MOBILE ROBOT TRAJECTORY TRACKING USING NEURAL NETWORKS

A THESIS IN MECHATRONICS

Presented to the faculty of the American University of Sharjah
College of Engineering
in partial fulfillment of
the requirement of the degree

MASTER OF SCIENCE

by
OMID MOHARERI

Sharjah, U.A.E
December 2009

We approve the thesis of Omid Mohareri

Date of signature

_____ _____

Dr. Rached Dhaouadi
Associate Professor, Department of Electrical Engineering
Thesis Advisor

_____ _____

Dr. Aydin yesildirek,
Associate Professor, Department of Electrical Engineering
Graduate Committee

_____ _____

Dr. Nabil Abdel Jabbar,
Associate Professor, Department of Chemical Engineering
Graduate Committee (External Examiner)

_____ _____

Dr. Rached Dhaouadi,
Coordinator
Mechatronics Engineering Graduate Program

_____ _____

Dr. Hany El Kadi,
Associate Dean, College of Engineering

_____ _____

Dr. Yousef Al Assaf,
Dean, College of Engineering

_____ _____

Mr. Kevin Mitchell,
Director, Graduate & Undergraduate Programs

MOBILE ROBOT TRAJECTORY TRACKING USING NEURAL NETWORKS

Omid Mohareri, Candidate for the Master of Science Degree

American University of Sharjah, 2009

ABSTRACT

This work primarily addresses the design and implementation of a neural network based controller for the trajectory tracking of a differential drive mobile robot. Two different neural network based tracking control algorithms are proposed and their simulation and experimental results are presented. The first algorithm is a control structure that makes possible the integration of a backstepping controller and a neural network (NN) computed-torque controller for a nonholonomic mobile robot. Integration of a neural network controller and the kinematic based controller gives the advantage of dealing with unmodeled and unstructured uncertainties and disturbances to the system.

The second proposed control algorithm is an NN-based adaptive controller which tunes the gains of the backstepping controller online according to the robot reference trajectory and its initial posture. In this method, a neural network is needed to learn the characteristics of the plant dynamics and make use of it to determine the future inputs that will minimize error performance index so as to compensate the backstepping controller gains. The advantages and disadvantages of the two proposed control algorithms will be discussed in each section with illustrations.

Comprehensive system modeling including robot kinematics, dynamics and actuator modeling has been done. The dynamic modeling is done using Newtonian and Lagrangian methodologies for nonholonomic systems and the results are compared to verify the accuracy of each method. Simulation of the robot model and different controllers has been done using Matlab and Matlab Simulink.

The proposed trajectory tracking controllers has been tested on ERA-MOBI mobile robot platform which is a full featured industrial mobile robot in AUS Mechatronics center and all trajectory tracking results with different reference trajectories are presented.

Thesis Supervisor: Dr. Rached Dhaouadi
Associate Professor
Electrical Engineering department
Coordinator, Mechatronics graduate program, AUS

CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENT

My debt to my parents and my family is definitely beyond measure and I will be forever grateful for what they have sacrificed for me to get to this point. This work will remain as a great honor of my life and my heart is filled with nothing but gratefulness to all of you.

My great gratitude goes primarily to my supervisor Dr.Rached Dhaouadi for his support, guidance and enthusiastic assistance to my thesis. I appreciate his time, effort, suggestions and comments during this work.

I would like to give my special thanks to my colleagues in AUS Mechatronics center specially, my best friends, Engineer Hossein Sadjadi and Amir Hossein Jaffari for their support and honest friendship.

Many thanks are given to all my friends at American University of Sharjah in particular: Shahin Dehbashi, Amin Bigdeli, Mohammad Reza Karbassi and Mona Saghafi for their kind, thoughtful and loving friendship.

DEDICATION

It is my biggest joy and privilege to dedicate this work to my thoughtful, caring and compassionate parents, Farzad Mohareri and Maryam Mirahmadi.

Chapter 1

INTRODUCTION

This thesis primarily aims to present the design and development of a Neural Network based controller for trajectory tracking of a nonholonomic differential drive mobile robot, along with its implementation on a commercial mobile robot platform in AUS Mechatronics center. The idea behind the proposed trajectory tracking controllers and their implementation on the physical platform will be clearly addressed in this thesis.

1.1 BACKGROUND

In recent years a plethora of research has been carried out on the control problem of the autonomous mobile robotic systems. This is mainly due to the growing application of these systems both in industrial and service environments. Some typical applications of such systems are for instance order-pick robots in automated warehouses, post delivery robots in office buildings and deep see exploration robots. Different kinds of robots can be used in these applications [1]. In rough terrains, walking robots are usually preferred. On smoother surfaces, wheeled mobile robots have the advantage because they are much faster and more agile. Other kinds of systems such as unmanned aerial vehicles or unmanned under water vehicles are used when we need to maneuver in three dimensional spaces. Wheeled mobile robots are the type of mobile robotic systems that we are considering in this thesis because they are most widely used among the class of mobile robots. This is due to their fast maneuvering, simple controllers and energy saving characteristics [2].

A wheeled mobile robot is a wheeled vehicle which is capable of an autonomous motion without external human drivers, because it is equipped, for its motion, with motors that are driven by an embedded computer. We can categorize the wheeled mobile robot systems into two categories according to their wheel type: the *conventional* wheels and *Swedish* wheels.

For the *conventional* wheel, the contact between the wheel and the ground is supposed to satisfy the pure rolling without slipping constraint. This implies that the velocity of the contact point is equal to zero and the components of this velocity parallel and orthogonal to the plane of wheel are equal to zero. A schematic shape of a conventional wheel is shown in Figure 1-1:

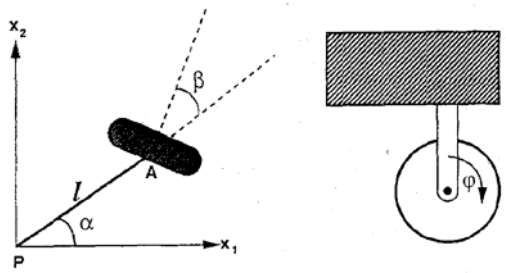

Figure 1-1: the conventional wheel shape

There are three types of conventional wheels: fixed wheels, centered orientable wheels and off-centered orientable wheels. These three types of conventional wheels are shown in Figure 1-2:

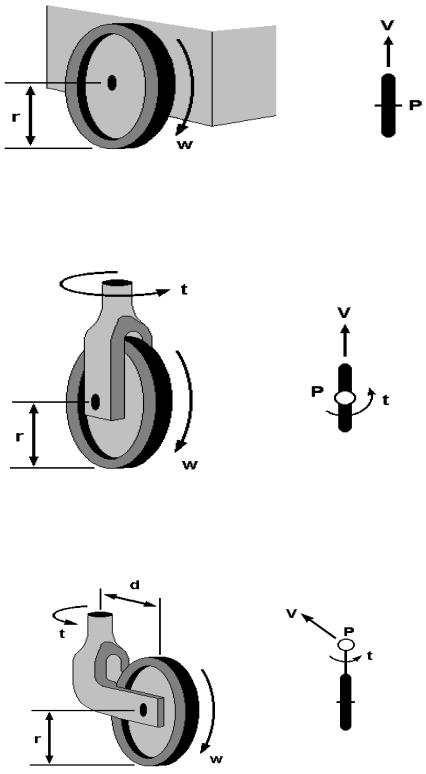

Figure 1-2: different types of conventional wheels

As it can be seen from the above figure, the velocity component which is orthogonal to the wheel plane is zero in all of the above different types of the conventional wheels.

For a *Swedish* wheel, only one component of the velocity of the contact point of the wheel with the ground is supposed to be equal to zero along the motion. The direction of this zero component is arbitrary but is fixed with respect to the orientation of the wheel as can be seen in Figure 1-3:

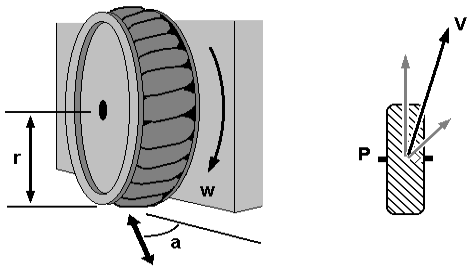

Figure 1-3: the schematic shape of a Swedish wheel

Different types of wheels can be used depending on the application and types of environment. The other important issue about wheeled mobile robots is their locomotion type. Wheeled mobile robots can have the following five types of locomotion system:

- Differential drive
- Steered Wheels: Tricycle, Bicycles and Wagon
- Synchronous drive
- Omni-directional
- Car drive (Ackerman steering)

The above different types of locomotion systems for wheeled mobile robots are shown in Figure 1-4:

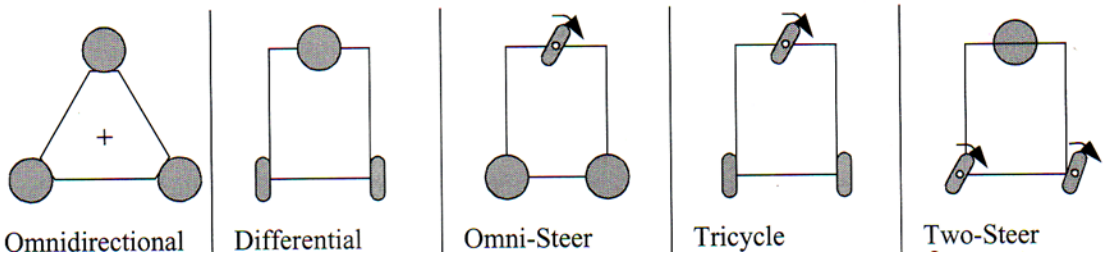

Figure 1-4: different kinds of locomotion types for wheeled mobile robots

The differential drive wheeled mobile robot is the simplest and most popular mobile robot which has the simplest drive mechanism. It is a platform equipped with a front castor and a pair of rear co-axial drive wheels. Each of these drive wheels are independently driven by a DC motor which is in turn energized by a control voltage.

It is known that the stabilization of nonholonomic wheeled mobile robots with restricted mobility to an equilibrium state is in general quit difficult. A well known work of Brockett [3] identifies nonholonomic systems as a class of systems that cannot be stabilized via smooth state feedback. It implies that problems of controlling nonholonomic systems cannot be applied to methods of linear control theory, and they are not transformable into linear control problems. Due to both their richness and hardness, such nonlinear control problems have motivated a large number of researchers involving various techniques of automatic control [4].

Another difficulty in controlling nonholonomic mobile robots is that in the real world there are uncertainties in their modeling. Taking into account intrinsic characteristics of mobile robots such as actual vehicle dynamics, inertia and power limits of actuators and localization errors, their dynamic equations could not be described as a simplified mathematical model. Therefore, the problem of dealing with the system uncertainties is the other problem and challenge in this research field. The above problems and challenges are the main motivations behind

The consequence is that the standard planning and control algorithms developed for robotic manipulators without constraints are no more applicable [5]. This has given rise to recently to an abundant literature dealing with the derivation of planning and control algorithms especially dedicated to specific simplified kinematic models [6].

Several controllers were proposed for mobile robots with nonholonomic constraints, where the two main approaches to controlling mobile robots are posture stabilization and trajectory tracking. The aim of posture stabilization is to stabilize the robot to a reference point, while the aim of trajectory tracking is to have the robot follow a reference trajectory [7] .Recently; interest in tracking control of mobile robots has increased with various theoretical and practical contributions being made. The tracking control approaches for mobile robots are mainly divided to six approaches:

1. State feedback linearization
2. Sliding mode control
3. Backstepping control
4. Computed torque
5. Adaptive control
6. Intelligent control

In these methods, when the stability of the tracking control algorithm, computed load of hardware and operability in the practical application are considered, the backstepping control becomes one of the most popular ones [8]. The major disadvantage of this method is that it needs an accurate modeling and it cannot deal with uncertainties in the model and in the environment. Neural network controllers are proposed in this work to be

combined with the backstepping controller and solve the problem of dealing with unstructured and unmodeled uncertainties or disturbances.

Recently much research has been done on applications of neural networks for control of nonlinear dynamic processes. These works are supported by two of the most important capabilities of neural networks, their ability to learn and their good performance for approximation of nonlinear functions [9]. These two abilities are the main reason behind combining the neural network controller with the backstepping to fix its disadvantage in this thesis.

1.2 PROBLEM STATEMENT

The work includes the development and construction of two different neural network based trajectory tracking controllers. The first controller is a structure that makes possible the integration of a kinematic-based controller and a neural network computed torque controller for a nonholonomic differential drive mobile robot. The combined kinematic and torque control method is developed using backstepping and the system stability is proved using Lyapunov theory. The second control algorithm is a NN-based adaptive controller which tunes the gains of the backstepping controller online according to the robot reference trajectory and its initial posture. In this method, a neural network is needed to learn the characteristics of the plant dynamics and make use of it to determine the future inputs that will minimize error performance index so as to compensate the backstepping controller gains. The controller will be applied to the trajectory tracking problem and the results of its implementation on a mobile robot platform will be presented.

To clarify and explain more details about the above statement regarding the proposed control algorithm and the implementation on the physical platform, this chapter will continue by providing a brief background about wheeled mobile robot systems. Moreover, to position this work, next chapter will thoroughly explore similar related works.

1.3 CONTRIBUTIONS

This thesis in general contributes to the growing field of mobile robot systems. Specific contributions of this thesis are mainly the design and implementation of novel trajectory tracking controllers for nonholonomic mobile robots. Listed below are the detailed contributions:

- Designing a novel NN-based adaptive backstepping controller using the neural network direct model approximation of the robot which highly improves the traditional backstepping controller tracking performance.

- Designing a novel neural network computed torque controller using the neural network inverse model approximation of the robot which improves the tracking performance of the backstepping controller in presence of bounded disturbance and uncertainty in the model.
- Detailed comparison analysis between two traditional tracking controllers and the proposed controller and pointing out the advantages and disadvantages of each one of them.
- Implementation of different trajectory tracking control algorithms on the ERA-MOBI platform.

1.4 THESIS OUTLINE

Having introduced the topic in this chapter, we will continue to have a literature review to look at related works in this field in chapter 2. This chapter will specifically review briefly the history of the field and overview the available different kinds of trajectory tracking controllers for mobile robots. In chapter 3, modeling and simulation of the mobile robot is presented in detail. In chapter 4, the control algorithms will be introduced in details and some simulation results will be presented. Chapter 5 is dedicated to the details of the physical mobile robot platform and implementation of the proposed control algorithms. Finally, chapter 6 will conclude the work by a summary along with the limitations and direction for future research.

CHAPTER 2

RELATED WORK

Much has been written on solving the problem of motion under nonholonomic constraints and so many different types of nonlinear controllers have been proposed in the literature for the trajectory tracking of such robots. Different kinds of trajectory tracking controllers will be briefly explained in this chapter so that the proposed control algorithms in this thesis will be clearer.

