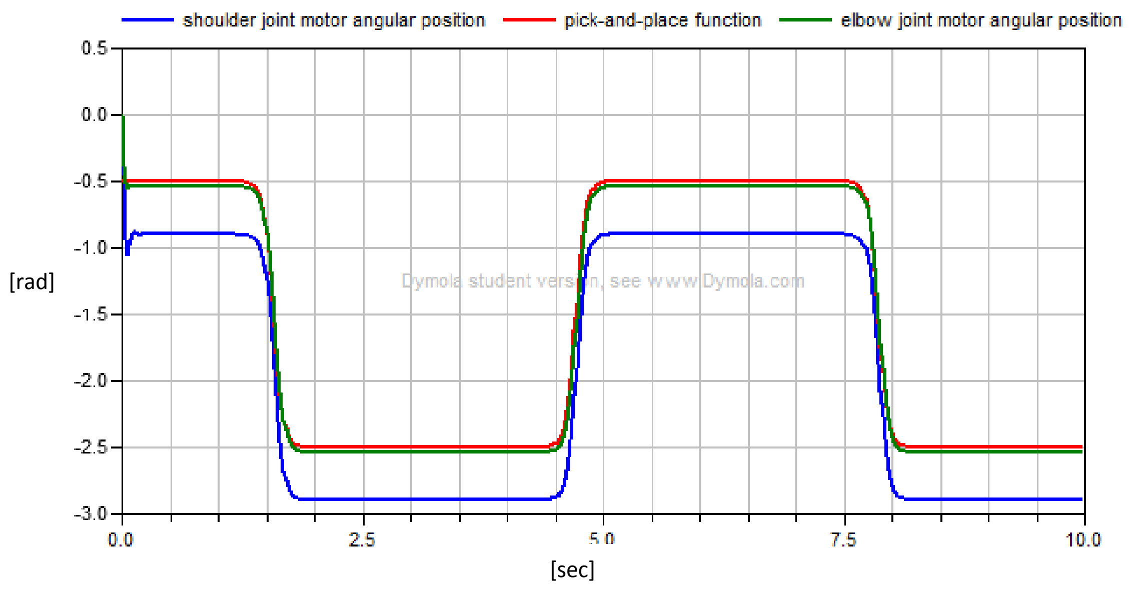


Figure 3.14: Effects of the structured uncertainties on motor angular positions

As shown in figure (3.14), the actual angular positions of both joint driving motors deviated from the desired pick-and-place trajectory despite the presence of the designed disturbance torque rejection and the feed-forward controllers. The error in the horizontal parts of the response for the shoulder joint motor angular position is about 0.5 rad and for the elbow joint motor angular position is about 0.04 rad. This error is due to the fact that the actual disturbance torques of the joints are greater than the estimated torques computed by the designed disturbance rejection controller. The reason for this difference is because the robotic arm parameters used for computing the estimated inertia matrix and the estimated vector of coriolis and centrifugal forces are different from the accurate parameters of the arm.

Figure (3.14) also shows an error in the vertical parts of the response of about 0.5 rad for the shoulder joint motor angular position and about 0.12 rad for the elbow joint motor angular position. This error is partly due to the inaccurate compensation for the actual disturbance torques of the joints and partly due to the inaccurate motor model used in the design of the feed-forward controller.

The effects of the structured uncertainties on the trajectory followed by the manipulator end-effector are shown by the 3D visualization shown in figure (3.15).

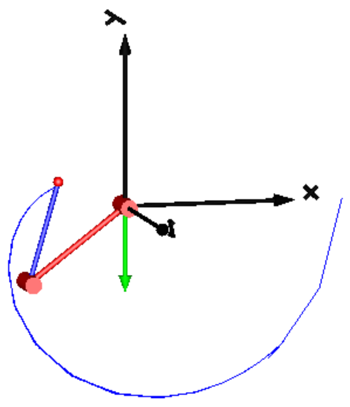


Figure (3.15): Trajectory of the arm with structured uncertainties

Figure (3.15) shows that in the presence of the structured uncertainties, the end-effector slightly deviated from its steady state trajectory during the period from $t = 0.04\, sec$ to $t = 0.14\, sec$. Such a slight deviation is caused by the small overshooting of the shoulder joint driving motor position during that period as shown in figure (3.14).

Simulation in the case of an unstructured uncertainty:

In this simulation, the joints of the robotic arm are assumed to experience disturbance torques at the time instant $t = 5\ sec$. The disturbance torques applied to the arm joints are assumed to be constants with the values $\tau_{d1} = 2.5\ N.m$ and $\tau_{d2} = 2\ N.m.$ These disturbance torques simulate the cases when the robotic arm are subject to external forces that might be applied on the arm links during its operation or when the robotic arm is required to pick and place a payload of a certain mass value.

In such cases, a disturbance torque is added to the dynamic torque of each joint of the robotic arm. Figure (3.16) shows the effects imposed by the externally applied disturbance torques on the angular positions of the joint driving motors at the time instant of $t = 5\ sec$.

As shown in figure (3.16), the designed computed torque disturbance rejection and the feed-forward controllers are incapable of compensating for the suddenly applied external torques at the time instant $t = 5\ sec$.

This can be clearly seen from the errors in the horizontal and vertical parts of the response between the desired pick-and-place trajectory and the actual positions of the joint driving motors that began to appear at $t = 5\ sec$.

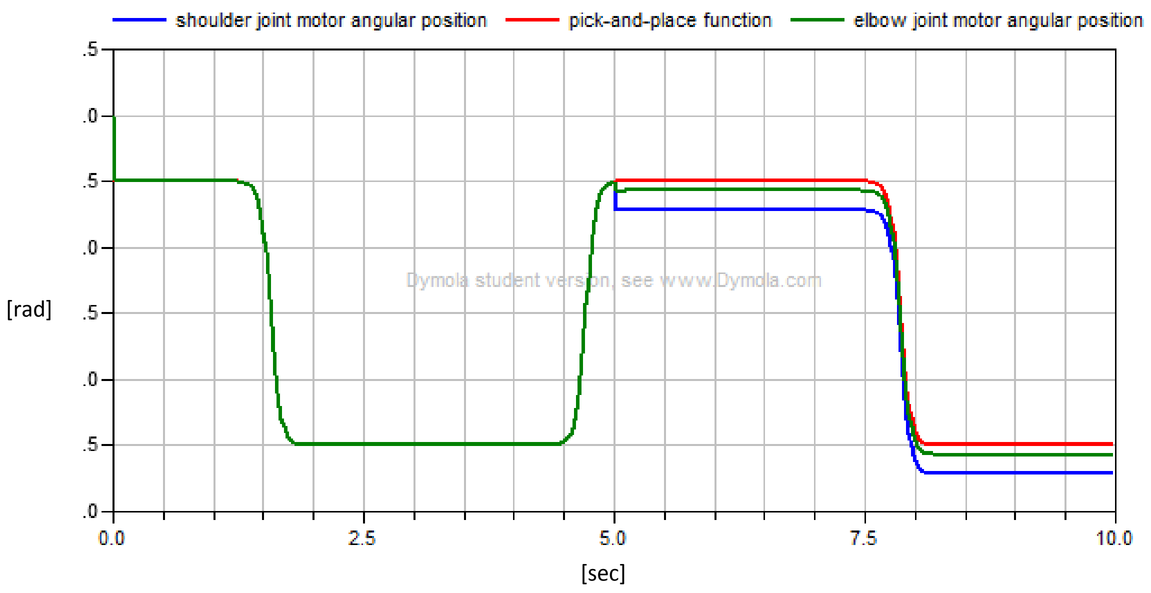


Figure 3.16: Effects of the unstructured uncertainty on motor angular positions

Figure (3.17) shows a 3D visualization of the effects of the unstructured uncertainties on the trajectory followed by the manipulator end-effector.

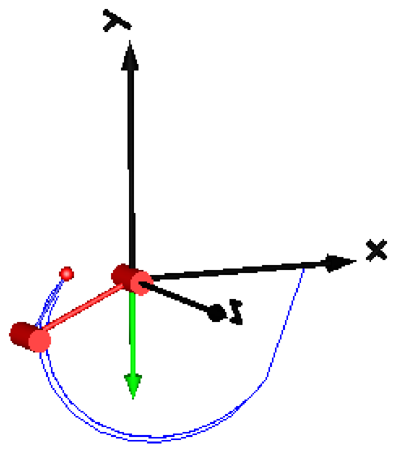


Figure 3.17: Trajectory of the arm with unstructured uncertainties

As shown in figure (3.17), the end-effector started to follow a completely different trajectory from the desired one at the time instant when the external disturbance torques were applied on the manipulator links.

From the simulations conducted above, it can be concluded that the computed torque disturbance rejection method is effective only in the case of having accurate models for both the controlled manipulator and its joints' driving motors which is not necessarily possible in practice. This is due to the presence of both structured and unstructured uncertainties that might suddenly be generated into the manipulator system during its online operation in which case both the computed torque disturbance rejection and the feed-forward controllers would fail to compensate for such uncertainties.

In the next chapter, the problem of compensating for the structured and unstructured uncertainties is handled by using an adaptive control method which is developed based on the learning capabilities of artificial neural networks.

CHAPTER 4

Artificial Neural Network (ANN) Control

Artificial Neural Networks (ANNs) are considered one of the key intelligent tools used in modern research to solve complex problems encountered in a wide variety of engineering applications. They are playing key roles in fields like pattern classification and recognition [19], optimization problem solving [20], prediction and forecasting [21], as well plant identification and control [22]. The development of Artificial Neural Networks (ANNs) was motivated by the scientists conventional understanding of the operation of biological neurons of the human brain [22]. This understanding led to the development of a simple mathematical model called a *perceptron* that emulates the functional behavior of the biological neural network. In the beginning of this chapter, a brief description of the structure and the operational behavior of the biological neuron is given. Then, the mathematical model of an artificial neuron is introduced and its analogy with the biological neuron is explained.

4.1 Biological neuron

A neuron is one of the brain cells that are responsible for the processing and transfer of information represented by electrical impulses. Figure (4.1) shows a graphical sketch that provides a typical description of the biological neuron structure. As shown in figure (4.1), the neuron consists of a number of main components each of which has a critical function. These components include the cell body which contains the nucleus that is responsible for producing the chemical material needed for the neuron; the dendrites are responsible for receiving electrical impulses from other neurons. After the impulses are received and processed, they get transferred to other neurons through the axon which is ended by a number of strands and substrands through which the neuron is connected to other neurons. At the end of these strands, there are synapses whose effectiveness determines the ability of the signal receiving neuron to generate electrical impulses. The effectiveness of a synapse can be enhanced by its previous behavior with the informational signals passing through it to other neurons which implies that a synapse has got *a memory* that learns from its previous activities and is thought to be responsible for the human memory [22].

The cerebral cortex of the human brain contains a vast number of about 10^{11} neurons each of which is connected to $10^3 - 10^4$ other neurons [22]. The message is transferred through the biological neurons superimposed on a train of pulses whose frequency ranges from a few to hundred hertz which is much slower than the high signal transmission frequency in modern digital electronic circuits. However, the parallel distribution of the signals through the human neurons makes them capable of carrying out perceptual tasks such as face recognition much faster than the serially operated electronic circuits. Therefore, the strength of the human neurons lies in their parallel computing and signal distribution capabilities as well as their self-memorizing and learning characteristics.

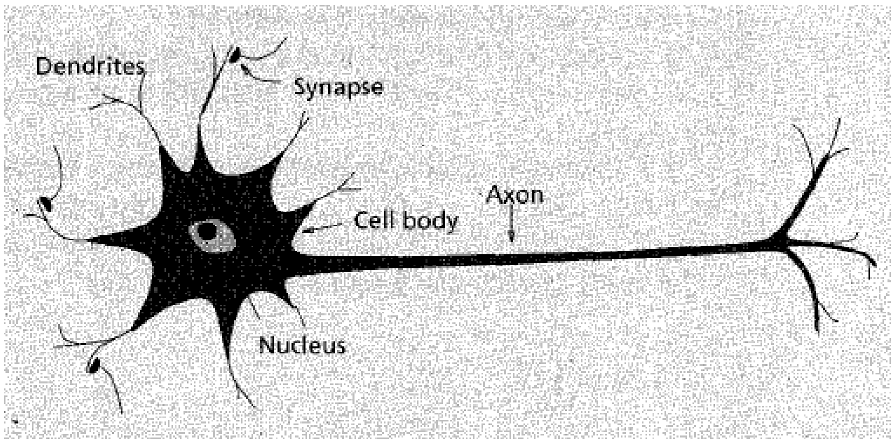


Figure 4.1: A sketch of a biological neuron

4.2 Neuron mathematical model

The understanding of the structure and operational behavior of the biological neuron described in the previous section led to the development of a simple mathematical model for an artificial neuron that emulates the working strategy of the biological neuron. Due to the key perceptual characteristic of the biological neural networks, a layer of artificial neurons is called a perceptron or a *connectionist* [3,22]. Figure (4.2) shows a general graphical notation that represents the mathematical model of a neuron receiving a number R of inputs and producing one output.