2.1 TRACKING CONTROL BASED ON THE ROBOT KINEMATIC MODEL

This trajectory tracking controller was first proposed in [10]. The work mainly proposed a stable tracking control rule for nonholonomic vehicles. Stability of the rule is proved through the use of a Lyapunov function. The inputs to the controller are the vehicle posture and the reference velocities and the output is the target linear and rotational velocities. The control rule proposed in this work was based on the vehicles kinematic model and it does not consider the effects of robot dynamics. This controller is robot independent and can be applied to any kind of mobile robot with dead reckoning ability. The appropriate controller parameters were found out by linearizing the systems differential equations and finding a condition for critical damping. The need for velocity and acceleration limitations is also discussed in this work.

To compute the vehicles control inputs, it is assumed that there is perfect velocity tracking. There are three problems with this approach: first, the perfect velocity tracking problem does not hold in practice. Second, disturbances are ignored and third, complete knowledge of dynamics is needed [11]. In order to improve the performance of this proposed control algorithm, other controllers which include the vehicle dynamics and can deal with disturbances are proposed.

2.2 TRACKING CONTROL BY BACKSTEPPING KINEMATIC INTO DYNAMICS

The work which is a dynamic extension that makes possible the integration of a kinematic controller and a torque controller for nonholonomic mobile robots was first proposed in [12]. A combined kinematic/torque controller is developed using backstepping and the stability is guaranteed using Lyapunov theory. The proposed control algorithm can be applied to three robot navigation problems: tracking a reference trajectory, path following and stabilization about a reference posture. This control algorithm provides a rigorous method of taking into account the specific vehicle dynamics to convert the steering system command into control input for the actual vehicle. First, feedback velocity control inputs are designed for the kinematic steering system to make the position error asymptotically stable. Then, a computed torque controller is designed such that the mobile robot velocities converge to the given velocity inputs. This control approach can be applied to a class of smooth kinematic system control velocity inputs and therefore, the same design procedure works for all of the three basic navigation problems.

This control law which is capable of dealing with the three basic nonholonomic navigation problems and considers the complete dynamics of the mobile robot is derived using the backstepping approach. The method is valid as long as the velocity control inputs are smooth and bounded and the dynamics of the actual cart are completely known. In fact, the perfect knowledge of mobile robot parameters is unattainable. For example friction is very difficult to model in these systems. To confront this problem a robust adaptive controller should be designed so that it can deal with uncertainties and unknown unmodeled disturbances. There no need of a priori information of the dynamic parameters of the mobile robot because the controller will learn them on the fly [12].

Robust control methods such as sliding mode control and adaptive and intelligent control methods such as fuzzy logic and neural network controllers are the possible solutions to this problem and will be explained in the rest of this chapter.

2.3 SLIDING MODE TRAJECTORY TRACKING CONTROLLERS

A robust tracking control for a nonholonomic wheeled mobile robot is proposed in [4] and [13]. A novel sliding control law was proposed in this work for asymptotically stabilizing the mobile robot to a desired trajectory. It is shown that this proposed scheme is robust to bounded external disturbances.

A large difficulty in controlling nonholonomic mobile robots is that in the real world there are uncertainties in their modeling. Taking into account intrinsic characteristics of mobile robots such as actual vehicle dynamics, inertia and power limits of actuators and

localization errors, their dynamic equations could not be described as a simplified mathematical model. The advantages of using sliding mode control for the solution include being fast, having a good transient response and robustness with regard to parameter variations. The dynamic model of the robot is used in this work to describe their behavior with bounded disturbances. By means of a computed torque method, error dynamics of the mobile robot are linearized and sliding mode control law is applied for stabilizing the robot to a reference trajectory and compensating for the existing disturbances. So basically, the proposed control scheme uses the computed torque method for feedback linearization of the dynamic equation and a theory of sliding mode for the robust control.

This proposed control scheme has the ability to solve the trajectory tracking problem based on dynamic modeling when the reference trajectory is not given as a closed form and it is shown that by applying the sliding mode control, the controller behavior of the mobile robot is robust against initial condition errors and external disturbances such as integration errors, noise in control signal, localization errors and etc [4].

The main problem with this control approach is that it again needs the precise dynamic model of the robot for the computed torque part. When we want to do the feedback linearization of the nonlinear dynamic model of the system, we need to know the exact robot dynamics. This disadvantage of this control law leads us to use an adaptive or intelligent controller with learning ability so that we can learn the robot dynamics online without knowing for the controller design. So many different methods of adaptive and intelligent control can be used to deal with this problem. Some of them will be explained in the next sections of this chapter.

2.4 FUZZY LOGIC TRAJECTORY TRACKING CONTROLLERS

A trajectory tracking control for a wheeled mobile robot using fuzzy logic control (FLC) is proposed in [14], [15], [16] and [17].

In [14] a control algorithm based on the errors in postures of mobile robot which feed FLC, which generates correction signals for the left and right motor speeds, is proposed. The control strategy is based on a feed forward velocity profile and the correcting signal in the velocity generated from the FLC according to the postures errors. Simulation and experimental results demonstrate the effectiveness of the proposed algorithm, which proved the good tracking results and showed the robustness of the algorithm against the uncertainties in the system model.

The proposed algorithm in [15] deals with development of a controller of fuzzy-genetics algorithm for 2-DOF Wheeled Mobile Robot (WMR). The global inputs to the WMR are a reference position, and a reference velocity, which are time variables. The global output

of WMR is a current posture the position of WMR is estimated by dead-reckoning algorithm. Dead-reckoning algorithm determines the present position of WMR in real time by adding up the increased position data to the previous one in sampling period. The tracking controller makes position error to be converged zero. In order to reduce position error, a compensation velocities on the track of trajectory is necessary. Therefore, a controller using fuzzy-genetic algorithm is proposed to give velocity compensation in this system. Input variables of two fuzzy logic controllers (FLCs) are position errors in every sampling time. The output values of FLCs are compensation velocities. Genetic algorithms (GAs) are implemented to adjust the output gain of fuzzy logic.

The controller in [16] presents a new tracking method for a mobile robot by combining predictive control and fuzzy logic control. Trajectory tracking of autonomous mobile robots usually has non-linear time-varying characteristics and is often perturbed by additive noise. To overcome the time delay caused by the slow response of the sensor, the algorithm uses predictive control, which predicts the position and orientation of the robot. In addition, fuzzy control is used to deal with the non-linear characteristics of the system.

The fuzzy logic controller proposed in [17] is based on a backstepping approach to ensure asymptotic stabilization of the robots position and orientation around the desired trajectory, taking into account the kinematics and dynamics of the vehicle. Mamadani inference system is used to construct the controller; with nine if-then rules and the centroid of area method as our diffuzification strategy where the input torques and velocities are considered as linguistic variables. Many researchers used only the kinematic model (steering system) to solve the tracking control problem, where the velocity, used as input control, is assumed to be supplied by the mobile robot whose dynamic of actuators is neglected. Real prototype have actuated wheels whose slip rate, rolling, inertia moment and mass distribution contribute to the forces exerted on the structure of the vehicle thus affecting the accuracy and full maneuverability of the robot. Motivated by this fact, the dynamic model of the robot is used in this wok to convert the steering system into the actual vehicle. The triangle and trapezoidal-shaped membership functions are used in this design, along with three fuzzy partitions and nine rules.

2.5 ADAPTIVE TRAJECTORY TRACKING CONTROLLERS

Adaptive control methods for trajectory tracking of a wheeled mobile robot are proposed in [18], [19], [20] and [21].

In [21] an adaptive control rules, at the dynamics level, for the nonholonomic mobile robots with unknown dynamic parameters in proposed. Adaptive controls are derived for mobile robots, using backstepping technique, for tracking of a reference trajectory and stabilization to a fixed posture. For the tracking problem, the controller guarantees the asymptotic convergence of the tracking error to zero. For stabilization, the problem is

converted to an equivalent tracking problem, using a time varying error feedback, before the tracking control is applied. The designed controller ensures the asymptotic zeroing of the stabilization error. The proposed control laws include a velocity/acceleration limiter that prevents the robot's wheels from slipping.

A novel simple adaptive tracking controller is presented in [20] based on the kinematics models. An artificial potential field is used to navigate the wheeled robot in the controller. Easy design, fast convergence, and adaptability to other nonholonomic mobile are obvious advantages. Stability of the rule is proved through the use of a Lyapunov function. A dual adaptive dynamic controller for trajectory tracking of nonholonomic wheeled mobile robots is presented in [18]. The controller is developed entirely in discrete-time and the robot's nonlinear dynamic functions are assumed to be unknown. A Gaussian radial basis function neural network is employed for function approximation, and its weights are estimated stochastically in real-time. In contrast to adaptive certainty equivalence controllers hitherto published for mobile robots, the proposed control law takes into consideration the estimates' uncertainty, thereby leading to improved tracking performance. The proposed method is verified by realistic simulations and Monte Carlo analysis.

2.6 NEURAL NETWORK BASED TRAJECTORY TRACKING CONTROLLERS

Nowadays, neural networks have been proved to be a promising approach to solve complex control problems. The neural network controllers are generally based on the function approximation property and learning ability of the neural network. The use of neural network controllers for trajectory tracking of mobile robots has been proposed in [11], [9] and [8]. The neural network controller proposed in [11] is based on the neural network function approximation property and can deal with unmodeled bounded disturbances and unstructured unmodeled dynamics of the mobile robot. The neural network is combined with the backstepping controller to learn the full dynamics of the mobile robot and convert the velocity output of the backstepping controller to a torque input for the actual vehicle. The advantage of having neural networks in this approach is that there is no need to know the dynamic model of the robot and the neural network will learn it online without a priori knowledge of the dynamics.

The neural network trajectory tracking controller proposed in [8] uses the learning property of the neural network to make an adaptive controller which adapts the backstepping controller gains. The proposed control approach has the properties to quickly drive the position error to zero and to indicate better smooth movement in the tracking performance process. This novel control approach integrated the backstepping controller with compound orthogonal networks and improves its performance by using the learning property of the neural network.

A wavelet neural network based controller for mobile robots is proposed in [23]. The work presents a predictive control scheme for mobile robots that possess complexity, nonlinearity and uncertainty. A multi layer back-propagation neural network in employed as a model for nonlinear dynamics of the robot. The neural network in constructed by the wavelet orthogonal decomposition to form a wavelet neural network that can overcome the problems caused by local minima of optimization. The wavelet network is also helpful to determine the number of the hidden nodes and the initial value of the weights.

CHAPTER 3

MOBILE ROBOT MODELING AND SIMULATION

Design, development, modification and control of a mechatronic system require an understanding and a suitable representation of a system; specifically, a "model" of the system is required. Any model is an idealization of the actual system. A mechatronic or robotic system may consist of several different types of components, and it is termed as a mixed system. One should use analogous procedures for modeling of such components. In this manner the component models can be conveniently integrated to obtain the overall model. Modeling of a differential drive mobile robot platform consists of kinematic and dynamic modeling in addition to the modeling of the system actuators. Kinematic modeling deals with the geometric relationships that govern the system and studies the mathematics of motion without considering the affecting forces. Dynamic modeling on the other hand is the study of the motion in which forces and energies are modeled and included. Actuator modeling is needed to find the relationship between the control signal and the mechanical system's input. Each part of this system's modeling will be explained separately throughout this chapter. After getting an accurate model, we can simulate the system using an appropriate computer package. The simulation process will be explained thoroughly in one section of this chapter. The first step for the mechanical modeling is to define appropriate coordinate systems for the platform which will be described in the next section.

3.1 COORDINATE SYSTEMS

The main function of the coordinate systems is to represent the position of the robot. The following two coordinate systems are used for mobile robot modeling and control purposes:

- Inertial Frame: $\{X_I, Y_I\}$ This is the fixed coordinate system in the plane of the robot.

- Robot Frame: $\{X_R, Y_R\}$ This is the coordinate system attached to the robot.

The above two coordinate systems are shown in Figure 3-1:

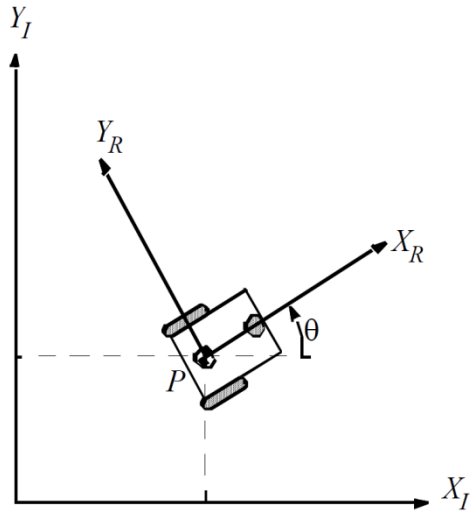

Figure 3-1: Inertial and Robot coordinate systems

Introducing these coordinate systems is helpful in the kinematic modeling of the robot which will be explained in the next section. The important issue that needs to be explained is the mapping between these two frames. The robot position in the inertial and robot frame can be defined as follows:

$$q_I = [x_I \quad y_I \quad \theta_I]^T \tag{3.1}$$

$$q_R = [x_R \quad y_R \quad \theta_R]^T \tag{3.2}$$

The mapping between these two frames is through the standard orthogonal rotation transformation:

$$\dot{q}_R = R(\theta)\dot{q}_I \tag{3.3}$$

$$R(\theta) = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

Using the above equations, we have the relation between the robot velocities in the local frame and the inertial frame which is very important in the robot kinematics.

3.2 KINEMATIC MODELING OF THE MOBILE ROBOT

The goal of the robot kinematic modeling is to find the robot speed in the inertial frame as a function of the wheels speeds and the geometric parameters of the robot

(configuration coordinates). In other words we want to establish the robot speed $\dot{q} = [\dot{x}\ \dot{y}\ \dot{\theta}]^T$ as a function of the wheel speeds $\dot{\varphi}_R$ and $\dot{\varphi}_L$ and the robot geometric parameters or we want to find the relationship between control parameters ($\dot{\varphi}_R$ and $\dot{\varphi}_L$) and the behavior of the system in the state space. The robot kinematics generally has two main analyses, one Forward kinematics and one Inverse kinematics:

- Forward kinematics:

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = f(\dot{\varphi}_R, \dot{\varphi}_L, gemometric\ parameters) \tag{3.5}$$

- Inverse kinematics:

$$\begin{bmatrix} \dot{\varphi}_R \\ \dot{\varphi}_L \end{bmatrix} = f(\dot{x}, \dot{y}, \dot{\theta}) \tag{3.6}$$

The Differential drive mobile robot forward kinematics will be discussed in the next section.

3.2.1 Forward kinematic model

Assume a differential drive mobile robot setup which has two wheels with the radius of R_a placed with a distance L from the robot center as shown in Figure 3-2:

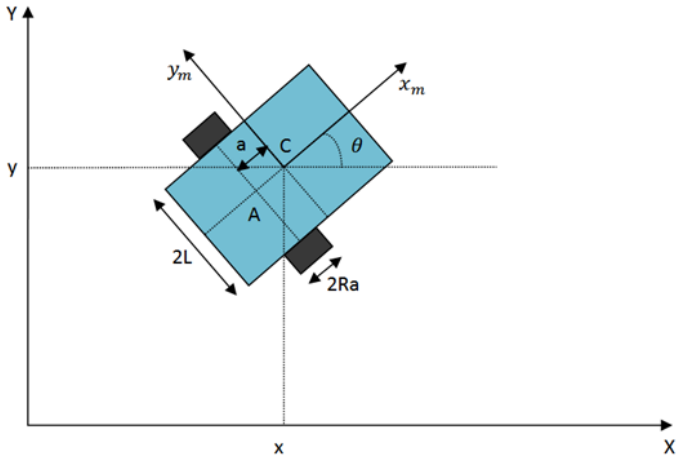

Figure 3-2: The differential drive mobile robot model

The following notations will be used in this thesis:

A: The intersection of the axis of symmetry with the driving wheels axis

C: The center of mass of the platform

a: The distance between the center of mass and driving wheels axis in *x*-direction

L: The distance between each driving wheel and the robot axis of symmetry in *y*-direction

R_a: The radius of each driving wheel

$\dot{\varphi}_R$: The rotational velocity of the right wheel

$\dot{\varphi}_L$: The rotational velocity of the left wheel

v: The translational velocity of the platform in the local frame

ω: The rotational velocity of the platform in the local and global frames

The forward kinematic problem can be described as the problem of finding the following function:

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = f(\dot{\varphi}_R, \dot{\varphi}_L, L, R_a, \theta) \tag{3.7}$$

The speed of each wheel in the robot frame is $R_a\dot{\varphi}$, therefore the translational speed in the robot frame is the average velocity:

$$v = R_a \frac{\dot{\varphi}_R + \dot{\varphi}_L}{2} \tag{3.8}$$

And the rotational velocity is:

$$\omega = \frac{R_a}{2L}(\dot{\varphi}_R - \dot{\varphi}_L) \tag{3.9}$$

Given that $\dot{q}_I = R(\theta)^{-1}\dot{q}_R$, the full model which is the robot velocity in the inertial frame is:

$$\dot{q}_I = R(\theta)^{-1} \frac{R_a}{2} \begin{bmatrix} \dot{\varphi}_R + \dot{\varphi}_L \\ 0 \\ \dfrac{\dot{\varphi}_R - \dot{\varphi}_L}{L} \end{bmatrix} \tag{3.10}$$

The inverse of the rotation matrix is:

$$R(\theta)^{-1} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

Therefore the robot velocity in the global or inertial frame is:

$$\dot{q}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \frac{R_a}{2} \begin{bmatrix} \dot{\varphi}_R + \dot{\varphi}_L \\ 0 \\ \frac{\dot{\varphi}_R - \dot{\varphi}_L}{L} \end{bmatrix} = \begin{bmatrix} R_a \frac{\dot{\varphi}_R + \dot{\varphi}_L}{2} \cos(\theta) \\ R_a \frac{\dot{\varphi}_R + \dot{\varphi}_L}{2} \sin(\theta) \\ \frac{R_a}{2L}(\dot{\varphi}_R - \dot{\varphi}_L) \end{bmatrix} \quad (3.12)$$

The above equation is the general forward kinematic equation for a differential drive mobile robot.

3.2.2 The kinematic constraints

The following assumptions about the wheel motion will cause the robot kinematic constraints:

- Movement on a horizontal plane
- Point contact between the wheels and ground
- Wheels are not deformable
- Pure rolling which means that we have the instantaneous center of zero velocity at the contact point of the wheels and ground.
- No slipping, skidding or sliding
- No friction for rotation around contact points
- Steering axes orthogonal to the surface
- Wheels are connected by a rigid frame (chassis)

Considering the above assumptions about the wheel motion, the robot will have a special kind of constraint called Nonholonomic constraint. A nonholonomic constraint is a constraint on the feasible velocities of a body. For the case of the differential drive mobile robot, it can simply mean that the robot can move in some directions (Forward and backward) but not others (Sideward) as is shown in Figure 3-3:

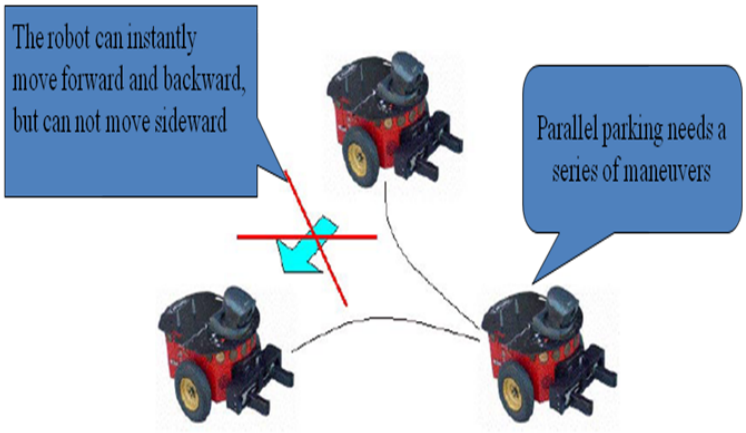

Figure 3-3: the nonholonomic constraint on the robot motion

Having this kind of constraints in a system will cause some challenges in the motion planning and control of such systems which will be explained in the control algorithm section of this thesis. The equations of the nonholonomic constraints for the differential drive mobile robot shown in figure 31 are as follows:

- No lateral slip constraint:

$$\dot{y}_c cos\theta - \dot{x}_c sin\theta - \dot{\theta} a = 0 \tag{3.13}$$

$\dot{y}_c$ and $\dot{x}_c$ are the robot velocity components in the inertial frame. This constraint means that the velocity of the robot center point will be in the direction of the axis of symmetry and the motion in the orthogonal plane will be zero.

- Pure rolling constraint:

$$\dot{x}_c cos\theta + \dot{y}_c sin\theta + L\dot{\theta} = R_a \dot{\varphi}_R \tag{3.14}$$

$$\dot{x}_c cos\theta + \dot{y}_c sin\theta - L\dot{\theta} = R_a \dot{\varphi}_L \tag{3.15}$$

This constraint shows that the driving wheels do not slip. The three constraints can be written in the following form:

$$A(q)\dot{q} = 0 \tag{3.16}$$

In which:

$$A(q) = \begin{bmatrix} -sin\theta & cos\theta & a & 0 & 0 \\ cos\theta & sin\theta & L & -R_a & 0 \\ cos\theta & sin\theta & -L & 0 & -R_a \end{bmatrix} \tag{3.17}$$

$$\dot{q} = \begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\theta} \\ \dot{\varphi}_R \\ \dot{\varphi}_L \end{bmatrix} \tag{3.18}$$

The above representation of the nonholonomic constraint is useful when we want to take the constraints in to account in the dynamic modeling. The system dynamic modeling which is the other important part of the robot modeling will be discussed in the next section.

3.3 DYNAMIC MODELING OF THE MOBILE ROBOT

Formulation of equations of motion, or dynamics, of a robot is essential in the analysis, design and control of a robot. Dynamic modeling in general is the study of the system's motion in which forces are modeled and it can include energies and the speeds associated with the motions. The main difference between dynamic and kinematic modeling is that

in kinematics we study the motion without considering the forces that affect the motion and we just deal with the geometric relationships that govern the system.

A mobile robot system having an *n*-dimensional configuration space *L* with generalized coordinates $(q_1, q_2, \dots, q_n)$ and subject to *m* constraints can be described by the following general dynamic equation:

$$M(q)\ddot{q} + V(q,\dot{q}) + F(\dot{q}) + G(q) + \tau_d = B(q)\tau - A^T(q)\lambda \tag{3.19}$$

Where:

$M(q)$ is the symmetric positive definite inertia matrix

$V(q,\dot{q})$ is the centripetal and coriolis matrix

$F(\dot{q})$ is the surface friction matrix

$G(q)$ is the gravitational vector

τ_d Denoted bounded unknown disturbances including unstructured unmodeled dynamics

$B(q)$ is the input transformation matrix

τ is the input vector

$A^T(q)$ is the matrix associated with the constraints

λ is the vector of the constraint forces

In the following two sections, the energy-based Lagrangian approach and the direct, Newton-Euler, vector mechanics approach are outlined for expressing the dynamics of this robot. The first approach is relatively more convenient to formulate and implement but the tradeoff is that physical insight and part of the useful information are lost in this process.

3.3.1 Lagrangian Dynamics approach

Analytical dynamics is an approach in dynamics which treats the system as a whole dealing with scalar quantities such as the kinetic and potential energies of the system. Lagrange (1736-1813) proposed an approach which provides a powerful and versatile method for the formulation of the equations of motion for any dynamical system. Lagrange's equations are differential equations in which one considers the energies of the system and the work done instantaneously in time. The derivation of the Lagrange equation for holonomic systems requires that the generalized coordinates be independent. For a nonholonomic system, however, there must be more number of generalized

coordinates than the number of degrees of freedom which is because of the constraints on the motion of the system.

If there are m nonholonomic constraint equations of the following form in the system:

$$\sum_{i=1}^{n} a_{ij} dq_i + a_{jt} dt = 0, \quad j = 1,2, \dots m \tag{3.20}$$

Where a_{ij} are functions of the generalized coordinates, then the general form of the Lagrange equation is as follows:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = \sum_{j=1}^{n} \lambda_j a_{ji} + Q_i \quad i = 1,2, \dots n \tag{3.21}$$

Where:

$q_1, q_2, \dots\ q_n$ are the generalized coordinates

$L = T - V$ is the Lagrangian which is the difference between the systems kinetic and potential energy.

λ_j is the Lagrangian multiplier which relates the constraints to the constraint forces

Q_i Noncoservative forces in the system

In addition to these *n* equations, we have *m* equations of the constraints to solve for the *(m+n)* unknowns, i.e. q's and λ's.

The first step in finding the dynamic equation is to find the systems kinetic and potential energy. The potential energy of the system is zero because the motion is restricted to the ground. The kinetic energy function of the robot can be derived according to the velocities shown in Figure 3-4:

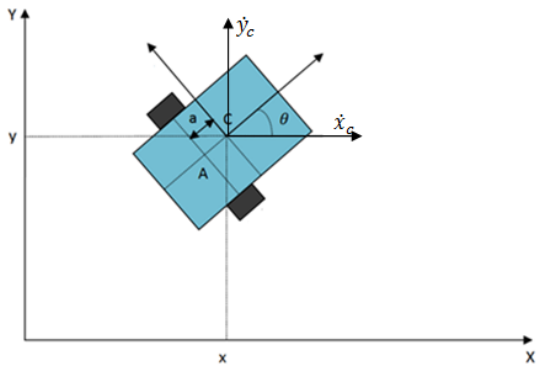

Figure 3-4: the velocities of the robot center

To find the velocity of the point A and C:

$$x_c = x_A + a\cos\theta \tag{3.22}$$

$$y_c = y_A + a\sin\theta \tag{3.23}$$

Therefore:

$$\dot{x}_c = \dot{x}_A - a\dot{\theta}\sin\theta \tag{3.24}$$

$$\dot{y}_c = \dot{y}_A + a\dot{\theta}\cos\theta \tag{3.25}$$

The velocity of the robot center of rotation *A* is:

$$v_A = \dot{x}_c\vec{\imath} + \dot{y}_c\vec{\jmath} + a\dot{\theta}\sin\theta\vec{\imath} - a\dot{\theta}\cos\theta\vec{\jmath} \tag{3.26}$$

$$v_A = \left(\dot{x}_c + a\dot{\theta}\sin\theta\right)\vec{\imath} + \left(\dot{y}_c - a\dot{\theta}\cos\theta\right)\vec{\jmath} \tag{3.27}$$

The kinetic energy is:

$$T = \frac{1}{2}m{v_A}^2 + \frac{1}{2}I_A\dot{\theta}^2 \tag{3.28}$$

$$\begin{aligned} T = \frac{1}{2}m\left[\left(\dot{x}_c + a\dot{\theta}\sin\theta\right)^2 + \left(\dot{y}_c - a\dot{\theta}\cos\theta\right)^2\right] \\ = \frac{1}{2}m\left[{\dot{x}_c}^2 + 2\dot{x}_c a\dot{\theta}\sin\theta + a^2\dot{\theta}^2\sin\theta^2 + {\dot{y}_c}^2 - 2\dot{y}_c a\dot{\theta}\cos\theta\right. \\ \left. + a^2\dot{\theta}^2\cos\theta^2\right] + \frac{1}{2}I_A\dot{\theta}^2 \end{aligned} \tag{3.29}$$

$$T = \frac{1}{2}m{\dot{x}_c}^2 + \frac{1}{2}m{\dot{y}_c}^2 + m\dot{x}_c a\dot{\theta}\sin\theta - m\dot{y}_c a\dot{\theta}\cos\theta + \frac{1}{2}ma^2\dot{\theta}^2 + \frac{1}{2}I_A\dot{\theta}^2 \tag{3.30}$$

From the parallel axis theorem we can say that:

$$I_C + ma^2 = I_A \tag{3.31}$$

Therefore the kinetic energy of the system will be:

$$T = \frac{1}{2}m\dot{x_c}^2 + \frac{1}{2}m\dot{y_c}^2 + \frac{1}{2}I_C\dot{\theta}^2 + ma^2\dot{\theta}^2 + m\dot{x}_c a\dot{\theta}sin\theta - m\dot{y}_c a\dot{\theta}cos\theta \quad (3.32)$$

From the above kinetic energy equation and knowing the potential energy is zero, the Lagrangian will become:

$$L = \frac{1}{2}m\dot{x_c}^2 + \frac{1}{2}m\dot{y_c}^2 + \frac{1}{2}I_C\dot{\theta}^2 + ma^2\dot{\theta}^2 + m\dot{x}_c a\dot{\theta}sin\theta - m\dot{y}_c a\dot{\theta}cos\theta \quad (3.33)$$

The generalized coordinates to use in the Lagrange formulation are as follows:

$$q = [x_c, y_c, \theta] \quad (3.34)$$

$$\dot{q} = [\dot{x}_c, \dot{y}_c, \dot{\theta}] \quad (3.35)$$

The step by step approach to find the dynamic equations using the above generalized coordinates and Lagrangian is as follows:

$$\frac{\partial L}{\partial \dot{x}_c} = m\dot{x}_c + ma\dot{\theta}sin\theta \quad (3.36)$$

$$\frac{\partial L}{\partial \dot{y}_c} = m\dot{y}_c - ma\dot{\theta}cos\theta \quad (3.37)$$

$$\frac{\partial L}{\partial \dot{\theta}} = I_C\dot{\theta} + 2ma^2\dot{\theta} + m\dot{x}_c asin\theta - m\dot{y}_c acos\theta \quad (3.38)$$