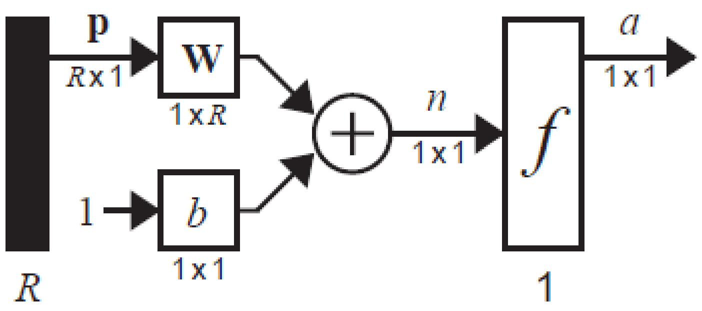


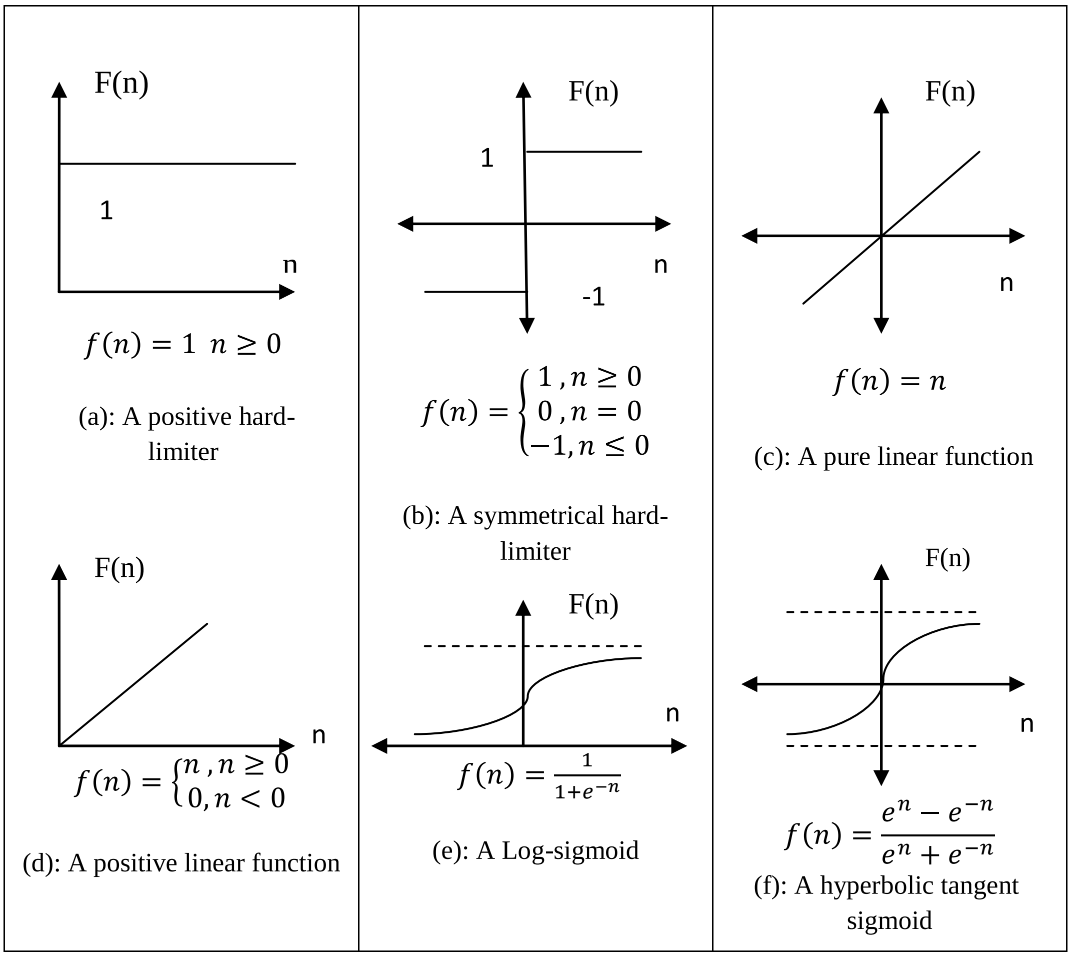
Figure 4.2: A graphical notation of an R-input neuron

As shown in figure (4.2), the mathematical model of an artificial neuron is composed of an input vector R containing the input signals received by the neuron from either an external source or other neurons. The input vector is the analogy of the dendrites of the biological neuron; a weight vector W which is responsible for connecting each input signal with the neuron. The weight vector represents the mathematical model of the synapses of the biological neuron; a bias b is an optional component of the neuron's model and is usually used to change the location of the boundary decision line to improve the classification of input patterns [3]; the net input n of a neuron defines the sum of the weighted inputs and the bias associated with that neuron. This net input is the critical quantity that determines, through an activation function f, the nature of the neuron's output.

From figure (4.2), the mathematical relationship between the neuron's output a and the input vector p is defined as:

$$a = f(n) = f(Wp + b) ------(4.1)$$

There are different types of linear and non-linear activation functions that can be used in the mathematical model of a neuron depending on the nature of the problem that the neuron is required to solve. Some of the commonly-used activation functions include *a positive hard-limiter*, *a symmetrical hard-limiter*, *a pure linear function*, *a positive linear function*, *a log-sigmoid*, and *a hyperbolic tangent sigmoid*. The graphs of such types of activation functions and their mathematical relationships are shown in figure (4.3).



$f(n) = 1 \;\; n \geq 0$

(a): A positive hard-limiter

$$f(n) = \begin{cases} 1\,, n \geq 0 \\ 0\,, n = 0 \\ -1, n \leq 0 \end{cases}$$

(b): A symmetrical hard-limiter

$f(n) = n$

(c): A pure linear function

$$f(n) = \begin{cases} n\,, n \geq 0 \\ 0, n < 0 \end{cases}$$

(d): A positive linear function

$$f(n) = \frac{1}{1+e^{-n}}$$

(e): A Log-sigmoid

$$f(n) = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$$

(f): A hyperbolic tangent sigmoid

Figure (4.3): Different types of neuronal activation function

As shown in figure (4.3), the hard-limiting and linear activation functions are linear and therefore can be used to solve linear problems such as the classification of linearly separated patterns [3]. Problems like complex nonlinear function approximation, nonlinear system identification, and process control require the use of nonlinear activation functions such as the log-sigmoid functions shown in figures (4.3-e) and (4.3-f).

As mentioned earlier, a number of neurons can be placed in different configurations (layers) to form a neural network (perceptron). A perceptron network may consist of a single layer or multiple layers of neurons. A single layer perceptron consists of a number of neurons placed in a parallel structure. Figure (4.4-a) shows a single layer perceptron consisting of a number of S neurons each of which has its own activation function which might be equal or different from those of other neurons, and its own weight vector that connects it to the input vector.

It is clear from figure (4.4-a) how complex the connections of the inputs with the neurons of a single layer perceptron are. Since there is no limitation for the number of inputs applied to a single layer perceptron or the number of neurons that may be placed in one layer, a more simplified representative notation of a single layer perceptron is used as shown in figure (4.4-b).

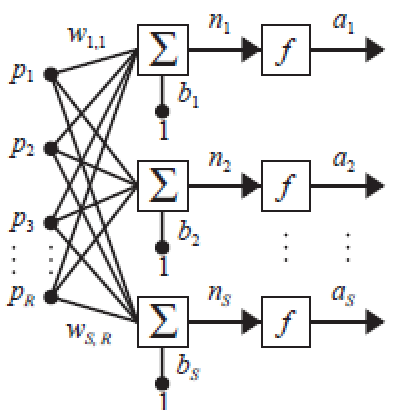


Figure (4.4-a): A single layer perceptron of S neurons

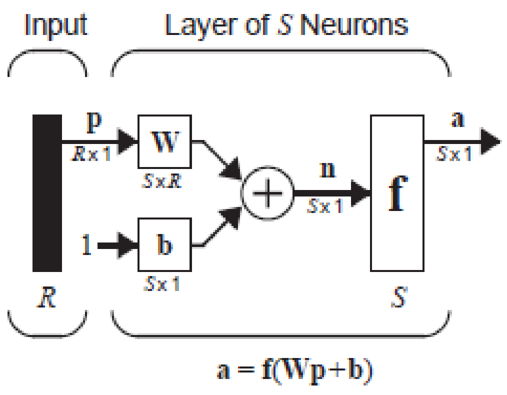


Figure (4.4-b): An Abbreviated notation of a single layer perceptron of S neurons

It is shown in figure (4.4-b) that the weights of the S neurons of the single layer perceptron is represented by a single weight matrix $\boldsymbol{W}$ with the dimensions $S \times R$. This indicates that each row of the weight matrix $\mathbf{W}$ is associated with a single neuron of the layer. Likewise, the biases associated with the S neurons of the layer are gathered to form a single column vector denoted by **b**.

Since each neuron in the layer has its own activation function, the abbreviated notation in figure (4.4-b) represents the neurons activation functions by a single matrix **f** with the dimensions$S \times S$ and the following form:

$$\mathbf{f} = \begin{bmatrix} f(n_1) & 0 & \dots & 0 \\ 0 & f(n_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & f(n_s) \end{bmatrix}$$

The mathematical relationship between the output vector of a single layer perceptron **a** and the input vector **p** is written in a matrix form as:

$$\mathbf{a} = \mathbf{f}(\mathbf{Wp} + \mathbf{b}) - - - - - - - - -(4.2)$$

A multiple-layer perceptron consists of a number of single-layer perceptrons connected to each other in a series structure such that the outputs of each layer constitute the inputs applied to the next layer. Figure (4.5) shows a multiple-layer perceptron consisting of three single layer perceptrons.

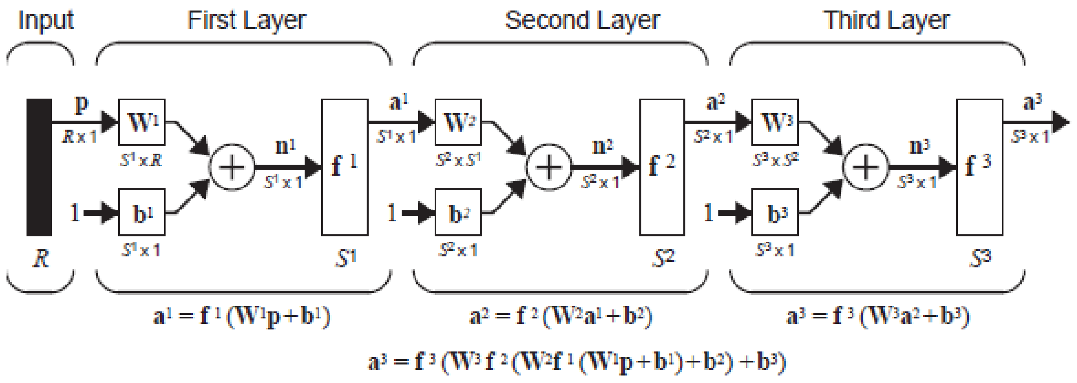


Figure 4.5: A multiple layer perceptron of three single layer perceptrons

A multiple layer perceptron may contain a large number of single layer perceptrons. Therefore, in order to simplify the description of the mathematical relationship between the output and the input vectors of a certain layer, a superscript using the order number of the layer in the network is used as an indication of that layer. This can be clearly seen from figure (4.5) in which every symbol is attached with a superscript number to denote the layer referred to by that symbol. As an example, the weight matrix of the first layer is denoted by $\mathbf{W^1}$; the output vector of the second layer is denoted by $\mathbf{a^2}$**,** and so on. Following this superscript notation, the mathematical description of the whole three-layer perceptron shown in figure (4.5) can be simply written in a single line as:

$$\mathbf{a^3} = \mathbf{f^3}(\mathbf{W^3 f^2}(\mathbf{W^2 f^1}(\mathbf{W^1 p} + \mathbf{b^1}) + \mathbf{b^2}) + \mathbf{b^3}) - - - - - - - - (4.3)$$

The last layer of a multiple layer perceptron is given the name of the output layer and all the other layers are called hidden layers. For the perceptron shown in figure (4.5), the third layer is called the output layer and the two others are hidden layers.

4.3 Neural network learning

As mentioned earlier, the key strength of the biological neural network lies in its capability of learning from the previous activities of its synapses. This learning process helps to enhance the effectiveness of the neural synapses to generate better response signals.

Likewise, in order for an artificial neural network (perceptron) to produce a desired output response to a certain applied input signal, it must undergo a learning process through which a teaching signal such as an error function determines whether the network output reaches an acceptable level or requires further optimization. If the teaching signal decides to further optimize the output value of the neural network (ANN), the weight matrices and bias vectors of all the layers of the perceptron must be adjusted through a set of updating rules determined by a learning algorithm.

There are various types of learning algorithms used in the literature for the learning process of (ANNs) depending on the type of the neural network as well as the nature of the problem to be solved. For example, the perceptron rule was used for pattern classification problems as in [3], the Hebbian rule in [3] was used to teach an Adaptive Linear (ADALINE) neural network to recognize decimal number patterns.

The learning algorithms usually used for teaching multi-layer perceptrons are called backpropagation algorithms due to the fact that the derivatives used in the updating rules of these algorithms are propagated from the last layer of the perceptron back to the first layer [3]. Some of the well-known backpropagation algorithms include the steepest descent algorithm (SDA), Newton, and Levenberg-Marquardt (LM) algorithm [3,22]. Despite the faster convergence of the Newton and LM algorithm than the SDA algorithm, the later one has been mostly used in the literature for the learning process of multiple-layer perceptrons used in robotic manipulator trajectory tracking control applications. This is due to its simple and easily-programmable rules used for updating the network weights and biases.

The SDA algorithm was used frequently for the training of the feed-forward multiple layer perceptrons in order to enable them to produce an output that continuously converges to the desired value. Therefore, it can be considered as an optimization method whose performance index is the output error of the neural network.

Since the SDA algorithm is used to minimize a function of the output error of a multiple-layer perceptron, it is best to formulate a mathematical description of such a performance index to be minimized.

Consider the feed-forward three-layer perceptron shown in figure (4.5). The performance index function to be minimized by the SDA is described as follows:

$$\mathrm{E}(k) = \mathbf{e}^{\mathbf{T}}(k)\mathbf{e}(k) ---------(4.4)$$

Where $\mathrm{E}(k)$ is the value of the performance index at the k^{th} iteration, and $\mathbf{e}(k)$ denotes the output error vector of the perceptron at the k^{th} iteration and is given by:

$$\mathbf{e}(k) = \mathbf{a}^{3}(k) - \mathbf{a}_{d}(k) -------- (4.5)$$

Where $\mathbf{a}^{3}(k)$ is the actual output vector of the perceptron at the k^{th} iteration and $\mathbf{a}_{d}(k)$ is the desired output vector of the perceptron at the k^{th} iteration.

In order to use the SDA algorithm, a set of proper responses of the perceptron should be defined for a specified set of input values. These input-output pairs are used as the training data of the neural network. After defining the training data set, it is now suitable to describe the steps of the SDA which are shown in the flow chart in figure (4.6).

For each iteration of the algorithm, a new input vector is applied and the response of the perceptron is evaluated. The value of the performance index defined in (4.4) is then checked if it satisfies a certain error value. If it does, the algorithm is terminated and the perceptron is said to have learned the set of data used in the training process. If the performance index value does not satisfy the desired criterion, the weights and biases of the perceptron are modified according to the following update rules [3,22]:

$$w_{ij}^{m}(k+1) = w_{ij}^{m}(k) - \alpha \frac{\partial E}{\partial w_{ij}^{m}} -------- (4.6)$$

$$b_{i}^{m}(k+1) = b_{i}^{m}(k) - \alpha \frac{\partial E}{\partial b_{i}^{m}} -------(4.7)$$

Where α is the learning rate which is usually chosen to be a small value and m is the superscript indicating the order number of the layer referred to.

The updating rules in (4.6) and (4.7) are based on the steepest descent optimization method in which the next step in the process of searching for the minimum value of a function is taken in the direction of the negative gradient of the function to be optimized with respect to the search variable [23]. The function to be minimized in this case is the error performance index defined in (4.4) and its search variables are the perceptron weights and biases.

The computation of the partial derivatives involved in (4.6) and (4.7) can be conducted using the chain rule as follows:

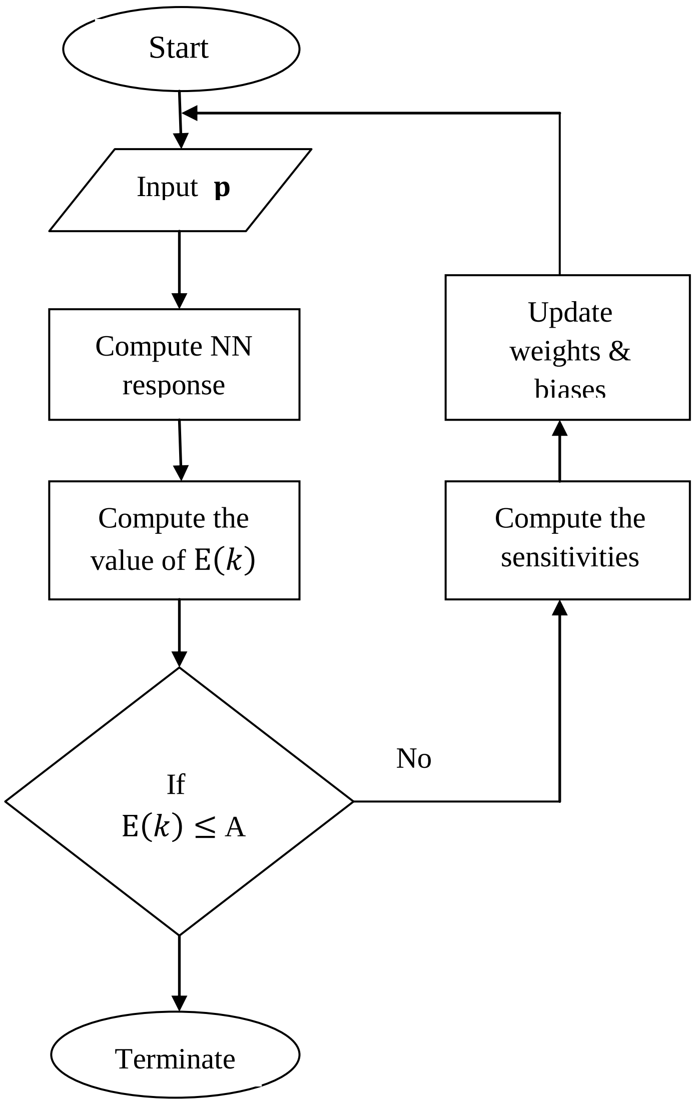


Figure 4.6: Flow chart of the SDA algorithm

$$\frac{\partial E}{\partial w_{ij}^m} = \frac{\partial E}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial w_{ij}^m} - - - - - - - - (4.8)$$

and,

$$\frac{\partial E}{\partial b_i^m} = \frac{\partial E}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial b_i^m} - - - - - - - - (4.9)$$

Where n_i^m denotes the i^{th} element of the net input vector of the m^{th}layer.

From figure (4.5), it can be shown that:

$$n_i^m = \sum_{j=1}^{s^{m-1}} w_{ij}^m \, a_j^{m-1} + b_i^m - - - - - - - - (4.10)$$

The second partial derivative in each of (4.8) and (4.9) can be computed using (4.10) as follows:

$$\frac{\partial n_i^m}{\partial w_{ij}^m} = a_j^{m-1} \quad , \quad \frac{\partial n_i^m}{\partial b_i^m} = 1 - - - - - - -(4.11)$$

Let the first partial derivative involved in (4.8) and (4.9) be defined as the sensitivity of the error performance index function due to changes in the i^{th} element of the net input vector of the m^{th}layer and let it be denoted by S_i^m. By substituting the equations given in (4.8), (4.9), and (4.11) into the equations (4.6) and (4.7), the updating rules of the weights and biases of the multiple-layer perceptron are given in the following forms:

$$w_{ij}^m(k+1) = w_{ij}^m(k) - \alpha S_i^m a_j^{m-1} - - - - - - - - (4.12)$$

$$b_i^m(k+1) = b_i^m(k) - \alpha S_i^m - - - - - - -(4.13)$$

The above updating rules can be reformulated in matrix form as follows:

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{S}^{\mathrm{m}}(\mathbf{a}^{m-1})^{\mathbf{T}} - - - - - - - - (4.14)$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{S}^{\mathrm{m}} - - - - - - - - (4.15)$$

The sensitivity vectors associated with the layers of the perceptron $\mathbf{S}^{\mathrm{m}}$ are computed using the following equations:

$$\mathbf{S}^{\mathrm{M}} = -2\dot{\mathbf{F}}^{\mathrm{M}}(n^{\mathrm{M}})\, \mathbf{e}(k) - - - - - (4.16)$$

Where (M) is the order number referring to the output layer of the perceptron, and $\dot{\mathbf{F}}^{\mathrm{M}}(n^{\mathrm{M}})$ is the derivative of the matrix of the activation functions used in the output layer with respect to their corresponding net inputs.

$$\mathbf{S}^m = \dot{\mathbf{F}}^m(n^m)(\mathbf{W}^{m+1})^{\mathbf{T}}\mathbf{S}^{m+1} - - - - - (4.17)$$

The equation (4.17) is used for computing the sensitivity vector associated with the m^{th} hidden layer. $\dot{\mathbf{F}}^m(n^m)$ is the derivative of the matrix of the activation functions used in the m^{th} hidden layer with respect to their corresponding net inputs.

It can be observed from the equation (4.17) that the sensitivity vector of a certain layer in the perceptron depends on the sensitivity vector of the next layer. This implies that the sensitivity vectors are computed in a backward direction starting from the output layer going back towards the input layer. That is the reason why the SDA algorithm is named as an error backpropagation algorithm.

4.4 Neural network-based control

As mentioned earlier, (ANNs) are one of the modern intelligent tools that have been used for the identification and control of complex nonlinear plant models. One of the major applications in which neural networks are being used is the trajectory tracking control of robotic manipulators.

In general, neural network-based control structures can be classified into model-based and non-model based configurations. A model-based neural control strategy requires the presence of a model of the plant to be controlled. An example of a model-based neural network controller is the one proposed in [8] in which two feed-forward multiple-layer perceptrons were trained offline to learn the highly complicated and nonlinear estimated inertia matrix and the vector of centrifugal, coriolis, and gravitational forces of a 2-DOF robotic arm. The output vector of the linear PD controller was used as the performance index to update the weights and biases of both perceptrons to compensate for any structured or unstructured uncertainties in the estimated model.

In contrast, a non-model based neural control strategy does not require a mathematical model or the knowledge of the parameters of the plant to be controlled. Suel *et. al.* [26] introduced a non-model based neural control strategy called a feedback error learning structure in which a multiple-layer perceptron was used as a feed-forward controller to conduct online learning of the unknown inverse dynamics model of a 2-DOF robotic manipulator. A PD controller was used to stabilize the angular positions of the joint driving motors and its output was taken as the teaching signal of the feed-forward neural network controller. There are various other non-model based neural network-based control configurations that have been used for trajectory tracking control of robotic manipulators such as the Reference Compensation Scheme used by Jung and Hsia *et. al.* [24,25].

In the next subsection, a model-based neural network controller is designed and tested for helping the computed torque disturbance rejection controller to compensate for the structured and unstructured uncertainties.

4.4.1 Model-based neural network control (Online Torque Compensator (OTC))

As mentioned earlier, a model based control strategy depends on the presence of a certain model of the controlled plant. For the 2-DOF robotic arm, the model that will be used is the inverse dynamics model derived in (2.28) and (2.29) which was the basis of designing the computed torque disturbance rejection controller in chapter 3.