The time derivatives of the above three equations are as follows:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}_c}\right) = m\ddot{x}_c + ma\ddot{\theta}sin\theta + ma\dot{\theta}^2cos\theta \quad (3.39)$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{y}_c}\right) = m\ddot{y}_c - ma\ddot{\theta}cos\theta + ma\dot{\theta}^2sin\theta \quad (3.40)$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) = I_C\ddot{\theta} + 2ma^2\ddot{\theta} + ma\ddot{x}_c sin\theta + ma\dot{x}_c\dot{\theta}cos\theta - ma\ddot{y}_c cos\theta + m\dot{y}_a\dot{\theta}sin\theta \tag{3.41}$$

The rest of the derivatives to complete the Lagrange equations are as follows:

$$\frac{\partial L}{\partial x_c} = 0 \tag{3.42}$$

$$\frac{\partial L}{\partial y_c} = 0 \tag{3.43}$$

$$\frac{\partial L}{\partial \theta} = m\dot{x}_c a\dot{\theta}cos\theta + m\dot{y}_c a\dot{\theta}sin\theta \tag{3.44}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) = I_C\ddot{\theta} + 2ma^2\ddot{\theta} + ma\ddot{x}_c sin\theta + ma\dot{x}_c\dot{\theta}cos\theta - ma\ddot{y}_c cos\theta + m\dot{y}_a\dot{\theta}sin\theta \tag{3.45}$$

Substituting the above term in the Lagrange equation, we have:

$$m\ddot{x}_c + ma\ddot{\theta}sin\theta + ma\dot{\theta}^2 cos\theta = F_x + C_1 \tag{3.46}$$

$$m\ddot{y}_c - ma\ddot{\theta}cos\theta + ma\dot{\theta}^2 sin\theta = F_y + C_2 \tag{3.47}$$

$$(I_C + 2ma^2)\ddot{\theta} + ma\ddot{x}_c sin\theta + ma\dot{x}_c\dot{\theta}cos\theta - ma\ddot{y}_c cos\theta + m\dot{y}_c a\dot{\theta}sin\theta - m\dot{x}_c a\dot{\theta}cos\theta - m\dot{y}_c a\dot{\theta}sin\theta = \tau + C_3 \tag{3.48}$$

Simplifying the above equations we have:

$$m\ddot{x}_c + ma\ddot{\theta}sin\theta + ma\dot{\theta}^2cos\theta = F_x + C_1 \tag{3.49}$$

$$m\ddot{y}_c - ma\ddot{\theta}cos\theta + ma\dot{\theta}^2sin\theta = F_y + C_2 \tag{3.50}$$

$$(I_C + 2ma^2)\ddot{\theta} + ma\ddot{x}_c sin\theta - ma\ddot{y}_c cos\theta = \tau + C_3 \tag{3.51}$$

Where:

F_x is the actuator force in the x-direction

F_y is the actuator force in the y-direction

τ is the actuator rotational torque on the robot

C_X, C_Y, C_θ are the constraint forces in the x ,y and θ directions.

Representing the above equations in the matrix form we have:

$$\begin{bmatrix} m & 0 & masin\theta \\ 0 & m & -macos\theta \\ masin\theta & -macos\theta & I_C + 2ma^2 \end{bmatrix} \ddot{q} + \begin{bmatrix} ma\dot{\theta}^2cos\theta \\ ma\dot{\theta}^2sin\theta \\ 0 \end{bmatrix} = \begin{bmatrix} F_x \\ F_y \\ \tau \end{bmatrix} + \begin{bmatrix} C_X \\ C_Y \\ C_\theta \end{bmatrix} \tag{3.52}$$

We can relate the forces in the x, y and θ directions to the actuator torques on each wheel according to the free body diagram of the robot shown in Figure 3-5:

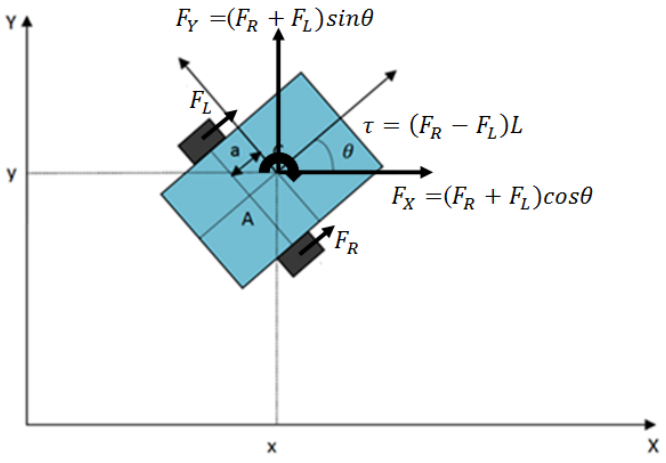

Figure 3-5: The robots free body diagram

$$F_R = \frac{\tau_R}{R_a} \tag{3.53}$$

$$F_L = \frac{\tau_L}{R_a} \tag{3.54}$$

$$F_X = \left(\frac{\tau_R}{R_a} + \frac{\tau_L}{R_a}\right) cos\theta = \frac{cos\theta}{R_a}(\tau_R + \tau_L) \tag{3.55}$$

$$F_y = \left(\frac{\tau_R}{R_a} + \frac{\tau_L}{R_a}\right) sin\theta = \frac{sin\theta}{R_a}(\tau_R + \tau_L) \tag{3.56}$$

$$\tau = (F_R - F_L)L = \frac{L}{R_a}(\tau_R - \tau_L) \tag{3.57}$$

According to the above equations, we can write the input force matrix in the system's dynamic equation as follows:

$$\begin{bmatrix} F_x \\ F_y \\ \tau \end{bmatrix} = \begin{bmatrix} \frac{cos\theta}{R_a}(\tau_R + \tau_L) \\ \frac{sin\theta}{R_a}(\tau_R + \tau_L) \\ \frac{L}{R_a}(\tau_R - \tau_L) \end{bmatrix} = \frac{1}{R_a}\begin{bmatrix} cos\theta & cos\theta \\ sin\theta & sin\theta \\ L & -L \end{bmatrix}\begin{bmatrix} \tau_R \\ \tau_L \end{bmatrix} \tag{3.58}$$

As it is mentioned in the previous section the nonholonomic constraints of this system are:

$$\dot{y}_c cos\theta - \dot{x}_c sin\theta - \dot{\theta} a = 0 \tag{3.59}$$

$$\dot{x}_c cos\theta + \dot{y}_c sin\theta + L\dot{\theta} = R_a \dot{\varphi}_R \tag{3.60}$$

$$\dot{x}_c cos\theta + \dot{y}_c sin\theta - L\dot{\theta} = R_a \dot{\varphi}_L \tag{3.61}$$

According to these constraint equations, the constraint forces will become:

$$C_X = m(\dot{x}_c cos\theta + \dot{y}_c sin\theta)\dot{\theta} sin\theta \tag{3.62}$$

$$C_Y = -m(\dot{x}_c cos\theta + \dot{y}_c sin\theta)\dot{\theta} cos\theta \tag{3.63}$$

$$C_\theta = ma(\dot{x}_c cos\theta + \dot{y}_c sin\theta)\dot{\theta} \tag{3.65}$$

The matrix representation of the above constraint forces is as follows:

$$\begin{bmatrix} C_X \\ C_Y \\ C_\theta \end{bmatrix} = \begin{bmatrix} m(\dot{x}_c cos\theta + \dot{y}_c sin\theta)\dot{\theta} sin\theta \\ -m(\dot{x}_c cos\theta + \dot{y}_c sin\theta)\dot{\theta} cos\theta \\ ma(\dot{x}_c cos\theta + \dot{y}_c sin\theta)\dot{\theta} \end{bmatrix} = A^T(q)\lambda \tag{3.66}$$

$$A^T(q) = \begin{bmatrix} -sin\theta \\ cos\theta \\ -a \end{bmatrix} \tag{3.67}$$

$$\lambda = -m(\dot{x}_c cos\theta + \dot{y}_c sin\theta)\dot{\theta} \tag{3.68}$$

According to the above derivations, the complete form of the differential drive mobile robot dynamic equation is as follows:

$$\begin{bmatrix} m & 0 & masin\theta \\ 0 & m & -macos\theta \\ masin\theta & -macos\theta & I_C + 2ma^2 \end{bmatrix} \ddot{q} + \begin{bmatrix} ma\dot{\theta}^2 cos\theta \\ ma\dot{\theta}^2 sin\theta \\ 0 \end{bmatrix} = \frac{1}{R_a} \begin{bmatrix} cos\theta & cos\theta \\ sin\theta & sin\theta \\ L & -L \end{bmatrix} \begin{bmatrix} \tau_R \\ \tau_L \end{bmatrix} + \begin{bmatrix} m(\dot{x}_c cos\theta + \dot{y}_c sin\theta)\dot{\theta} sin\theta \\ -m(\dot{x}_c cos\theta + \dot{y}_c sin\theta)\dot{\theta} cos\theta \\ ma(\dot{x}_c cos\theta + \dot{y}_c sin\theta)\dot{\theta} \end{bmatrix} \tag{3.69}$$

Comparing the above equation and the general form of the robot dynamic equation (3.19), we have the following parameters for the differential drive mobile robot system:

$$M(q) = \begin{bmatrix} m & 0 & masin\theta \\ 0 & m & -macos\theta \\ masin\theta & -macos\theta & I_C + 2ma^2 \end{bmatrix}$$

$$V(q, \dot{q}) = \begin{bmatrix} ma\dot{\theta}^2 cos\theta \\ ma\dot{\theta}^2 sin\theta \\ 0 \end{bmatrix}$$

$F(\dot{q}) = 0$ considered to be zero in this derivation

$G(q) = 0$ motion is constrained to the ground

$\tau_d = 0$ considered to be zero in this derivation

$$\tau = \begin{bmatrix} \tau_R \\ \tau_L \end{bmatrix}$$

$$A^T(q) = \begin{bmatrix} -sin\theta \\ cos\theta \\ -a \end{bmatrix}$$

$$\lambda = -m(\dot{x}_c cos\theta + \dot{y}_c sin\theta)\dot{\theta}$$

$$B(q) = \frac{1}{R_a}\begin{bmatrix} cos\theta & cos\theta \\ sin\theta & sin\theta \\ L & -L \end{bmatrix}$$

The above system can be transformed into a more proper representation for control and simulation purposes. In this transformation, we are trying to find a way to eliminate the constraint term from the equation. The following two matrices are defined to do this transformation:

$$v_a(t) = \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \dot{X}_R \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \tag{3.70}$$

$$S(q) = \begin{bmatrix} cos\theta & -asin\theta \\ sin\theta & acos\theta \\ 0 & 1 \end{bmatrix} \tag{3.71}$$

By taking a look at the forward kinematic equation, one can realize that the *S (q)* matrix is the modified forward kinematic matrix which has two velocity terms related to the distance between the robot centroid and wheel axis. Therefore, we can write the following equation for the system:

$$\dot{q} = \begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\theta} \end{bmatrix} = S(q)v_a(t) = \begin{bmatrix} cos\theta & -asin\theta \\ sin\theta & acos\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.72}$$

It can easily be proved that the *S (q)* matrix has the following relation with *A(q)* matrix:

$$S^T(q)A^T(q) = 0 \tag{3.73}$$

The above equation is useful when we want to eliminate the constraint term from the main dynamic equation as you will see in the next step. Differentiating equation (3.72), we have:

$$\ddot{q} = \dot{S}(q)v_a(t) + S(q)\dot{v_a}(t) \tag{3.74}$$

Substituting the above equation in (3.69) will result the following equation:

$$\begin{aligned} M(q)\left[\dot{S}(q)v_a(t) + S(q)\dot{v_a}(t)\right] + V_m(q,\dot{q})S(q)v_a(t) + F(\dot{q}) + G(q) + \tau_d \\ = B(q)\tau - A^T(q)\lambda \end{aligned} \tag{3.75}$$

$$\begin{aligned} M(q)\dot{S}(q)v_a(t) + M(q)S(q)\dot{v_a}(t) + V_m(q,\dot{q})S(q)v_a(t) + F(\dot{q}) + G(q) + \tau_d \\ = B(q)\tau - A^T(q)\lambda \end{aligned} \tag{3.76}$$

Where:

$$V_m(q,\dot{q}) = \begin{bmatrix} 0 & 0 & ma\dot{\theta}cos\theta \\ 0 & 0 & ma\dot{\theta}sin\theta \\ 0 & 0 & 0 \end{bmatrix} \tag{3.77}$$

The next step to eliminate the constraint matrix $A^T(q)\lambda$ is to multiply equation (3.76) by $S^T(q)$ as follows:

$$\begin{aligned} [S^T(q)M(q)S(q)]\dot{v_a}(t) + [S^T(q)\, M(q)\dot{S}(q) + S^T(q){V_m}^T(q,\dot{q})S]v_a(t) \\ + S^T(q)\, F(\dot{q}) + S^T(q)\, G(q) + S^T(q)\, \tau_d \\ = S^T(q)\, B(q)\tau - S^T(q)\, A^T(q)\lambda \end{aligned} \tag{3.78}$$

As it can be seen from the above equation, we have $S^T(q)\, A^T(q)$ which is zero according to equation (3.73). Therefore the constraint term is eliminated and the new dynamic equation is:

$$\begin{aligned} [S^T(q)M(q)S(q)]\dot{v_a}(t) + [S^T(q)M(q)\dot{S}(q) + S^T(q){V_m}^T(q,\dot{q})S]v_a(t) \\ + S^T(q)\, F(\dot{q}) + S^T(q)\, G(q) + S^T(q)\, \tau_d = S^T(q)\, B(q)\tau \end{aligned} \tag{3.79}$$

By the following appropriate definitions we can rewrite the above equation as follows:

$$\bar{M}(q)\dot{v_a}(t) + \bar{V}_m(q,\dot{q})v_a(t) + \bar{F}(\dot{q}) + \bar{G}(q) + \bar{\tau}_d = \bar{B}(q)\tau \tag{3.80}$$

$$S^T(q)M(q)S(q) = \bar{M}(q) \tag{3.81}$$

$$S^T(q)M(q)\dot{S}(q) + S^T(q){V_m}^T(q,\dot{q})S = \bar{V}_m(q,\dot{q}) \tag{3.82}$$

$$V_m(q,\dot{q}) = \begin{bmatrix} 0 & 0 & ma\dot{\theta}cos\theta \\ 0 & 0 & ma\dot{\theta}sin\theta \\ 0 & 0 & 0 \end{bmatrix} \tag{3.83}$$

$$S^T(q)F(\dot{q}) = \bar{F}(\dot{q}) = 0 \tag{3.84}$$

$$S^T(q)G(q) = \bar{G}(q) = 0 \tag{3.85}$$

$$S^T(q)\tau_d = \bar{\tau}_d \tag{3.86}$$

$$S^T(q)B(q) = \bar{B}(q) \tag{3.87}$$

$$S^T(q)A^T(q)\lambda = 0 \tag{3.88}$$

The new mass matrix $\bar{M}(q)$ will become:

$$\begin{aligned}\bar{M}(q) &= S^T(q)M(q)S(q)\\ &= \begin{bmatrix} cos\theta & sin\theta & 0\\ -asin\theta & acos\theta & 1\end{bmatrix}\begin{bmatrix} m & 0 & masin\theta\\ 0 & m & -macos\theta\\ masin\theta & -macos\theta & I_C + 2ma^2\end{bmatrix}\begin{bmatrix} cos\theta & -asin\theta\\ sin\theta & acos\theta\\ 0 & 1\end{bmatrix}\\ &= \begin{bmatrix} mcos\theta & msin\theta & 0\\ 0 & 0 & I_C + ma^2\end{bmatrix}\begin{bmatrix} cos\theta & -asin\theta\\ sin\theta & acos\theta\\ 0 & 1\end{bmatrix} = \begin{bmatrix} m & 0\\ 0 & I_C + ma^2\end{bmatrix}\end{aligned} \quad (3.89)$$

Equation (3.80) is the equation which is used for the control and simulation analysis of the robot. The above approach for the dynamic modeling of the robot is used in references [11] and [12].