The simulation results obtained in chapter 3 proved that the computed torque disturbance rejection controller is not capable of compensating for the structured and unstructured uncertainties. Therefore, an intelligent control technique based on the learning capabilities of neural networks is employed to learn the difference between the actual torques generated by the arm joints and the torques generated by the designed computed torque disturbance rejection controller.

Figure (4.7) shows the inclusion of the designed neural OTC controller into the arm control system. In this figure, it is shown that the designed neural OTC controller receives the actual angular positions of the joints as input signals and generates two corresponding torque signals. The weights and biases of the neural network are updated according to the rules of the SDA algorithm. The performance index function used to guide the updating process of the network weights and biases is the sum of the squares of the outputs generated by the PD and the feed-forward (FFC) controllers of the joint driving motors.

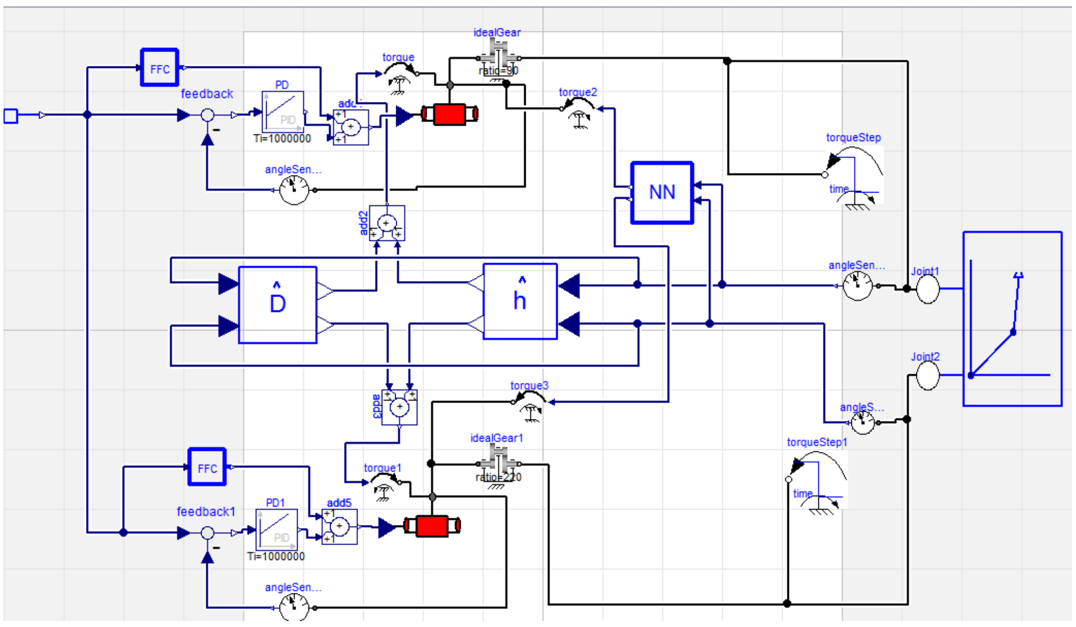


Figure 4.7: Dymola model of the robotic arm with neural network (OTC) controller

The neural OTC controller used in figure (4.7) is a feed-forward multiple-layer perceptron consisting of two layers. The output layer contains two neurons with a pure linear activation function each to produce the two torque difference output signals. The first layer of the perceptron is a hidden layer containing three neurons with a log-sigmoid activation function each. The number of neurons in the hidden layer was selected based on two aspects:

1- The complexity of the problem to be solved by the neural network is not that large since the proposed network is required to learn only the unknown factors of the system which constitute the inaccurate parameters of the joint driving motors and the robotic arm links as well as any un-modeled dynamics that were not considered in the developed model of the manipulator such as the un-modeled torques and forces applied on the joints. The network is not required to learn the whole manipulator dynamics.

2- It has been shown by several experiments in [3] that increasing the number of neurons in the hidden layer of a neural network does increase the complex function approximation capability of the network for the data set used in the training process but affects its generalization capability when other data not included in the training set is introduced to the network.

Therefore, the number of neurons in the hidden layer of the proposed perceptron was chosen to be as small as 3.

Figure (4.8) shows the structure of the designed two-layer perceptron used for learning the uncertainties in the 2-DOF robotic arm control system.

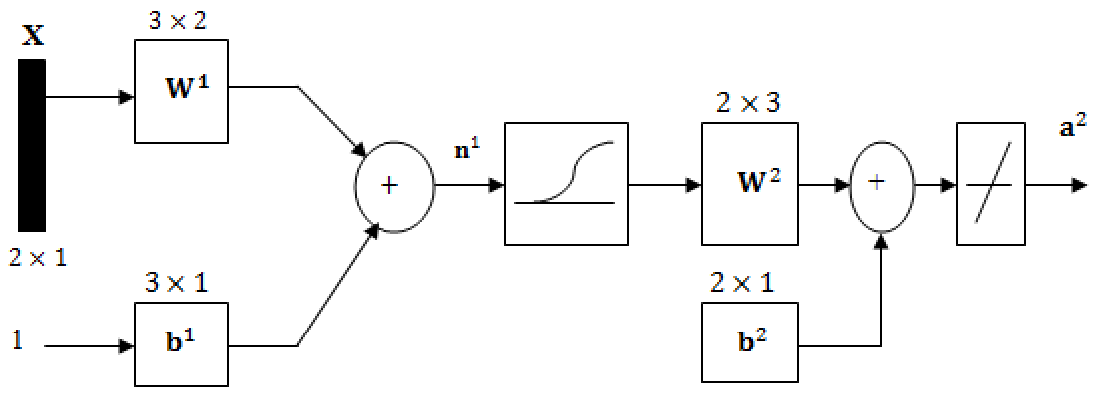


Figure 4.8: Structure of the designed two-layer perceptron neural network

The modelica model developed in Dymola together with a diagram icon for the two-layer perceptron shown in figure (4.8) can be found in Appendix D.

As mentioned earlier, the learning process of the two-layer perceptron controller aims to minimize the error between the actual torques of the arm joints and the torques generated by the computed torque disturbance rejection controller. This means that the main goal of using the neural network controller is to eliminate the total disturbance torques applied on the inertias of the joint driving motors.

In this case, the only controllers affecting the responses of the motors angular positions are the PD and feed-forward (FFC) controllers. This implies that the performance index function to be minimized by the network learning algorithm is the sum of the squares of the outputs generated by the PD and FFC controllers.

The weights and biases of the perceptron are updated using the SDA rules derived in (4.14 - 4.17). A modelica code was written to simulate the application of the SDA algorithm for training the designed two-layer perceptron network. This Modelica code can be found in Appendix D.

In order to test the effectiveness of the designed OTC controller to compensate for both structured and unstructured uncertainties, the following simulation is conducted:

The structured uncertainty is simulated by changing the values of the parameters of the joint driving motors and the robotic arm to their accurate values mentioned in table (3.1). The unstructured uncertainty is simulated by applying a constant disturbance torque of $\tau_d = 2\ N.m$ on both joints of the robotic arm at the time instant $t = 5\ sec$. as shown in figure (4.7).

Figure (4.9) shows the motor angular positions after using the neural network (OTC) controller.

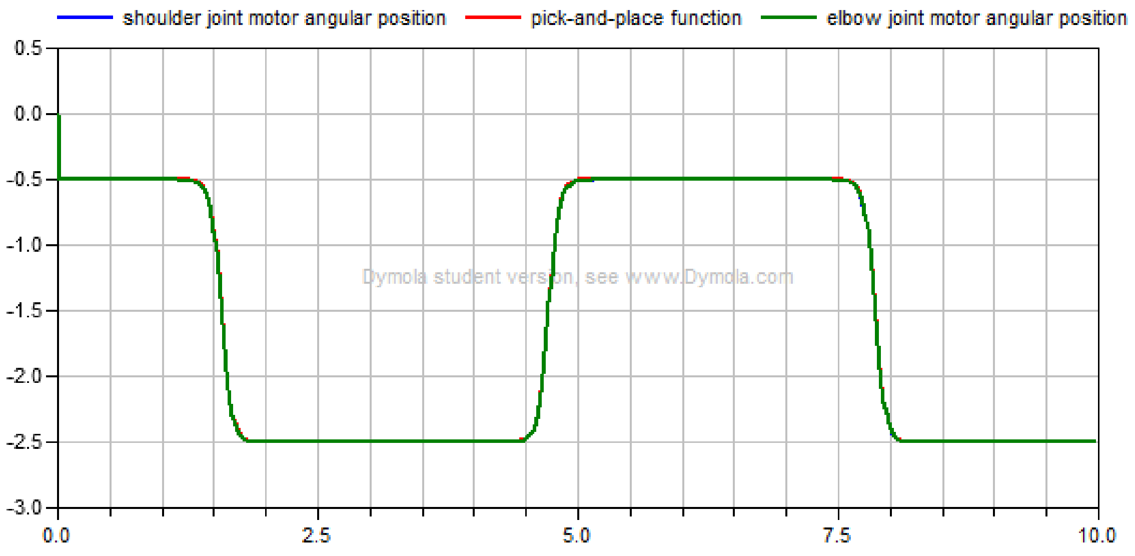


Figure 4.9: Angular positions of the motors with OTC controller

By comparing the simulation results in figures (3.14) and (3.16) with those obtained in figure (4.9), it can be clearly seen that the OTC controller is effectively capable of compensating for both structured and unstructured uncertainties involved in the robotic arm control system.

The large motor position errors appearing in figures (3.14) and (3.16) due to the structured and unstructured uncertainties approximately totally disappeared from figure (4.9). The position errors in the horizontal and vertical parts of the response are both equal to approximately $0.006\ rad$ for both motors.

In order to see the effectiveness of the designed two-layer perceptron in learning the difference between the actual disturbance torques of the arm joints and the torques generated by the computed torque disturbance rejection controller, the total disturbance torque applied on the inertia of each motor is plotted before and after adding the (OTC) controller as shown in figures (4.10) and (4.11) respectively.

Figure (4.10) shows that before the time instant $t = 5\,sec$, the shoulder joint driving motor was experiencing a disturbance torque of about $-0.05\,N.m.$, and the elbow joint driving motor was experiencing a disturbance torque of about $-0.005\,N.m.$ Such disturbance torques were due to the structured uncertainties in the parameter values of the arm.