3.3.2 Newton-Euler Dynamics approach

The first and one of the most important steps in Newtonian dynamic modeling is to draw the free body diagram of the system and analyzing the forces on it. The free body diagram of the differential drive mobile robot is shown in Figure 3-6:

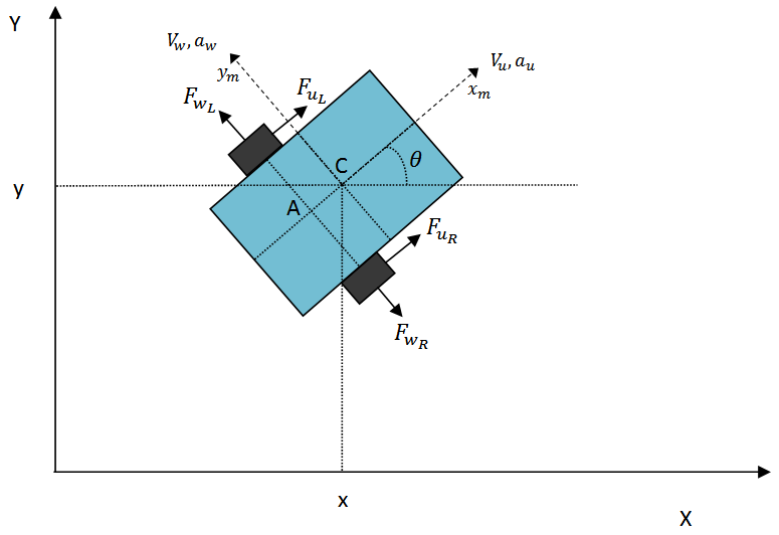

Figure 3-6: the robot's free body diagram for Newtonian dynamic modeling

The following notations are introduced in this figure and will be used for the Newtonian dynamic modeling:

(v_u, v_w): Represents the velocity of the vehicle in the local frame. v_u Is the longitudinal velocity and v_w is the lateral velocity.

(a_u, a_w): represent the acceleration of the vehicle's center of mass

(F_{u_l}, F_{u_r}): are the longitudinal forces exerted on the vehicle by the left and right wheels.

(F_{w_l} , F_{w_r}): are the lateral forces exerted on the vehicle by the left and right wheels

As it can be seen from the above free body diagram, the only forces acting on the robot are actuator forces acting on the robot wheels. We start the derivation by representing the robot position using polar coordinates. Assuming that the robot is a rigid body, its position can be represented using its angle and radius:

$$\hat{r} = re^{i\theta} \tag{3.90}$$

Differentiating the above position vector will give us the velocity and acceleration of the robot:

$$\dot{\hat{r}} = \dot{r}e^{i\theta} + r\dot{\theta}ie^{i\theta} \tag{3.91}$$

$$\ddot{\hat{r}} = \ddot{r}e^{i\theta} + \dot{r}\dot{\theta}ie^{i\theta} - r\dot{\theta}^2e^{i\theta} + \dot{r}\dot{\theta}ie^{i\theta} + r\ddot{\theta}ie^{i\theta} \tag{3.92}$$

Simplifying and writing the velocity and acceleration terms in radial and tangential terms, we have:

$$\dot{\hat{r}} = [\dot{r}]e^{i\theta} + [r\dot{\theta}]e^{i\theta+\frac{\pi}{2}} \tag{3.93}$$

$$\ddot{\hat{r}} = \left[\ddot{r} - r\dot{\theta}^2\right]e^{i\theta} + [2\dot{r}\dot{\theta} + r\ddot{\theta}]e^{i\theta+\frac{\pi}{2}} \tag{3.94}$$

The radial and tangential velocity and acceleration terms are defined as follows:

$$v_u = \dot{r} \tag{3.95}$$

$$v_w = r\dot{\theta} \tag{3.96}$$

$$a_u = \ddot{r} - r\dot{\theta}^2 \tag{3.97}$$

$$a_w = 2\dot{r}\dot{\theta} + r\ddot{\theta} \tag{3.98}$$

From the above four equations, we can write the following relations between the radial and tangential velocity and acceleration of the robot:

$$a_u = \dot{v}_u - v_w\dot{\theta} \tag{3.99}$$

$$a_w = \dot{v}_w + v_u\dot{\theta} \tag{3.100}$$

The next step is to write the Newton's second law in radial (u) and tangential (w) directions and finding the relation between the forces and accelerations:

$$\sum F_u = ma_u$$

$$ma_u = F_{u_l} + F_{u_r} \tag{3.101}$$

$$\sum F_w = ma_w$$

$$ma_w = F_{w_l} + F_{w_r} \tag{3.102}$$

Substituting the acceleration terms from equations (3.99) and (3.100) in equations (3.101) and (3.102) we have:

$$\dot{v}_u = v_w\dot{\theta} + \frac{F_{u_l} + F_{u_r}}{m} \tag{3.103}$$

$$\dot{v}_w = -v_u\dot{\theta} + \frac{F_{w_l} + F_{w_r}}{m} \tag{3.104}$$

The above two equations show the acceleration of the robot in terms of the actuating forces and the velocity terms. The Newton's second law in rotation about the center of mass:

$$\sum M_C = I_C\ddot{\theta}$$

$$\ddot{\theta} = -\frac{(F_{w_l} + F_{w_r})a}{I_C} + \frac{(F_{u_r} - F_{u_l})L}{I_C} \tag{3.105}$$

Equations (3.103), (3.104) and (3.105) are the main dynamic equations of the differential drive mobile robot derived from the Newtonian dynamic approach.

The absence of slipping in the longitudinal (pure rolling) and lateral (no sliding) directions creates independence between the longitudinal, lateral and angular velocities and simplifies the dynamic equations.

In the absence of lateral slippage (no sliding constraint), the lateral velocity of the midpoint if the drive wheels is zero. Therefore, the lateral velocity of the center of mass will be:

$$v_w = a\dot{\theta} \tag{3.106}$$

And the lateral acceleration of the center of mass will be:

$$\dot{v}_w = a\ddot{\theta} \tag{3.107}$$

Combining equations (3.104) and (3.99) we have:

$$a\ddot{\theta} = -v_u\dot{\theta} + \frac{F_{w_l} + F_{w_r}}{m} \tag{3.108}$$

$$m(a\ddot{\theta} + v_u\dot{\theta}) = F_{w_l} + F_{w_r} \tag{3.109}$$

Combining equations (3.108) and (3.100) and solving for $\ddot{\theta}$ we have:

$$\ddot{\theta} = \frac{(F_{u_r} - F_{u_l})L}{ma^2 + I_C} - \frac{mav_u\dot{\theta}}{ma^2 + I_C} \tag{3.110}$$

Combining equations (3.105) and (3.98) we have:

$$\dot{v}_u = a\dot{\theta}^2 + \frac{F_{u_r} + F_{u_l}}{m} \tag{3.111}$$

The above two equations are the dynamic equations of the robot considering the nonholonomic constraints. The above two equations can easily be transformed to the matrix form using the same notations and matrices in the Lagrangian approach:

$$(ma^2 + I_C)\ddot{\theta} + mav\dot{\theta} = \frac{(\tau_R - \tau_L)L}{R_a} \tag{3.112}$$

$$m\dot{v} - ma\dot{\theta}^2 = \frac{(\tau_R + \tau_L)}{R_a} \tag{3.113}$$

The matrix form of the above two equation is shown in the following equation:

$$\begin{bmatrix} m & 0 \\ 0 & ma^2 + I_C \end{bmatrix}\begin{bmatrix} \dot{v} \\ \ddot{\theta} \end{bmatrix} + \begin{bmatrix} 0 & -ma\dot{\theta} \\ ma\dot{\theta} & 0 \end{bmatrix}\begin{bmatrix} v \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{1}{R_a} & \frac{1}{R_a} \\ \frac{L}{R_a} & \frac{-L}{R_a} \end{bmatrix}\begin{bmatrix} \tau_R \\ \tau_L \end{bmatrix} \tag{3.114}$$

As it can be seen from the above equation, both methods will reach the same dynamic equation for the mobile robot. Using equations (3.109) and (3.106) and the forward kinematics equations (3.8) and (3.9), we can easily write the general dynamic equations in terms of the actuator torques, wheels rotational velocities and geometric parameters. Differentiating the forward kinematic equations (3.8) and (3.9) we have:

$$\ddot{X}_R = \dot{v}_u = \frac{\ddot{\varphi}_R + \ddot{\varphi}_L}{2} R_a \tag{3.115}$$

$$\ddot{\theta} = \frac{\ddot{\varphi}_R - \ddot{\varphi}_L}{2L} R_a \tag{3.116}$$

Inserting equations (3.108) and (3.8) in equation (3.109) we have:

$$\frac{\ddot{\varphi}_R - \ddot{\varphi}_L}{2L} R_a = \frac{(F_{u_r} - F_{u_l})L}{ma^2 + I_A} - \frac{ma}{ma^2 + I_A}\left(\frac{\dot{\varphi}_R + \dot{\varphi}_L}{2} R_a\right)\left(\frac{\dot{\varphi}_R - \dot{\varphi}_L}{2L} R_a\right) \tag{3.117}$$

Inserting equations (3.107) and (3.9) in equation (3.106) we have:

$$\frac{\ddot{\varphi}_R + \ddot{\varphi}_L}{2} R_a = a(\frac{\dot{\varphi}_R - \dot{\varphi}_L}{2L} R_a)^2 + \frac{F_{u_r} + F_{u_l}}{m} \tag{3.118}$$

From (3.109) and (3.110) we have:

$$F_{u_r} - F_{u_l} = \frac{R_a(ma^2 + I_A)}{2L^2}(\ddot{\varphi}_R - \ddot{\varphi}_L) + \frac{ma}{L}\left(\frac{\dot{\varphi}_R + \dot{\varphi}_L}{2} R_a\right)\left(\frac{\dot{\varphi}_R - \dot{\varphi}_L}{2L} R_a\right) \tag{3.119}$$

$$F_{u_r} + F_{u_l} = \frac{mR_a}{2}(\ddot{\varphi}_R + \ddot{\varphi}_L) - am(\frac{\dot{\varphi}_R - \dot{\varphi}_L}{2L} R_a)^2 \tag{3.120}$$

Adding and simplifying the above two equation, we have:

$$F_{u_r} = \left(\frac{R_a(ma^2 + I_A)}{4L^2} + \frac{mR_a}{4}\right)\ddot{\varphi}_R + \left(-\frac{R_a(ma^2 + I_A)}{4L^2} + \frac{mR_a}{4}\right)\ddot{\varphi}_L - \left(\frac{maR_a{}^2}{4L^2}\right)\dot{\varphi}_L{}^2 + \left(\frac{maR_a{}^2}{4L^2}\right)\dot{\varphi}_R\dot{\varphi}_L \tag{3.121}$$

$$F_{u_l} = \left(\frac{R_a(ma^2 + I_A)}{4L^2} + \frac{mR_a}{4}\right)\ddot{\varphi}_L + \left(-\frac{R_a(ma^2 + I_A)}{4L^2} + \frac{mR_a}{4}\right)\ddot{\varphi}_R - \left(\frac{maR_a{}^2}{4L^2}\right)\dot{\varphi}_R{}^2 + \left(\frac{maR_a{}^2}{4L^2}\right)\dot{\varphi}_R\dot{\varphi}_L \tag{3.122}$$

Assuming that the system inputs are the DC motor torques, the system dynamic equations will be:

$$\tau_R = \left(\frac{R_a{}^2(ma^2 + I_A)}{4L^2} + \frac{mR_a{}^2}{4}\right)\ddot{\varphi}_R + \left(-\frac{R_a{}^2(ma^2 + I_A)}{4L^2} + \frac{mR_a{}^2}{4}\right)\ddot{\varphi}_L - \left(\frac{maR_a{}^3}{4L^2}\right)\dot{\varphi}_L{}^2 + \left(\frac{maR_a{}^3}{4L^2}\right)\dot{\varphi}_R\dot{\varphi}_L \tag{3.123}$$

$$\tau_L = \left(\frac{R_a{}^2(ma^2 + I_A)}{4L^2} + \frac{mR_a{}^2}{4}\right)\ddot{\varphi}_L + \left(-\frac{R_a{}^2(ma^2 + I_A)}{4L^2} + \frac{mR_a{}^2}{4}\right)\ddot{\varphi}_R - \left(\frac{maR_a{}^3}{4L^2}\right)\dot{\varphi}_R{}^2 + \left(\frac{maR_a{}^3}{4L^2}\right)\dot{\varphi}_R\dot{\varphi}_L \tag{3.124}$$

The following parameters were defined to simplify the rest of the calculations:

$$A = \frac{R_a{}^2(ma^2 + I_A)}{4L^2} + \frac{mR_a{}^2}{4} \tag{3.125}$$

$$B = -\frac{R_a{}^2(ma^2 + I_A)}{4L^2} + \frac{mR_a{}^2}{4} \tag{3.126}$$

$$C = \frac{maR_a{}^3}{4L^2} \tag{3.127}$$

Therefore, the dynamic equations can be written as follows:

$$\tau_R = A\ddot{\varphi}_R + B\ddot{\varphi}_L - C\dot{\varphi}_L{}^2 + C\dot{\varphi}_R\dot{\varphi}_L \tag{3.128}$$

$$\tau_L = A\ddot{\varphi}_L + B\ddot{\varphi}_R - C\dot{\varphi}_R{}^2 + C\dot{\varphi}_R\dot{\varphi}_L \tag{3.129}$$

Equations (3.128) and (3.129) are the dynamic equations relating the actuator torques, wheels rotational velocities and rotational acceleration. The same results we come up if we write the dynamic equations derived from Lagrangian approach in terms of wheels rotational velocities and accelerations. For simulation and control purposes, one can use either equations (3.128) and (3.129) from the Newtonian approach or equation (3.80) from the Lagrangian approach. I used equation (3.114) for simulating the system which is described in the next section. The same dynamic modeling approach is used in references [24] and [2].