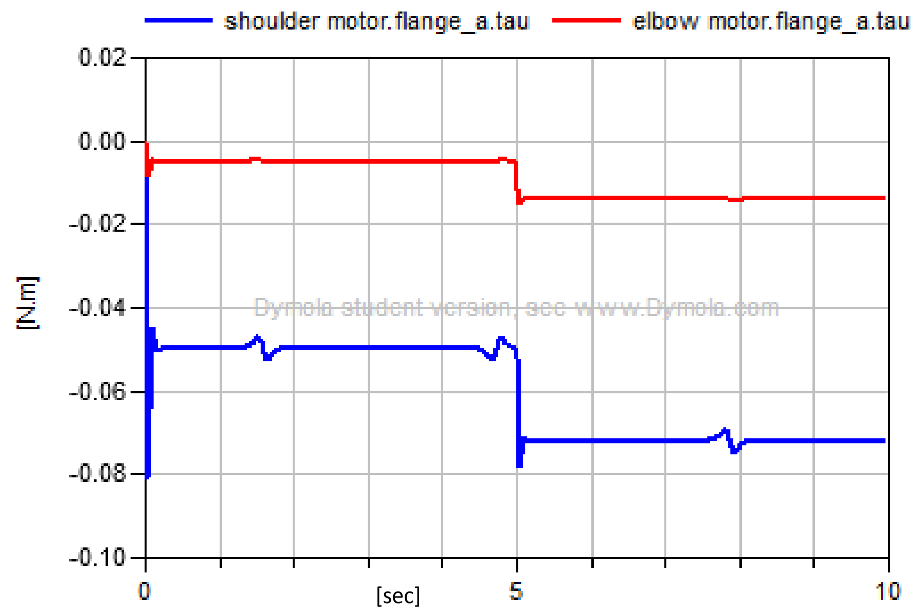


Figure 4.10: Total disturbance torques applied on motors without OTC controller

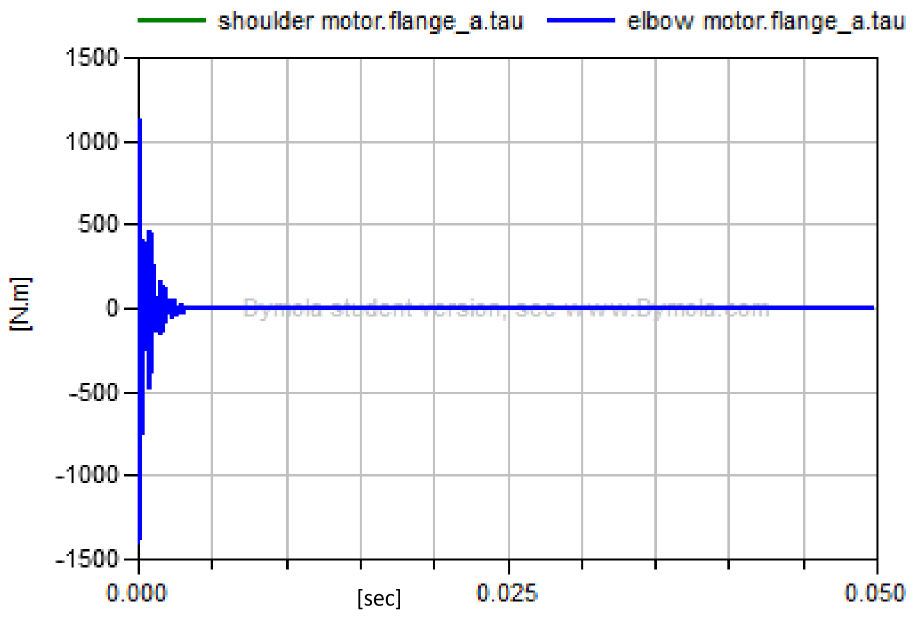


Figure 4.11: Total disturbance torques applied on motors with (OTC) controller

When an external constant disturbance torque is suddenly applied on each motor at the time instant $t = 5\, sec$ which is the case of the unstructured uncertainty, the total disturbance torques applied on the motors increased to become $-0.07\, N.m.$ for the shoulder joint driving motor and $-0.014\, N.m.$ for the elbow joint driving motor.

When the designed OTC controller is added to the system, it successfully learned to generate the disturbance torques and hence eliminated the total disturbance torque applied on each joint driving motor within a period of $9\, mSec$ as shown in figure (4.11).

Figures (4.12) and (4.13) show the sum of the outputs of the linear PD and feed-forward (FFC) controllers for both joints before and after adding the (OTC) controller, respectively.

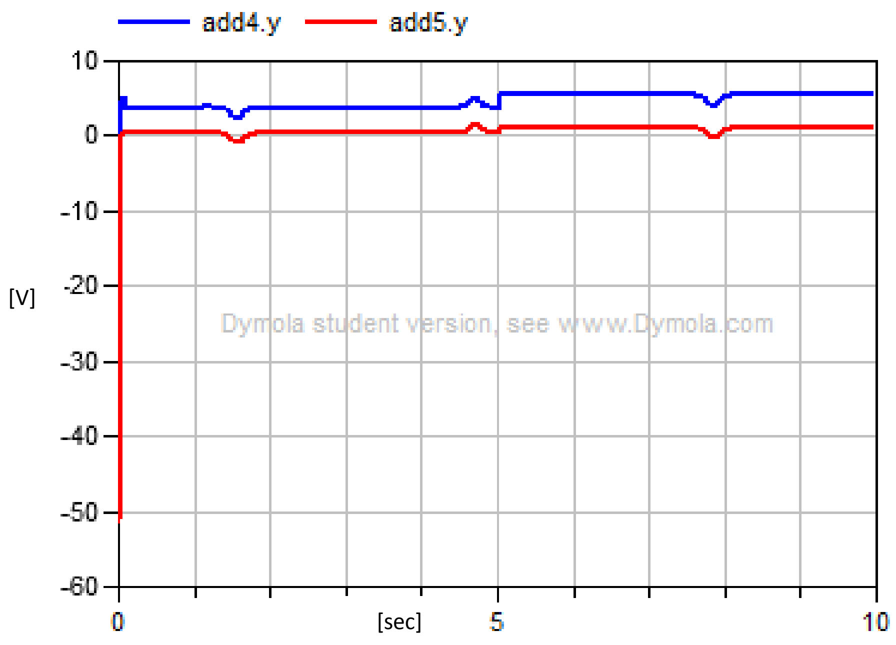


Figure 4.12: Sum of the outputs of PD and FFC controllers without (OTC) controller

Figure (4.12) clearly shows that the linear PD and feed-forward (FFC) controllers failed to enforce the joint driving motors to follow the desired pick-and-place trajectory. This is reflected by their almost continuous signals with the values of about $3.8\, V$ for the shoulder joint driving motor, and about $0.4\, V$ for the elbow joint driving motor before the time instant $t = 5sec$. After $t = 5sec$, the PD and FFC controllers supplied a signal of about $5.4\, V$ to the shoulder joint driving motor and a signal of about $1.06\, V$ to the elbow joint driving motor. Such non-decreasing signals of the linear PD and FFC controllers indicate that the computed torque controller alone is not effective in compensating for the structured and unstructured uncertainties.

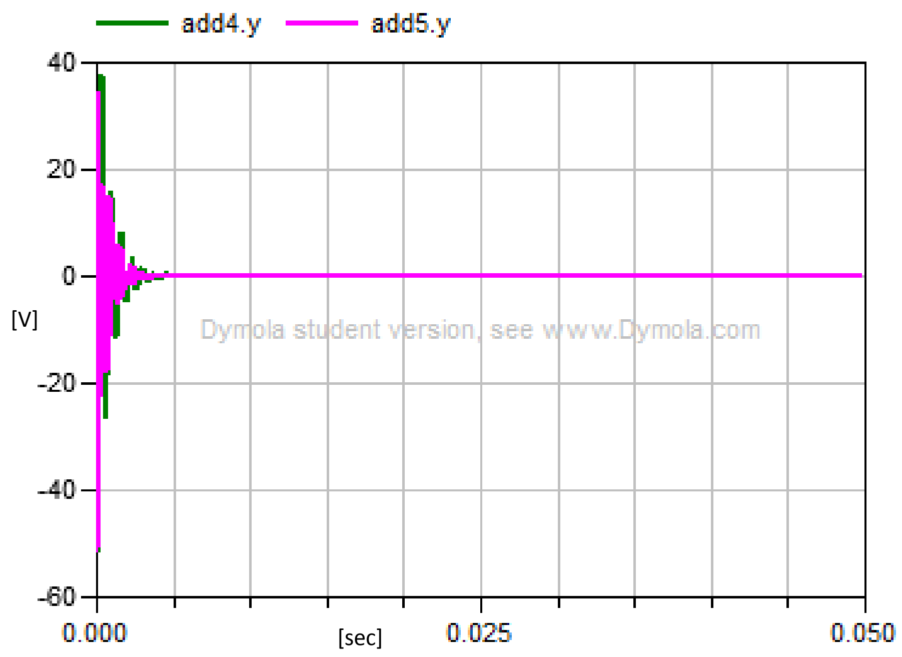


Figure 4.13: Sum of the outputs of PD and FFC controllers with (OTC) controller

On the other hand, when the neural network OTC controller is added to the system, the control signals supplied by the linear PD and FFC controllers began decreasing dramatically until they both have reached the value of $0.006\ V$ within a period of about $9\ mSec$ as shown in figure (4.13). This is also another indication that the neural network not only did compensate for the disturbance torque differences but also effectively compensated for the inaccurate parameters of the driving motors that were used previously for designing both the PD and FFC controllers.

Figure (4.14) shows a 3D visualization of the trajectory followed by the end-effector in the case of using the (OTC) controller. Comparing the figure (4.14) with the figures (3.15) and (3.17), it can be seen that the slight deviation of the end-effector from its steady state trajectory appearing in figure (3.15) did not appear in figure (4.14). Also, the large deviation of the end-effector from the desired trajectory in figure (3.17) is effectively handled by the use of the (OTC) controller as shown by the smooth trajectory in figure (4.14).

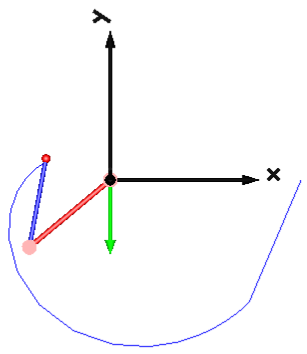


Figure 4.14: Trajectory of the arm with the (OTC) controller

4.4.2 Non-model based neural network control (Feedback Error Learning (FEL))

As mentioned earlier, the key strength of a non-model based neural network control strategy is that it does not require the knowledge of any parameter of the joint driving motors or the arm links. There is no need for developing a mathematical model for either the motor or the robotic arm.

In this case, the neural network of the non-model based controller would be responsible for both identifying the models of the robotic arm and the motors as well as compensating for the structured and unstructured uncertainties.

One of the non-model based neural control configurations used for the trajectory tracking control of robotic manipulators is the feedback error learning structure (FEL) as in [26]. In this configuration, a feed-forward multiple-layer perceptron is placed in the feed-forward path of the PD-controlled motors as shown in figure (4.15).

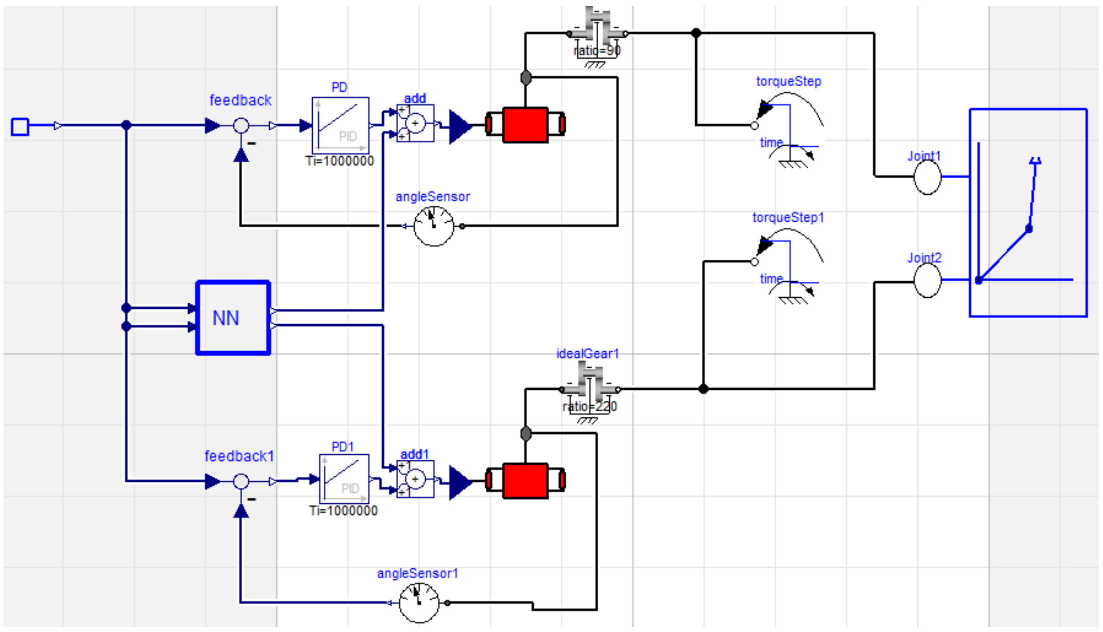


Figure 4.15: Dymola model of the robotic arm with (FEL) controller

As shown in figure (4.15), neither the computed torque disturbance rejection controller nor the (FFC) controller is used in the arm control system. It is also important to note that the parameters of the arm and the joint driving motors are assumed to be completely unknown.

It is shown in the figure that the (FEL) controller receives the desired trajectories for both joint driving motors as inputs and generates two control outputs to be added to the outputs of the PD controllers.

As mentioned earlier, the primary goal of using the neural (FEL) controller is to identify the inverse dynamics of the DC motor-controlled robotic arm in which case the joint driving motors would accurately follow their desired position trajectories. This implies that the outputs of the PD controllers need to be minimized to become as perfectly as zero.

In order to do so, the multiple-layer perceptron of the FEL controller is trained to minimize its performance index function which is defined as the sum of the squares of the outputs obtained from the PD controllers.

Like the model-based (OTC) controller shown in figure (4.7), the (FEL) controller consists of an output layer containing two neurons which are responsible for generating the two control outputs, and a hidden layer consisting of three neurons.

Again, the number of neurons in the hidden layer of a perceptron must be chosen to suit the complexity of the function to be approximated by the network at the same time of preserving the generalization capability of the network. For the purpose of a later comparison with the model-based (OTC) controller, the same number of neurons is chosen to be contained in the hidden layer of the (FEL) controller.

The neurons of the output layer both have a pure linear activation function, whereas the neurons of the hidden layer have a log-sigmoid activation function each. The Modelica code and Dymola icon developed for the (FEL) controller are found in Appendix D.

The learning algorithm used for updating the weights and biases of the two-layer perceptron of the (FEL) controller is the SDA algorithm derived in (4.14 - 4.17). The learning rate α used in the rules of the SDA algorithm is chosen to be 0.1.

The Modelica code used to simulate the learning process of the (FEL) controller by the SDA algorithm is similar to the one used for the learning process of the model-based (OTC) controller. The only difference is that the weights and biases of the (FEL) controller are initialized to different values at the outset of the learning process. Also, the performance index function to be minimized by the (FEL) controller is defined to be the sum of the squares of the outputs of the PD controllers.

In order to test the performance of the designed FEL controller in compensating for the structured and unstructured uncertainties, the PD-controlled robotic system is purposely made to experience the same structured and unstructured uncertainties used for the testing of the neural (OTC) controller. Figure (4.16) shows the motor angular positions after using the designed (FEL) controller.

It is clear from figure (4.16) that the designed neural network (FEL) controller did enforce the actual angular positions of the motors to follow the desired pick-and-place trajectory with an acceptable precision despite the presence of both structured and unstructured uncertainties in the system.

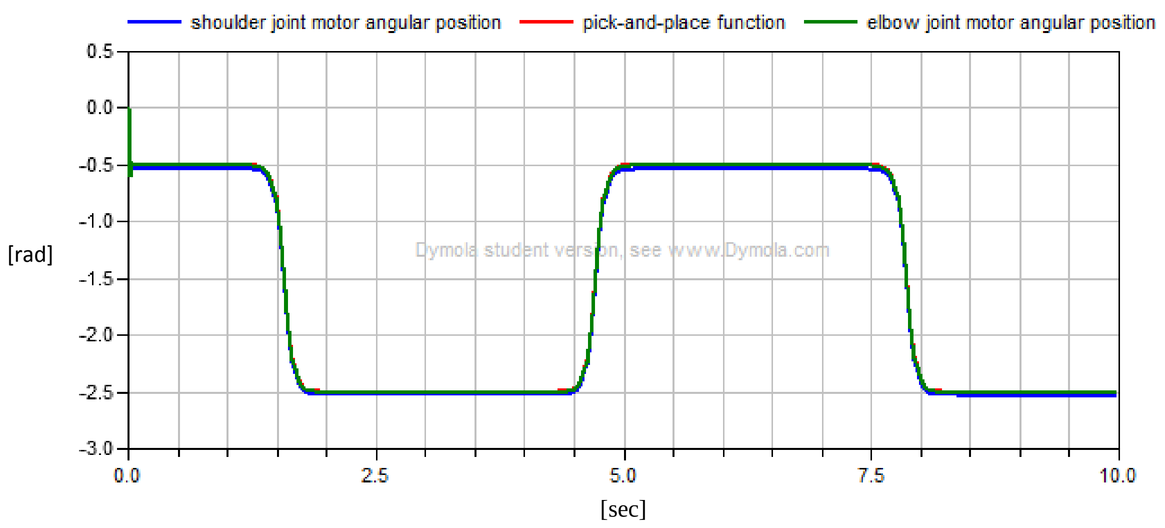


Figure 4.16: Angular positions of motors with FEL controller

The steady state errors in the horizontal and vertical parts of the response for the shoulder joint driving motor are about $0.04\ rad$ and $0.03\ rad$, respectively. For the elbow joint driving motor, the steady state errors in the horizontal and vertical parts of the response are about $0.005\ rad$ and $0.001\ rad$, respectively.
Table (4.1) shows the steady state errors in the horizontal and vertical parts of the response for both motors when using different control mechanisms. By comparing these results, it can be shown that the designed (FEL) controller reduced the steady state error in the horizontal part of the response of the shoulder joint driving motor by 92% and that of the elbow joint driving motor by 87% from the case of using the computed torque disturbance rejection and feed-forward (FFC) control in the presence of the structured uncertainties only. The steady state error in the vertical part of the response was improved by (FEL) controller by 94% for the shoulder joint driving motor and 99% for the elbow joint driving motor.

Table 4.1: Steady state errors for different control mechanisms

Control mechanism	Shoulder joint motor position		Elbow joint motor position	
	Horizontal part of response	Vertical part of response	Horizontal part of response	Vertical part of response
Disturbance rejection, feed-forward (FFC) with structured uncertainties	$0.5\ rad$	$0.5\ rad$	$0.04\ rad$	$0.12\ rad$
Neural network (FEL) with structured and unstructured uncertainty	$0.04\ rad$	$0.03\ rad$	$0.005\ rad$	$0.001\ rad$

In order to test the capability of the (FEL) controller in identifying the inverse dynamics of the robotic arm, the outputs of the PD controllers are plotted in figure (4.17). It is clearly seen from figure (4.17) that the (FEL) controller is gradually learning the inverse dynamics of the robotic arm as approved by the gradual decrease

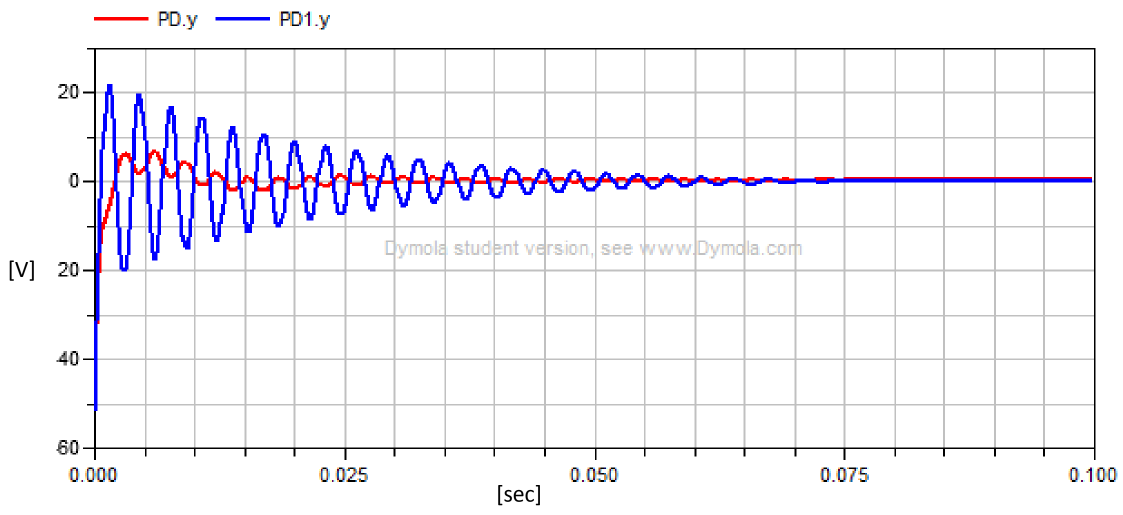


Figure 4.17: Outputs of PD controllers with the (FEL) controller

of the control signals supplied to the driving motors by the PD controllers. Both PD control signals continued to decay until they have reached the values of $0.36\ V$ for the the shoulder joint driving motor and $0.042\ V$ for the elbow joint driving motor within a period of $77\ mSec$. These results also imply that the (FEL) controller is gradually taking over the full control of the joint trajectory tracking system without getting any help from the PD controllers.

Chapter 5

Conclusion and Future Work

This thesis investigated the possibility of improving the trajectory tracking performance of a 2-DOF robotic manipulator using different configurations of neural network controllers. These configurations were classified into model-based and non-model based structures. A model-based control strategy required the presence of a mathematical model for the controlled manipulator and therefore considered to be highly complicated and time consuming for higher degree of freedom manipulators. A non-model based control strategy did not require a prerequisite knowledge of the parameters of either the manipulator or the driving motors and hence no mathematical model for the manipulator was needed.

The performance of each neural network based control strategy was compared with that of the conventional computed torque control method through carrying out several simulations of the robotic arm under the Dymola simulation environment based on the Modelica language. The simulation results obtained proved the superiority of the designed neural network-based controllers over a conventional computed torque disturbance rejection controller in compensating for both structured and unstructured uncertainties.

The non-zero position errors obtained by using the designed non-model based neural (FEL) controller were caused by the insufficient number of weights updating iterations used in the learning process of the neural network. It can be claimed that further increasing the number of weight updating iterations will enable the neural network controller to improve its learning of the arm inverse dynamics and hence would generate more accurate actuating torques which will result in further reduction of the driving motor position errors.

In addition, it was mentioned in chapter 4 that the error backpropagation SDA algorithm used in the training process of the designed neural networks in this thesis is considered the least efficient among other backpropagation algorithms due to its highly slow convergence rate. In a future work, the performance of the designed neural network controllers will be tested with the use of more efficient and faster learning algorithms such as Levenberg and Marquardt algorithms.

One of the planned future works is to employ the designed neural network controllers (OTC and FEL) to control the trajectory tracking performance of higher degree of freedom manipulators such a Self Compliant Articulated Robotic Arm (SCARA) and a PUMA 560 manipulator. Other neural network based control strategies such as reference compensation technique will be also employed, tested, and compared with the OTC and FEL controllers.

RFERENCES

[1] Mohsen Charmanirad. "Design and implementation of controller for robotic manipulators using Neural Networks", Master thesis, Malardalen University, 2009.

[2] Mark W. Spong, Seth Hutchinson, M. Vidyasagar. *Robot Modeling and Control*. John Willey and Sons, INC, New York, 2005.

[3] Martin T. Hagan, Howard B. Demuth, Mark Beale. *Neural Network Design*. PWS Publishing Company, USA, 1996.

[4] Ahmed Alassar. "Modeling and Control of 5-DOF Robot Arm Using Supervisory Control", Master thesis, Islamic University of Gaza, 2010.

[5] H. Delavari, R. Ghaderi, A. Ranjbar N., S.H. HosseinNia, S. Momani. "Adaptive Fractional PID Controller for Robot Manipulator", *Proceedings of FDA'10. The 4th IFAC Workshop Fractional Differentiation and its Applications*. Spain, October 2010.

[6] Jafar Tavoosi, Afshar Shamsi Jokandan, Muhammad Amin Daneshwar. "A new method for position control of a 2-DOF robot arm using neuro-fuzzy controller", *Indian Journal of Science and Technology,* Vol. 5, No. 3, March, 2012.

[7] Refaat S. Ahmed, Kuldip S. Rattan, Omar H. Abdallah. "Adaptive Neural Network for Identification and Tracking Control of a Robotic Manipulator", *Proceedings of the IEEE 1995 National Aerospace and Electronics Conference,* Vol. 5, pp 601-605, 1995.

[8] Tomachika Ozaki, Tatsuya Suzuki, Takeshi Furuhashi, Shigeru Okuma, Yoshiki Uchikawa. "Trajectory Control of Robotic Manipulators Using Neural Networks". *IEEE Transactions on Industrial Electronics,* Vol. 38, No. 3, June 1991.

[9] Joel Perez P., Jose P. Perez, Rogelio Soto, Angel Flores, Francisco Rodriguez, Jose Luis Meza, "Trajectory Tracking Error Using PID Control Law for Two-Link Robot Manipulator via Adaptive Neural Networks", *Procedia Technology,* Vol. 3, pp. 139-146, 2012.

[10] Kamel Kara, Tedji Eddine Missoum, Kamel Eddine Hemsas, Mohamed Laid Hadjili, "Control of a Robotic Manipulator Using Neural Network Based Predictive Control", *17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS),* pp. 1104-1107, 2010.

[11] Tzu-Chun Kuo, Ying-Jeh Huang, Chin-Yun Wang. "Real-time Learning Controller Design For a Two-Link Robotic Arm". *Proceedings of the Sixth International Conference on Machine Learning and Cybernetics,* Hong Kong, 19-22 August, 2007.