3.4 ACTUATOR MODELING

Actuator is a device that mechanically drives a robotic system. There are many classifications of actuators. Those that directly operate a process (load, plant) are termed process actuators. Joint motors in robotic manipulators are good examples of process actuators. Actuators that automatically use response error signals from a process in feedback to correct the operation of the process are termed servo actuators. In particular motors that use position, speed, and perhaps load torque measurements and armature current or field current in feedback, to drive a load according to a specified motion, are termed servo actuators. Clearly, the dc motors which are used for locomotion of the differential drive mobile robot system are considered to be servo actuators. If the system characteristics and loading conditions are very accurately known and if the system is stable, it is possible to schedule the input signal to a motor (the armature voltage or field voltage) so as to obtain a desired response (motion trajectory or torque). Parameter variations, model uncertainties and external disturbances can produce errors which can be reduced by using feedback control methods

In an armature-controlled dc motor which is the case for our system, the armature voltage v_a is used as the control input while keeping the conditions in the field circuit constant. In particular the field current i_f is assumed to be constant. Consequently we have the following equations for the dc motor model:

$$T_m = k_m i_a \tag{3.117}$$

$$v_b = k_b \omega_m \tag{3.118}$$

Where:

ω_m is the angular speed of the motor

i_a is the armature current

The parameters k_m and k_b are termed the torque constant and the back e.m.f. constant respectively. Note that with consistent units, $k_m = k_b$ in the case of ideal electrical to mechanical energy conversion at the motor rotor.

The equation for the armature rotor circuit is as follows:

$$v_a = R_a i_a + L_a \frac{di_a}{dt} + v_b \tag{3.119}$$

Where

v_a is the supply voltage to the armature

v_b is the back e.m.f. voltage

R_a is the resistance of the armature winding

L_a is the leakage inductance in the armature winding

It should be noted here that the leakage inductance is usually neglected.

The mechanical equation of the motor which is obtained by applying the Newton's second law to the rotor is as follows:

$$J_m \frac{d\omega_m}{dt} = T_m - T_L - b_m \omega_m \tag{3.120}$$

Where:

J_m is the rotor inertia

b_m is the equivalent damping constant of the rotor

T_m is the rotor torque

T_L is the load torque of the motor rotor

The mechanical and electrical equations of the dc motor in Laplace domain become:

$$v_a - v_b = (L_a s + R_a) i_a \tag{3.121}$$

$$T_m - T_L = (J_m s + b_m) \omega_m \tag{3.122}$$

Equations (3.121) and (3.122) in addition to equations (3.117) and (3.118) give us the model of the dc motor which is used for the locomotion of the system and will be used for simulation purpose.

3.5 SYSTEM SIMULATION

Simulation has been recognized as an important research tool since the beginning of the 20th century and now the simulation is a powerful visualization, planning, and strategic tool in different areas of research and development. The simulation has also a very important role in robotics. Different tools are used for the analysis of kinematics and dynamics of robotic systems, for off-line programming, to design different control algorithms, to design mechanical structure of robots, to design robotic cells and production lines, etc. However, a suitable computer package is needed to do the task of simulation for robotic systems. Matlab and Matlab Simulink are the two powerful softwares which have been used broadly in the field of robotics research and we used them in this project to do the simulations. The Simulink block diagram of the different parts of the robot model which has been derived in the previous sections of this chapter are shown and explained separately in this section.

The Simulink block diagram of the robot mechanical model which includes the dynamic and kinematic models is shown in Figure 3-7:

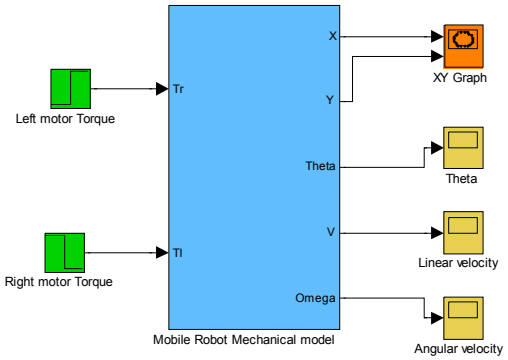

Figure 3-7: The Simulink block diagram to simulate the Mechanical part of the robot model

The above block diagram simulates the mechanical model without including the actuator effects. We simply apply different torques as the output of the right and left wheel actuators to the mechanical model and analyze the motion of the robot in response to

these different torques. The details of the mobile robot mechanical model block which is the main block in the above figure are shown in Figure 3-8:

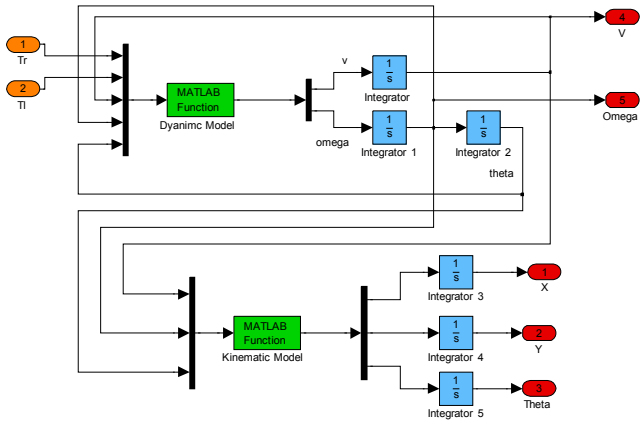

Figure 3-8: The Simulink block diagram of the Dynamic and Kinematic models

Equations (3.80) to (3.88) for the dynamic model and equation (3.52) for the kinematic model are implemented in the form of Matlab functions to simulate the robot mechanical model as is shown in the above figure. The dynamic model and kinematic model Matlab functions are included in the appendix.

The next step is to add the actuator block diagram and simulate their model which is shown in Figure 3-9:

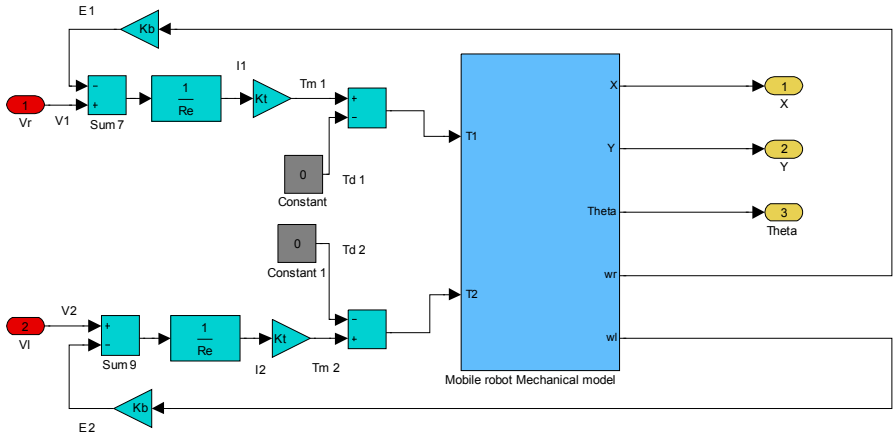

Figure 3-9: The Simulink block diagram of the actuator models added to the Mechanical model

Equations (3.117), (3.118), (3.121) and (3.122) are used to implement each wheels dc motor models in Matlab Simulink. Adding the actuator model to the mechanical model, we can simulate the complete system which is shown in Figure 3-10:

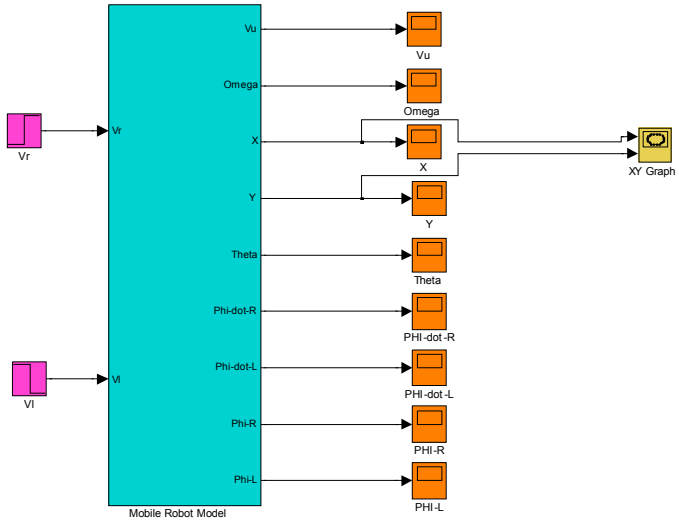

Figure 3-10: The Simulink block diagram of the complete robot model

After construction the model in Matlab Simulink we should define the robot parameters and do some tests to verify the performance of the model. The robot parameters used to simulate the model are shown in Figure 3-11:

parameter name	parameter value	description
M	12 (KG)	the robot weigh with three batteries
L	20 (cm)	the distance between the drive wheel and the axis of symmetry
a	5 (cm)	the distance between the center of mass and drive wheel axis
Ra	7.5 (cm)	wheel radius
I	5 (KG.m2)	the mass moment of inertia about the center of mass
Km	0.35(N.m/A)	motor torque constant
Kb	0.35(V.s/rad)	motor back-emf constant
B	0(N.m.s)	viscous friction
Le	0(Henry)	motor inductance
Re	8(ohm)	resistance of the dc motor armature

Figure 3-11: the robot parameters used for simulation

The above parameters are found from the robot data sheet. The parameters that have negligible effect are considered to be zero. After defining the robot parameters in one m-file, we can do different test to verify the model. The first test is to apply same magnitude but opposite sign voltages to the robot motors. The robot motion in response to such input is shown in Figure 3-12:

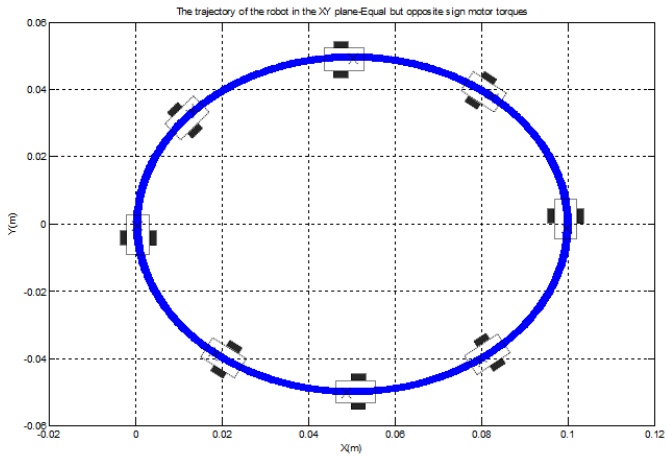

Figure 3-12: the robot motion in response to same magnitude opposite sign motor voltages

As it can be seen from the above figure, the robot will make circle around a center point when we have equal opposite sign motor torques. The robot is supposed to rotate around itself in response to such an input. The reason it makes a circle and rotates about it is because of the misalignment between its center of mass and center of rotation. The other test that can verify the model is to give same sign but not equal input torques to the robot and observe the motion. Figures 3-13 and 3-14 show the motion of the robot in response to the same sign but not equal input torques:

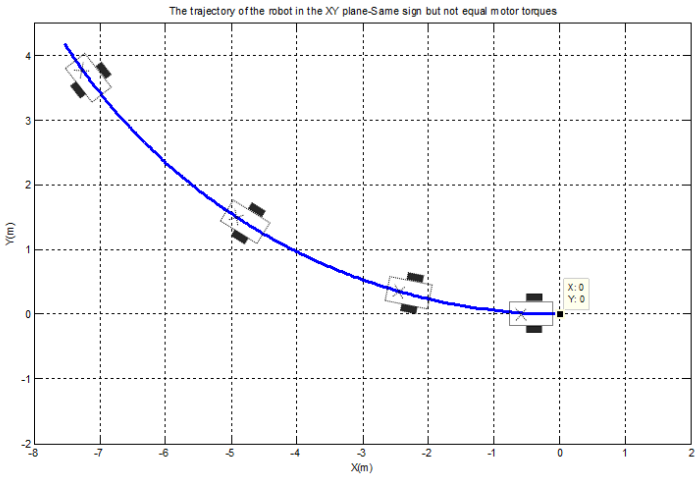

Figure 3-13: the motion of the robot in response to two different negative input torques

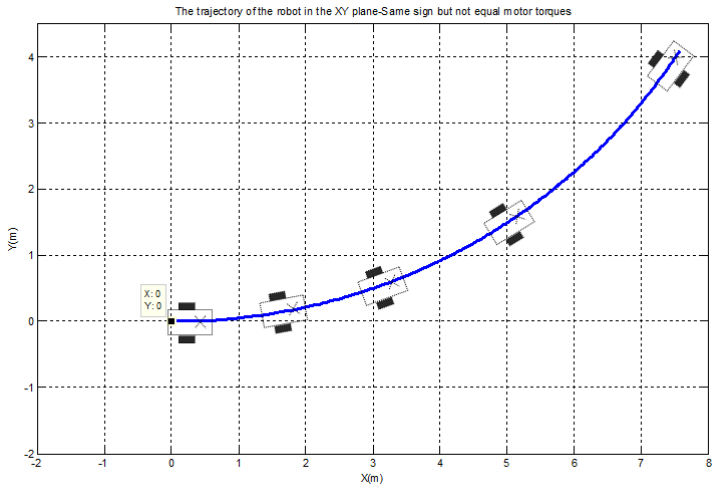

Figure 3-14: the motion of the robot in response to two different positive input torques

By taking a look at the above figures, one can find out that the simulation results match the motion of a differential drive mobile robot and the derived model can be used for control and analysis purposes which will be done in the next chapters.

CHAPTER 4

TRAJECTORY TRACKING CONTROLLER DESIGN

In the literature, the nonholonomic tracking problem is simplified by neglecting the vehicle dynamics and considering only the steering system. To compute the vehicle control inputs, it is assumed that there is a perfect velocity tracking. There are three problems with this approach: first, the perfect velocity tracking assumption does not hold in practice, second, disturbances are ignored and finally, complete knowledge of dynamics in needed. The two different proposed control approaches in this thesis correct this omission by means of a NN controller. The first algorithm provides a method of taking into account the vehicle dynamics to convert the steering system commands into control inputs for the actual vehicle. Simply, we desire to convert the prescribed control input $v(t)$ from the kinematic based controller into a torque control $\tau(t)$ for the actual physical cart. This method allows the steering system commands $v(t)$ to be converted to torques that take into account the mass, friction , etc. parameters of the actual cart. In order to design such a controller, the following steps should be taken: first, feedback velocity control inputs are designed based on a kinematic based controller to make the position error asymptotically stable. Then, a backstepping computed torque controller is designed such that the mobile robot's velocities converge to the given velocity inputs from the kinematic based controller. Finally, the kinematic neuro-controller that can deal with unmodeled disturbances and unstructured unmodeled dynamics in the nonholonomic mobile robot is designed.

The second proposed control algorithm is an adaptive self-tuning controller which tunes the gains of the backstepping controller online according to the robot reference trajectory and its initial posture. In this method, a neural network is needed to learn the characteristics of the plant dynamics and make use of it to determine the future inputs that will minimize error performance index so as to compensate the backstepping controller gains.