[12] Zhao-Hui Jiang, Taiki Ishita "A Neural Network Controller for Trajectory Control of Industrial Robot Manipulators", *Journal of Computers*, Vol. 3, No. 8, August 2008.

[13] Bruno Siciliano, Oussama Khatib. *Handbook of Robotics*. Springer, Berlin, 2008.

[14] Dassault Systems AB, *Dymola (Dynamic Modeling Laboratory) – Getting Started with Dymola*, Dassault Systems AB, Sweden, March 2013.

[15] Michael Tiller. *Introduction to Physical Modeling with Modelica*. Kluwer Academic Publishers, Massachusetts, 2001.

[16] Katsuhiko Ogata. *Matlab for Control Engineers*. Prentice Hall Inc., New Jersey, October, 2007.

[17] Benjamin C. Kuo. *Automatic Control Systems*. Prentice Hall Inc., New Jersey, 1995.

[18] Richard C. Dorf, Robert B. Bishop. *Modern Control Systems*. Addison-Wesley Longman, Inc., California 1998.

[19] Richard P. Lippmann, "Pattern Classification Using Neural Networks", *IEEE Communications Magazine*, Vol. 27, Issue 11, pp. 47-50, November 1989.

[20] Hartati R.S., El-Hawary M.E., "New Approach for Solving Optimization Problems in Economic Load Dispatch using Hopfield Neural Networks", *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, Vol. 2, pp. 722-725, 2000.

[21] Wei Xu, Tingting Zheng, Ziang Li, "A Neural Network Based Forecasting Method For the Unemployment Rate Prediction Using the Search Engine Query Data", *IEEE 8th International Conference on e-Business Engineering (ICEBE),* Beijing, 19-21 October, 2011.

[22] Anil K. Jain, Jianchang Mao, "Artifical Neural Networks: A Tutorial", *Journal of IEEE Computer*, Vol. 29, pp. 31-44, March 1996.

[23] A. Ravindran, K. M. Ragsdell, G. V. Reklaitis. *Engineering Optimization – Methods and Applications*. John Wiley & Sons, *Inc.*, New Jersey, 2006.

[24] Suel Jung, "Neural Network Controllers for Robot Manipulators". PhD Dissertation, University of California, 1996.

[25] Suel Jung, T.C. Hsia, "Neural Network Reference Compensation Technique for Position Control of Robot Manipulators", *Proceedings of the IEEE International Conference on Neural Networks*, Vol. 3, pp. 1765-1770, 1996.

APPENDIX A

Derivation of the Open Loop Transfer Function for the DC-Motor

The following analysis discusses the procedure of deriving the transfer function from the motor angular position θ_m to the input voltage signal applied to the motor armature circuit $V(t)$.

Referring to the armature circuit shown in figure (2.6), the dynamic equation of the DC-motor can be derived as follows:

$$L\frac{di_a}{dt} + Ri_a = V - V_b \text{ --------- } (A.1)$$

The torque applied to the motor inertia τ_m is directly proportional to the armature current i_a through the motor constant K_m. The back-electromotive force voltage V_b is directly proportional to the angular velocity of the motor through the proportionality constant K_b. This can be mathematically written as:

$$\tau_m = K_m\, i_a \text{ -------- } (A.2)$$

$$V_b = K_b\frac{d\theta_m}{dt} \text{ -------- } (A.3)$$

From figure (2.7), the mathematical equation that describes the mechanical behavior of the DC-motor is given as follows:

$$J_m\frac{d^2\theta_m}{dt^2} + B_m\frac{d\theta_m}{dt} = \tau_m - \frac{\tau_L}{r} \text{ -------- } (A.4)$$

Where J_m denotes the motor inertia, B_m denotes the motor damping coefficient, τ_m is the motor torque and τ_L denotes the torque generated by the load.

As mentioned in Chapter 2, the motor is connected to the load through a gear train with gear reduction ratio $r:1$. The main goal of using a gear train in a DC-motor is to enable the motor to drive large loads which require the generation of large torques. So, the gear train with the gear reduction ratio $(r:1)$ magnifies the torque of the motor and reduces its angular position by r times as shown in figure (2.7).

Taking the Laplace transform of the electrical characteristic equation (A.1) reveals the following:

$$LSI_a(s) + RI_a(s) = V(s) - V_b(s) \text{ -------- } (A.5)$$

$V_b(s)$ can be found by taking the Laplace transform of the equation (A.3) as follows:

$$V_b(s) = K_b S\theta_m(s) - - - - - (A.6)$$

Substituting (A.6) into (A.5) gives the following:

$$(LS + R)I_a(s) = V(s) - K_b S\theta_m(s) - - - - - - - (A.7)$$

By letting $\tau_L = 0$ and taking the Laplace transform of the mechanical characteristic equation (A.4) and using the equation (A.2), the following equation is revealed:

$$J_m S^2\theta_m(s) + B_m S\theta_m(s) = K_m I_a(s)$$

$Therefore\,,$

$$(J_m S^2 + B_m S)\theta_m(s) = K_m I_a(s) \ - - - - - - - - -(A.8)$$

By using the equations (A.7) and (A.8), the transfer function $\frac{\theta_m(s)}{V(s)}$ is found as follows:

$$\frac{\theta_m(s)}{V(s)} = \frac{K_m}{(LS + R)(J_m S^2 + B_m S) + K_b K_m S} - - - - - - - - (A.9)$$

APPENDIX B

PD Controller Design Using the Root Locus Method

B.1: Root locus method

Let $L(s)$ be the open-loop transfer function of a system $G(s)$. Then, the closed-loop system transfer function is given by:

$$G_{cl}(s) = \frac{L(s)}{1+L(s)} - - - - - - - - (B.1)$$

In order to find the closed-loop poles of the system, the following equation must be achieved:

$$1 + L(s) = 0$$

Therefore,

$$L(s) = -1$$

$hence\ \|L(s)\| = 1$ and the phase angle of $L(s) = (2k+1)180^0$ where $k = \{0, \pm 1, \pm 2, \pm 3, \ldots\ldots\} - - - - - - - - (B.2)$

This implies that in order for a design point to be located on the root locus of the open-loop system, it should achieve the above magnitude and phase requirements of $L(s)$.

The question now is how to find the design point that helps to achieve the time response requirements which are mainly the settling time t_s and the peak overshoot $P.O.$ The answer to this question is explained as follows:

The settling time t_s is defined as the time required for the step response of the closed-loop system to reach and stay within the region of 2% of the steady state value [4,18,19]. The equation used to compute the settling time is as follows:

$$t_s = \frac{4}{\rho_d \omega_n} - - - - - - - - (B.3)$$

Where ρ_d denotes the damping ratio and ω_n is the natural un-damped frequency in $rad/sec.$

The peak overshoot $P.O.$ is defined as the magnitude by which the step response of the system overshoots the steady state value [4,18,19,20]. The peak overshoot requirement is sometimes given as a percentage of the required steady state value. The equation used to compute the peak overshoot is given as follows:

$$P.O. = e^{-\frac{\pi\rho}{\sqrt{1-\rho^2}}} ------(B.4)$$

Equations (B.3) and (B.4) can be used to compute the corresponding damping ratio ρ and un-damped frequency ω_n for specific settling time and overshoot requirements.

After finding the required damping ratio and un-damped frequency, the design point (S_D) that needs to be located on the root locus of the open-loop system for some gain K in order to achieve the required settling time and peak overshoot can be calculated as follows:

$$S_D = -\rho_d\omega_n + j\omega_n\sqrt{1-\rho^2} ----- (B.5)$$

B.2: Computation of the design point S_D used in the PD controller design

Given the step response requirements mentioned in section 3.1, the corresponding damping ratio ρ_d and un-damped frequency ω_n are calculated as follows:

From equation (B.3), we have the following:

$$t_s = \frac{4}{\rho_d\omega_n} = 0.02$$

Therefore,

$$\rho_d\omega_n = 200 \ ------(B.6)$$

From equation (B.4), we have the following:

$$P.O. = e^{-\frac{\pi\rho}{\sqrt{1-\rho^2}}} = 0.0001$$

Therefore,

$$\rho_d = 0.95 \ ------(B.7)$$

By substituting the value of the damping ratio ρ_d from (B.7) into (B.6), the un-damped frequency ω_n can be revealed as follows:

$$\omega_n = \frac{200}{\rho_d} = \frac{200}{0.95} = 210.53\,\frac{rad}{sec} - - - - - - - - - (B.8)$$

By substituting equations (B.7) and (B.8) into equation (B.5), the design closed-loop system pole S_D that achieves the time response requirements is found to be:

$$S_D = -200 + j65.74 - - - - - - - - (B.9)$$

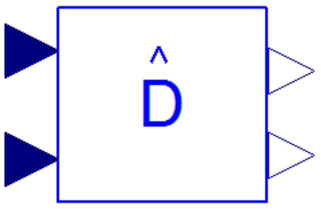
APPENDIX C

Modelica Models Used in the Computed Torque-based Controlled Design

C.1: Modelica model of the estimated inertia matrix

```
model Estimated_D
  import Modelica.Math;

  Modelica.Blocks.Interfaces.RealInput theta1
                                        a;
  Modelica.Blocks.Interfaces.RealOutput y1
                                        a;
  Modelica.Blocks.Interfaces.RealOutput y2 a;
  Modelica.Blocks.Interfaces.RealInput theta2 a;

parameter Real m1=1.95;
parameter Real a1=0.25;
parameter Real I=0.0980;
parameter Real m2=0.93;
parameter Real a2=0.15;

Real d11;
Real d12;
Real d22;
Real d21;
Real theta1dot;
Real theta2dot;

equation
  theta1dot=der(theta1);
  theta2dot=der(theta2);
  d11=(1/90)*(0.25*m1*a1^2+m2*(a1^2+0.25*a2^2+a1*a2*cos(theta2))+2*I);
  d12=(1/90)*(m2*(0.25*a2^2+0.5*a1*a2*cos(theta2))+I);
  d22=(1/220)*(m2*0.25*a2^2+I);
  d21=(1/220)*(m2*(0.25*a2^2+0.5*a1*a2*cos(theta2))+I);
  y1=d11*der(theta1dot)+d12*der(theta2dot);
  y2=d21*der(theta1dot)+d22*der(theta2dot);

  a;
end Estimated_D;
```

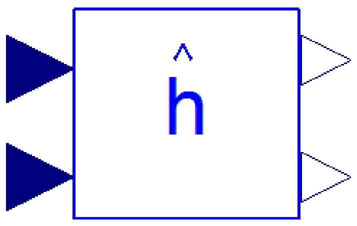
C.2: Dymola icon for the estimated inertia matrix



C.3: Modelica Model for the estimated vector of centrifugal, coriolis, and gravitational forces

```
model Estimated_h
import Modelica.Math;

  Modelica.Blocks.Interfaces.RealInput theta1 a;
  Modelica.Blocks.Interfaces.RealInput theta2 a;
  Modelica.Blocks.Interfaces.RealOutput y1 a;
  Modelica.Blocks.Interfaces.RealOutput y2 a;

parameter Real m1=1.95;
parameter Real a1=0.25;
parameter Real m2=0.93;
parameter Real a2=0.15;
constant Real g=9.81;

Real h11;
Real h21;
Real theta2dot;
Real theta1dot;

equation
  theta1dot=der(theta1);
  theta2dot=der(theta2);
  h11=(1/90)*(-(m2*a1*a2*sin(theta2)*theta1dot*theta2dot)-(m2*a1*0.5*a2*sin(theta2)*theta2dot^2) +
       m1*g*0.5*a1*cos(theta1)+m2*g*(0.5*a2*cos(theta1+theta2)+a1*cos(theta1)));

  h21=(1/220)*((m2*0.5*a1*a2*sin(theta2)*theta1dot^2)+(m2*g*0.5*a2*cos(theta1+theta2)));
  y1=h11;
  y2=h21;

  a;
end Estimated_h;
```

C.4: Dymola icon for the estimated vector of centrifugal, coriolis, and gravitational forces



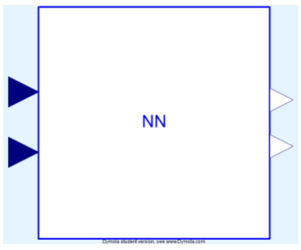
APPENDIX D

Modelica Models Used in the Design of ANN Controllers

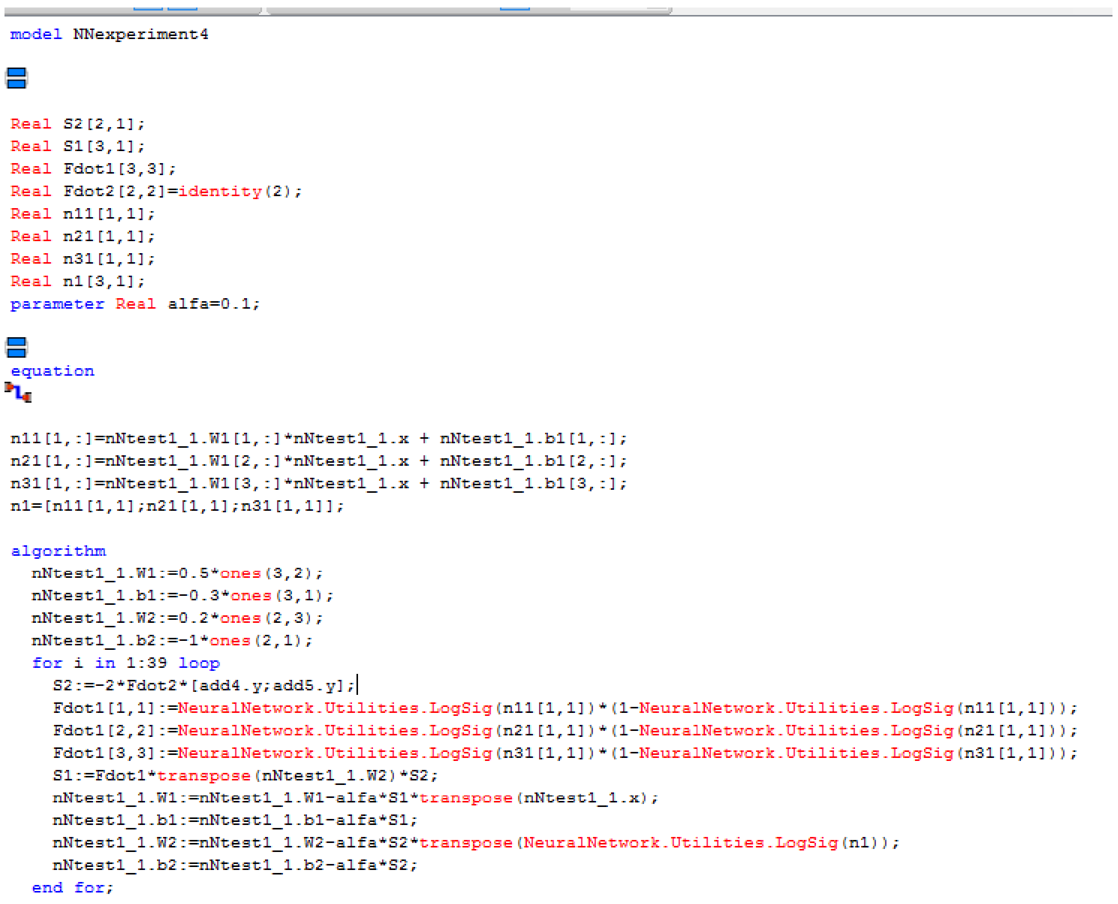
D.1: Modelica code of the ANN used in both (OTC) and (FEL) controllers

```
model NNtest1

  Modelica.Blocks.Interfaces.RealInput u
    a;
  Modelica.Blocks.Interfaces.RealInput u1
    a;
  Modelica.Blocks.Interfaces.RealOutput y
    a;
  Modelica.Blocks.Interfaces.RealOutput y1
    a;

Real W1[3,2];
Real b1[3,1];
Real W2[2,3];
Real b2[2,1];
Real a2[2,1];
Real x[2,1];

equation
  x=[u;u1];
  a2=W2*NeuralNetwork.Utilities.LogSig(W1*x+b1) +b2;
  y=a2[1,1];
  y1=a2[2,1];
  a;
end NNtest1;
```

D.2: Dymola icon of the ANN used in the OTC controller

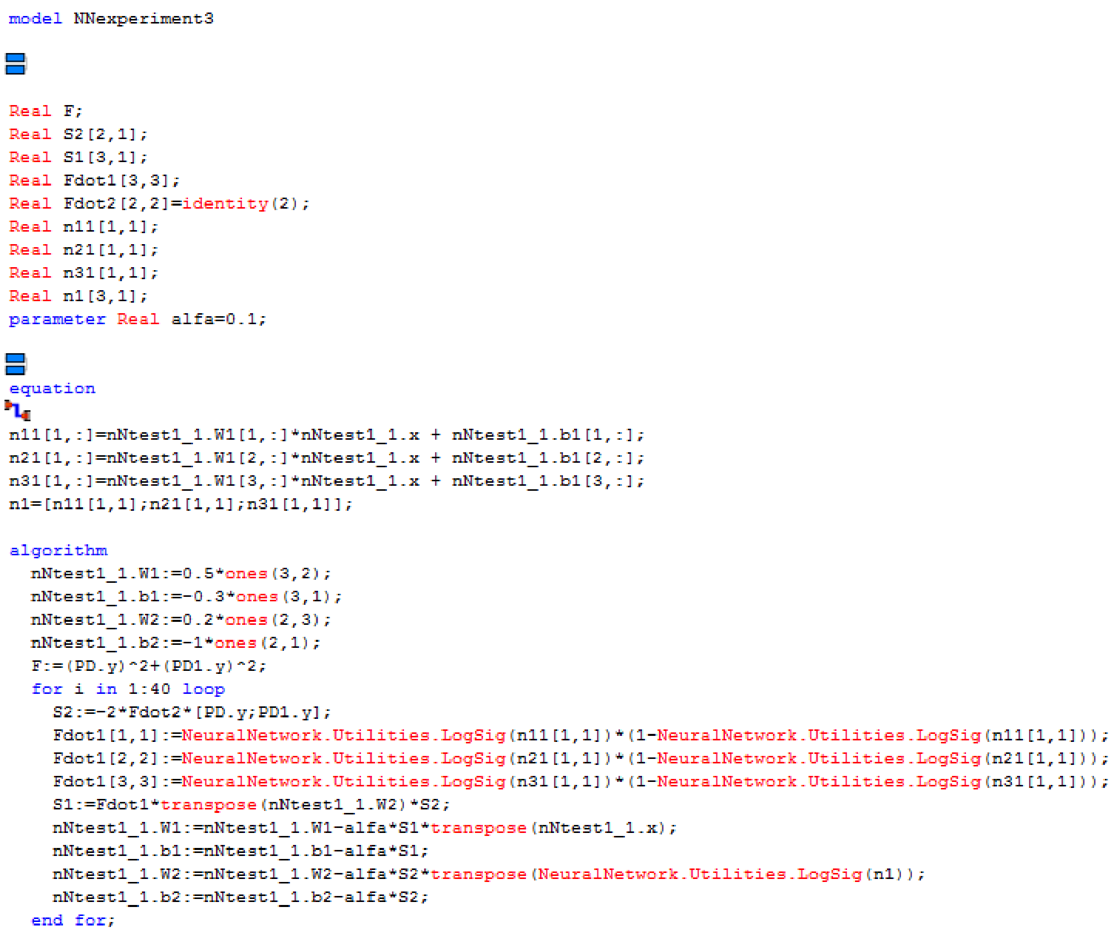


D.3: Modelica code of the EBA algorithm used for training the ANN of the OTC controller



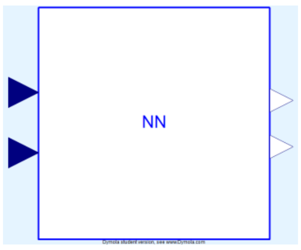
```
model NNexperiment4

Real S2[2,1];
Real S1[3,1];
Real Fdot1[3,3];
Real Fdot2[2,2]=identity(2);
Real n11[1,1];
Real n21[1,1];
Real n31[1,1];
Real n1[3,1];
parameter Real alfa=0.1;

equation

n11[1,:]=nNtest1_1.W1[1,:]*nNtest1_1.x + nNtest1_1.b1[1,:];
n21[1,:]=nNtest1_1.W1[2,:]*nNtest1_1.x + nNtest1_1.b1[2,:];
n31[1,:]=nNtest1_1.W1[3,:]*nNtest1_1.x + nNtest1_1.b1[3,:];
n1=[n11[1,1];n21[1,1];n31[1,1]];

algorithm
  nNtest1_1.W1:=0.5*ones(3,2);
  nNtest1_1.b1:=-0.3*ones(3,1);
  nNtest1_1.W2:=0.2*ones(2,3);
  nNtest1_1.b2:=-1*ones(2,1);
  for i in 1:39 loop
    S2:=-2*Fdot2*[add4.y;add5.y];
    Fdot1[1,1]:=NeuralNetwork.Utilities.LogSig(n11[1,1])*(1-NeuralNetwork.Utilities.LogSig(n11[1,1]));
    Fdot1[2,2]:=NeuralNetwork.Utilities.LogSig(n21[1,1])*(1-NeuralNetwork.Utilities.LogSig(n21[1,1]));
    Fdot1[3,3]:=NeuralNetwork.Utilities.LogSig(n31[1,1])*(1-NeuralNetwork.Utilities.LogSig(n31[1,1]));
    S1:=Fdot1*transpose(nNtest1_1.W2)*S2;
    nNtest1_1.W1:=nNtest1_1.W1-alfa*S1*transpose(nNtest1_1.x);
    nNtest1_1.b1:=nNtest1_1.b1-alfa*S1;
    nNtest1_1.W2:=nNtest1_1.W2-alfa*S2*transpose(NeuralNetwork.Utilities.LogSig(n1));
    nNtest1_1.b2:=nNtest1_1.b2-alfa*S2;
  end for;
```

D.4: Modelica code of the EBA algorithm used for training the ANN of the FEL controller



```
model NNexperiment3

Real F;
Real S2[2,1];
Real S1[3,1];
Real Fdot1[3,3];
Real Fdot2[2,2]=identity(2);
Real n11[1,1];
Real n21[1,1];
Real n31[1,1];
Real n1[3,1];
parameter Real alfa=0.1;

equation
n11[1,:]=nNtest1_1.W1[1,:]*nNtest1_1.x + nNtest1_1.b1[1,:];
n21[1,:]=nNtest1_1.W1[2,:]*nNtest1_1.x + nNtest1_1.b1[2,:];
n31[1,:]=nNtest1_1.W1[3,:]*nNtest1_1.x + nNtest1_1.b1[3,:];
n1=[n11[1,1];n21[1,1];n31[1,1]];

algorithm
  nNtest1_1.W1:=0.5*ones(3,2);
  nNtest1_1.b1:=-0.3*ones(3,1);
  nNtest1_1.W2:=0.2*ones(2,3);
  nNtest1_1.b2:=-1*ones(2,1);
  F:=(PD.y)^2+(PD1.y)^2;
  for i in 1:40 loop
    S2:=-2*Fdot2*[PD.y;PD1.y];
    Fdot1[1,1]:=NeuralNetwork.Utilities.LogSig(n11[1,1])*(1-NeuralNetwork.Utilities.LogSig(n11[1,1]));
    Fdot1[2,2]:=NeuralNetwork.Utilities.LogSig(n21[1,1])*(1-NeuralNetwork.Utilities.LogSig(n21[1,1]));
    Fdot1[3,3]:=NeuralNetwork.Utilities.LogSig(n31[1,1])*(1-NeuralNetwork.Utilities.LogSig(n31[1,1]));
    S1:=Fdot1*transpose(nNtest1_1.W2)*S2;
    nNtest1_1.W1:=nNtest1_1.W1-alfa*S1*transpose(nNtest1_1.x);
    nNtest1_1.b1:=nNtest1_1.b1-alfa*S1;
    nNtest1_1.W2:=nNtest1_1.W2-alfa*S2*transpose(NeuralNetwork.Utilities.LogSig(n1));
    nNtest1_1.b2:=nNtest1_1.b2-alfa*S2;
  end for;
```

D.5: Dymola icon of the ANN used in the FEL controller


NN
Dymola student version, see www.Dymola.com