The trajectory tracking problem for a nonholonomic mobile robot can be described and formulated as follows:

Let there be a prescribed reference trajectory:

$$\dot{x}_r = v_r cos\theta_r \tag{4.1}$$

$$\dot{y}_r = v_r sin\theta_r \tag{4.2}$$

$$\dot{\theta}_r = \omega_r \tag{4.3}$$

$$q_r = [x_r \quad y_r \quad \theta_r]^T \tag{4.4}$$

$$v_r = [v_r \quad \omega_r]^T \tag{4.5}$$

With $v_r > 0$ for all t, find a smooth velocity control $v_c = f(e_p, v_r, K)$ such that $\lim_{t\to\infty}(q_r - q) = 0$, where e_p, v_r and K are the tracking position error, the reference velocity vector and the control gain vector respectively. Then compute the torque input for the dynamic equation (3.80) such that $v \to v_c$ as $t \to \infty$. This chapter details the steps and different parts of the design of this control algorithm.

4.1 KINEMATIC MODEL BASED BACKSTEPPING CONTROLLER

The kinematic based backstepping controller for a nonholonomic mobile robot is first proposed in 1992 by Kanayama [10] and is been used by so many other researchers in this field. A stable tracking control rule for a nonholonomic mobile robot which neglects the vehicle dynamics and is based on the steering system is described [10]. In this control system two postures for the robot are going to be used: the reference posture $q_r = [x_r \quad y_r \quad \theta_r]^T$ and a current posture $q_c = [x_c \quad y_c \quad \theta_c]^T$. The reference posture is the goal posture and the current posture is the real posture at the moment. The block diagram of this control structure is shown in Figure 4-1:

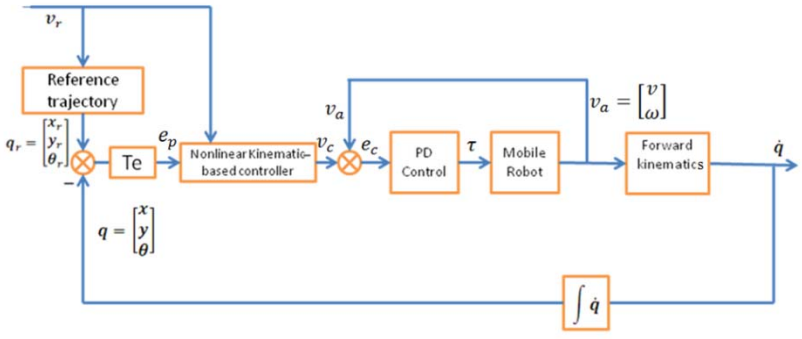

Figure 4-1: the kinematic based backstepping controller

We define an error posture or the tracking error e_p in the basis of the frame linked to the mobile platform or the local frame as follows:

$$e_p = \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = T_e(q_r - q)$$
$$\begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix} \quad (4.6)$$

The control problem in this case will be to calculate a control rule for the vehicle, which calculated the target velocities $v_c = f(e_p, v_r, K)$ that makes the system asymptotically stable. The proposed kinematic based control rule is as follows:

$$v_c = \begin{bmatrix} v_r cose_\theta + K_x e_x \\ \omega_r + K_y v_r e_y + K_\theta v_r sine_\theta \end{bmatrix}$$
$$v_c = f(e_p, v_r, K) \quad (4.7)$$
$$K = (K_x, K_y, K_\theta)$$

Where K_x, K_y and K_θ are positive constants. The PD controller in the block diagram of figure (5-1) is a linear controller which is responsible to convert the velocity output of the controller to torque inputs for the robot. The stability of the above control rule will be proved using the Lyapunov stability method in the next section.

4.1.1 Lyapunov stability analysis

The Lyapunov stability analysis of the control rule in equation (4.7) is described as follows:

Lemma 1:

According to equation (4.6) we have:

$$\begin{bmatrix} \dot{e}_x \\ \dot{e}_y \\ \dot{e}_\theta \end{bmatrix} = \dot{e}_p = f(t, e_p) = \begin{bmatrix} \omega(e_p, q_r)e_y - v(e_p, q_r) + v_r cose_\theta \\ -\omega(e_p, q_r)e_x + v_r sine_\theta \\ \omega_r - \omega(e_p, q_r) \end{bmatrix} \quad (4.8)$$

Proof:

Using the robot kinematic equations (3.3) and (3.12) and the tracking error equation (4.6) we have:

$$e_p = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix} = \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} \quad (4.9)$$

$$\begin{aligned}\dot{e}_x &= (\dot{x}_r - \dot{x})cos\theta + (\dot{y}_r - \dot{y})sin\theta - (x_r - x)\dot{\theta}sin\theta + (y_r - y)\dot{\theta}cos\theta \\ &= e_y\omega - v + \dot{x}_r\cos(\theta_r - \theta) + \dot{y}_r\sin(\theta_r - \theta) \\ &= e_y\omega - v + v_r cose_\theta\end{aligned} \tag{4.10}$$

$$\begin{aligned}\dot{e}_y &= -(\dot{x}_r - \dot{x})sin\theta + (\dot{y}_r - \dot{y})cos\theta - (x_r - x)\dot{\theta}cos\theta - (y_r - y)\dot{\theta}sin\theta \\ &= -e_x\omega - \dot{x}_r\sin(\theta_r - \theta) + \dot{y}_r\cos(\theta_r - \theta) \\ &= -e_x\omega + v_r sine_\theta\end{aligned} \tag{4.11}$$

$$\dot{e}_\theta = \dot{\theta}_r - \dot{\theta} = \omega_r - \omega \tag{4.12}$$

Substituting *v* and ω by $v(e_p, q_r)$ and $\omega(e_p, q_r)$ we obtain the lemma.

Proposition 1:

If we use the control rule in equation (4.7), $e_p = 0$ is a stable equilibrium point if the reference velocity $v_r > 0$.

Proof:

Let us propose a scalar function V as a Lyapunov function candidate for the above system:

$$V = \frac{1}{2}({e_x}^2 + {e_y}^2) + \frac{(1 - \cos(e_y))}{K_y} \tag{4.13}$$

Clearly $V \geq 0$ and $V = 0\ if\ e_p = 0$, therefore the above V function is a positive definite function.

Furthermore, by using *lemma 2* we have:

$$\begin{aligned}\dot{V} = \dot{e}_x e_x + \dot{e}_y e_y + \frac{\dot{e}_\theta \sin(e_\theta)}{K_y} \\ = \left[\left(\omega_r + v_r\left(K_y e_y + K_\theta sin(e_\theta)\right)\right)e_y - K_x e_x\right]e_x \\ + \left[-\left(\omega_r + v_r\left(K_y e_y + K_\theta sin(e_\theta)\right)\right)e_x + v_r\sin(e_\theta)\right]e_y \\ + \frac{[-v_r\left(K_y e_y + K_\theta sin(e_\theta)\right)]\sin(e_\theta)}{K_y} \\ = -K_x{e_x}^2 - \frac{v_r K_\theta sin^2 e_\theta}{K_y} \leq 0\end{aligned} \tag{4.14}$$

Therefore, the derivative of the proposed Lyapunov function V is a negative definite function which demonstrates that the point $e_p = 0$ is uniformly asymptotically stable

under the conditions that v_r and ω_r are continuous and v_r, ω_r, K_x, K_y and K_θ are bounded. The above Lyapunov stability analysis is used in reference [10] to prove the stability of the proposed controller.

We demonstrated that the system is stable for any combination of K_x, K_y andK_θ. However, since we need a non-oscillatory, but not too slow response of the robot, we have to find an optimal parameters set for this controller. At this stage, the proper gains for each reference trajectory have to found by tuning. An adaptive self-tuning controller will be explained in detail in section 4.3.3 to solve this problem. The simulation results of this controller on the robot model will be shown in the next section.

4.1.2 Kinematic model based backstepping controller simulation

The kinematic based control rule in equation (4.7) is added to the robot complete model with dynamics to see its performance on the real system. The Matlab Simulink model used to simulate this controller is shown in Figure 4-2:

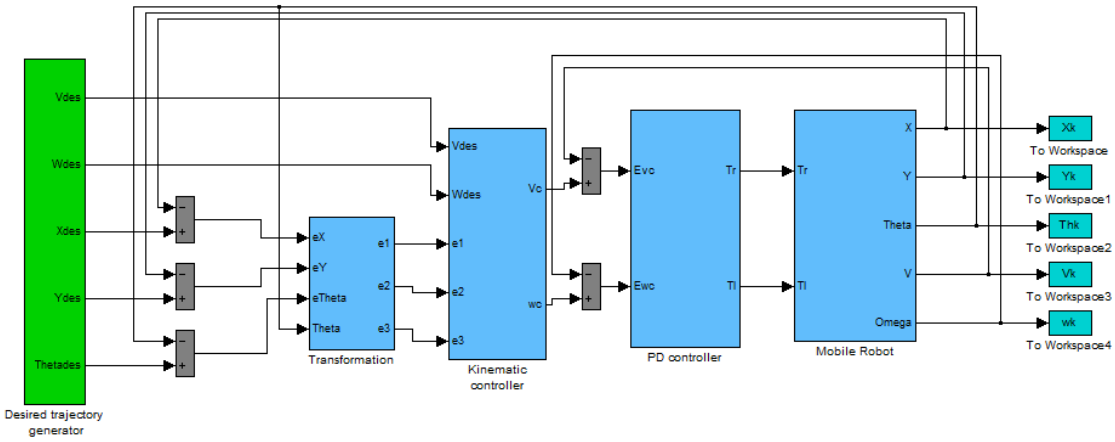

Figure 4-2: The Simulink block diagram to simulate the kinematic based controller

The desired trajectory generator block in the above block diagram generates the different desired trajectories for the robot. A trajectory is a function of time *q(t)* such that $q(t_0) = q_s$ and $q(t_f) = q_f$. In this case $t_f - t_0$ represents the amount of time taken to execute the trajectory. Since the trajectory is parameterized by time, we can compute velocities and accelerations by differentiation. It is common practice to choose trajectories from a finitely parameterizable family for example polynomials of degree *n*, where *n* depends on the number of constraints to be satisfied. Consider the mobile robot case in which we wish to generate a polynomial joint trajectory between two configurations and that we wish to specify the start and end velocities for the trajectories. This gives four constraints that the trajectory must satisfy. Therefore at minimum we require a polynomial with four

independent coefficients that can be chosen to satisfy these constraints. Thus we consider a cubic trajectory of the form:

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{4.15}$$

Then the desired velocity will be given as:

$$\dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^2 \tag{4.16}$$

Combining equations (4.15) and (4.16) with four constraints yields four equations with fours unknowns as follows:

$$\begin{bmatrix} 1 & t_0 & {t_0}^2 & {t_0}^3 \\ 0 & 1 & 2t_0 & 3{t_0}^2 \\ 1 & t_f & {t_f}^2 & {t_f}^3 \\ 0 & 1 & 2t_f & 3{t_f}^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} q_0 \\ v_0 \\ q_f \\ v_f \end{bmatrix} \tag{4.17}$$

Equation (4.17) always has a unique solution for nonzero time interval provided for the trajectory execution. For the control simulation purpose, we define q_0 and q_f as the start and end position of the robot and we assume $v_0 = v_f = 0$ for this case. The shape of the path which can be any path such as a straight line or a sinusoidal is defined as a function between *x* and *y* component of the robot path. The implementation of the cubic trajectory and the path shape is done through a Matlab function which is included in the appendix and is used in the trajectory generator block in Figure 4-2 as shown in Figure 4-3:

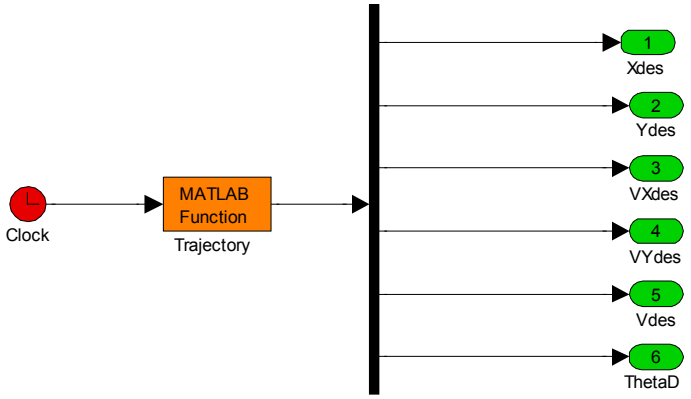

Figure 4-3: the trajectory generator block in Matlab Simulink

Using the block diagram in figure 4-2 and the trajectory generator block and Matlab function, we can test the kinematic based controller with different reference trajectories, different initial positions for the robot and different controller gains. Two typical simulation results for a straight line tracking are shown in Figures 4-4 and 4-5:

Robot initial states(x, y, θ)*: (1, 5, 0)*

Kinematic based controller gains: $K_x = 110\ K_y = 20\ K_\theta = 100$

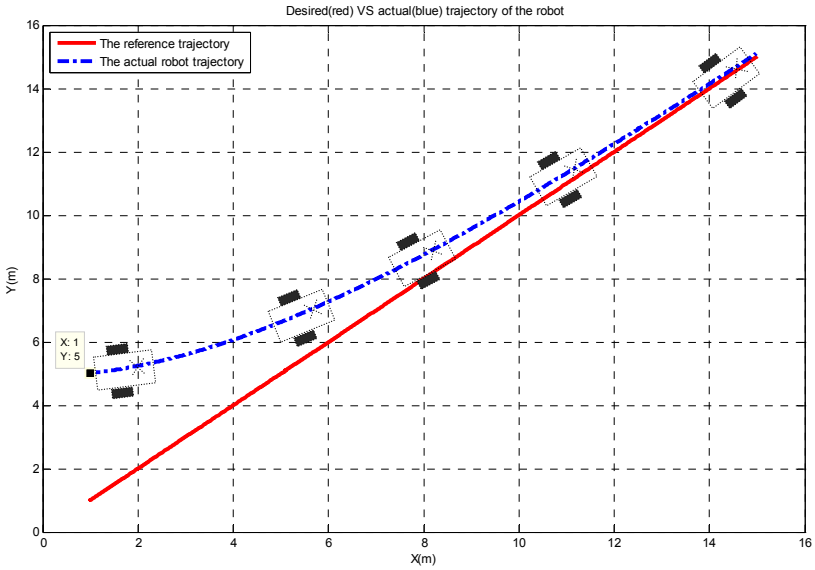

Figure 4-4: the robot actual and reference trajectory in x-y plane/Robot initial states(x, y, θ): (1, 5, and 0)

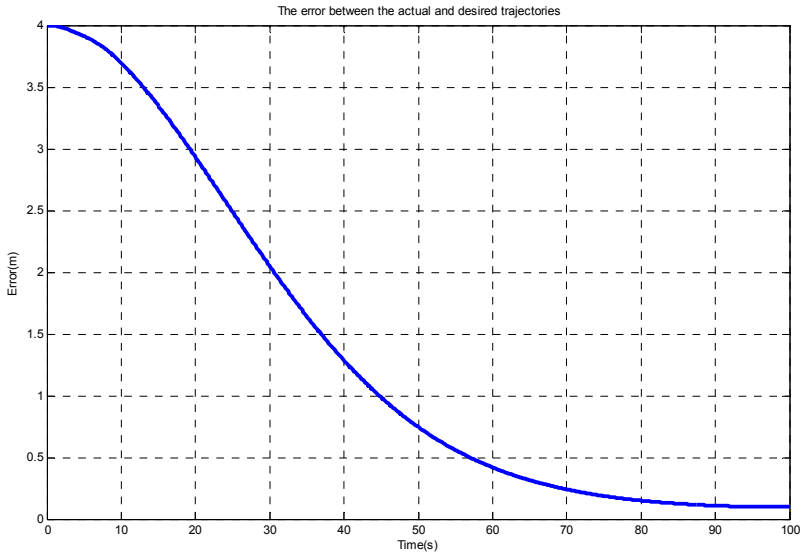

Figure 4-5: the robot trajectory error /Robot initial states(x, y, θ): (1, 5, and 0)

Figure (4-4) shows the robot trajectory and the reference trajectory in the x-y plane. As it can be seen the robot response is slow and the trajectory error is around zero after more than 100 seconds. The trajectory error shown in figure (20) is calculated as follows:

$$e_{tr} = \sqrt{{e_x}^2 + {e_y}^2} = \sqrt{(x - x_r)^2 + (y - y_r)^2} \tag{4.18}$$

The kinematic based controller gains are tuned to have the fastest tracking response but as it can be seen from the above figure we cannot reach a desirable fast tracking response because this control algorithm neglects the dynamics of the system. Another trajectory tracking response with different robot initial position and controller gains is shown Figures 4-6 and 4-7 to clarify this controller performance:

Robot initial states(x, y, θ)*: (5, 1, 0)*

Kinematic based controller gains: $K_x = 50\ K_y = 10\ K_\theta = 60$

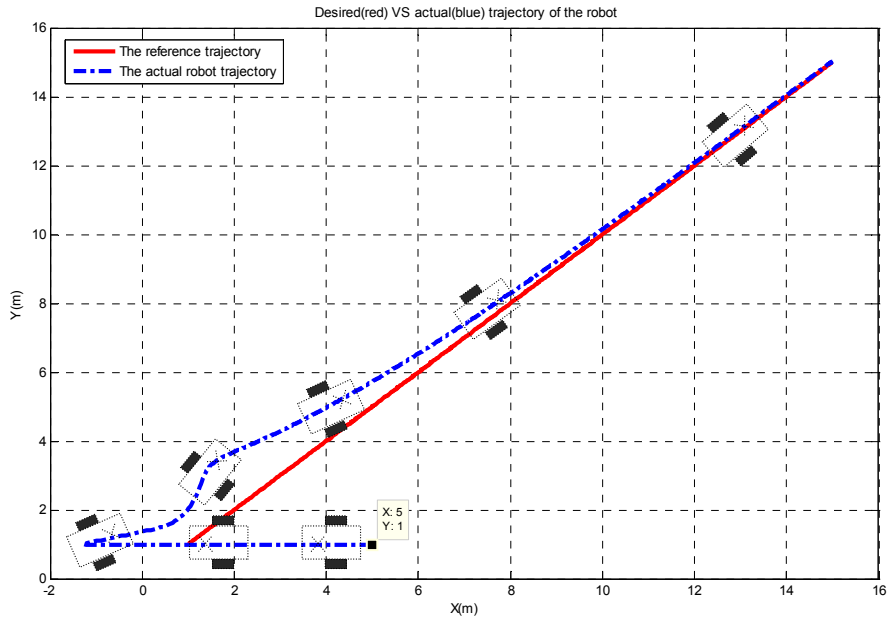

Figure 4-6: the robot actual and reference trajectory in x-y plane/Robot initial states(x, y, θ): (5, 1, and 0)

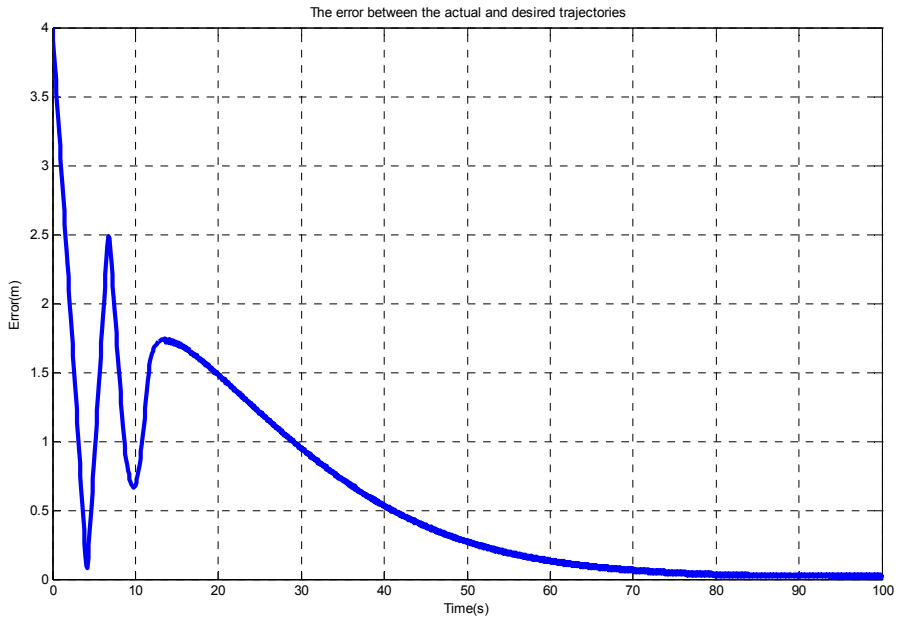

Figure 4-7: the robot trajectory error /Robot initial states(x, y, θ): (5, 1, and 0)

As it can be seen and concluded from the above figures, using the kinematic based controller will eventually make the error zero but the system tracking response is not desirable because in some cases it has too much oscillation and in the other cases it is very slow. There are two reasons for not having a desirable tracking response for the kinematic based controller. The first reason is that it neglects the system dynamics such as vehicles mass, moment of inertia and coriolis and centrifugal velocities. The second reason is that the controller gains should be tuned according to one reference trajectory and robot initial posture. One set of gains will be suitable for one reference trajectory and not others. The next step to improve the control architecture for this system is to add a nonlinear feedback linearization controller to the kinematic based controller. This method will linearize the nonlinear system dynamics and improves the trajectory tracking response. The combination of the kinematic based controller and the nonlinear feedback makes the control structure a backstepping controller with a nonlinear feedback which will be explained in the next section.

4.2 THE BACKSTEPPING CONTROLLER DESIGN USING THE NONLINEAR FEEDBACK METHOD

The previous section was about selecting a velocity control *v(t)* defined in equation (4.7) for the steering system or the kinematic model of the robot. In this section we desire to convert such a prescribed control *v(t)* into a torque control $\tau(t)$ for the actual physical platform. As it is mentioned in the modeling chapter, the complete equations of motion of the nonholonomic mobile platform are given by equations (3.72) and (3.80):

$$\dot{q} = \begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\theta} \end{bmatrix} = S(q)v(t) = \begin{bmatrix} cos\theta & -asin\theta \\ sin\theta & acos\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{4.19}$$

$$\bar{M}(q)\dot{v}(t) + \bar{V}_m(q,\dot{q})v(t) + \bar{F}(\dot{q}) + \bar{G}(q) + \bar{\tau}_d = \bar{B}(q)\tau \tag{4.20}$$

The complete dynamics consists of kinematic steering system (4.19) plus some extra dynamics (4.20). Standard approaches to nonholonomic control such as the kinematic based controller deal only with (4.19) ignoring the actual vehicle dynamics. This omission will be corrected in this section.

Feedback linearization is an approach to nonlinear control design which has attracted a great deal of research interest in recent years. The central idea of this approach is to first transform a nonlinear system into a (fully or partially) linear system and then use the well known and powerful linear design techniques to complete the control design. The application of this method to the case of the nonholonomic mobile robot is that by applying an appropriate nonlinear feedback we can linearize the nonlinear dynamics and then apply the kinematic based controller to the linear system. Let *u* be an auxiliary input, then by applying the nonlinear feedback:

$$\tau = \bar{B}^{-1}(q)[\bar{M}(q)u + \bar{V}_m(q,\dot{q})v(t) + \bar{F}(\dot{q})] \tag{4.21}$$

We can convert the dynamic equation (4.20) to:

$$\begin{aligned} \bar{M}(q)\dot{v}(t) + \bar{V}_m(q,\dot{q})v(t) + \bar{F}(\dot{q}) + \bar{G}(q) + \bar{\tau}_d \\ = \bar{B}(q)\left[\bar{B}^{-1}(q)[\bar{M}(q)u + \bar{V}_m(q,\dot{q})v(t) + \bar{F}(\dot{q})]\right] \\ \dot{v}(t) = u \end{aligned} \tag{4.22}$$

Therefore the complete equations of motion of the system will be:

$$\begin{aligned} \dot{q} = S(q)v(t) \\ \dot{v}(t) = u \end{aligned} \tag{4.23}$$

In performing the nonlinear feedback (4.21) it is assumed that all the dynamical quantities such as $\bar{M}(q)$, $\bar{V}_m(q,\dot{q})$ and $\bar{F}(\dot{q})$ are exactly known and $\bar{\tau}_d = 0$. Although a nonlinear system can be controllable, a stabilizable smooth state feedback may not exist. This is the case of the system (4.23) where the equilibrium point cannot be made asymptotically stable by any smooth time-invariant state feedback controller. A backstepping controller is used to stabilize system (4.23). Consider a system given by:

$$\begin{aligned} \dot{x} = f(x) + g(x)\xi \\ \dot{\xi} = u \end{aligned} \tag{4.24}$$

With *f(0)=0* and *f* and *g* are smooth functions. Designing a feedback control to stabilize the system at *(x=0, ξ=0)* is called a backstepping controller. Comparing equations (4.24)

and (4.23) one can find out that the linearized nonholonomic mobile robot equations are in the form of (4.24) and a backstepping control algorithm will make it stable. The proposed control input to stabilize system (4.23) is as follows:

$$u = \dot{v}_c + K_4(v_c - v) \tag{4.25}$$

Where:

$$v_c = \begin{bmatrix} v_r cose_\theta + K_x e_x \\ \omega_r + K_y v_r e_y + K_\theta v_r sine_\theta \end{bmatrix}$$
$$v_c = f(e_p, v_r, K)$$
$$K = (K_x, K_y, K_\theta) \tag{4.26}$$

$$\dot{v}_c = \begin{bmatrix} \dot{v}_r cose_\theta - v_r \dot{e}_\theta sine_\theta + K_x \dot{e}_x \\ \dot{\omega}_r + K_y \dot{v}_r e_y + K_y v_r \dot{e}_y + K_\theta \dot{v}_r sine_\theta + K_\theta v_r \dot{e}_\theta cose_\theta \end{bmatrix} \tag{4.27}$$

$$\dot{v}_c = \begin{bmatrix} \dot{v}_r cose_\theta \\ \dot{\omega}_r + K_y \dot{v}_r e_y + K_\theta \dot{v}_r sine_\theta \end{bmatrix} + \begin{bmatrix} K_x & 0 & -v_r sine_\theta \\ 0 & K_y v_r & K_\theta v_r cose_\theta \end{bmatrix} \dot{e}_p \tag{4.28}$$

$$e_p = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix} = \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} \tag{4.29}$$

K_4 is a positive definite, diagonal matrix given by:

$$K_4 = k_4 I \tag{4.27}$$

The general backstepping control structure is shown in the block diagram of Figure 4-8:

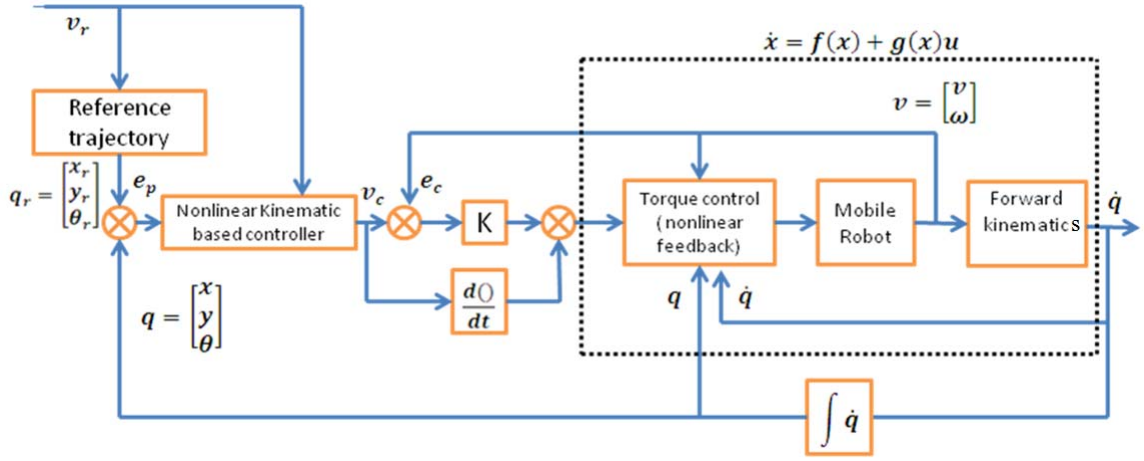

Figure 4-8: the backstepping controller with the nonlinear feedback structure

The stability of the above control algorithm (4.25) will be done using the Lyapunov stability analysis in the next section.

4.2.1 Lyapunov stability analysis

The controller stability problem can be stated as follows:

Proposition 1:

Given a nonholonomic system (4.19) and (4.20), let a nonlinear feedback control *u* given by equation (4.25) be used and the vehicle input commands be given by equation (4.21). Then the origin $e_p = 0$ is uniformly asymptotically stable, and the velocity vector of the mobile robot platform satisfies $v \to v_c$ as $t \to \infty$.

Proof:

We define the error between the kinematic controller output v_c and the mobile platform velocity *v* as follows:

$$e_c = v - v_c \tag{4.28}$$

$$e_c = \begin{bmatrix} e_4 \\ e_5 \end{bmatrix} = \begin{bmatrix} v \\ \omega \end{bmatrix} - \begin{bmatrix} v_r cose_\theta + K_x e_x \\ \omega_r + K_y v_r e_y + K_\theta v_r sine_\theta \end{bmatrix} = \begin{bmatrix} v - v_r cose_\theta - K_x e_x \\ \omega - \omega_r - K_y v_r e_y - K_\theta v_r sine_\theta \end{bmatrix} \tag{4.29}$$

Substituting equation (4.25) in equation (4.23) we have:

$$\dot{v}(t) - \dot{v}_c = -K_4(v - v_c) \tag{4.30}$$

Therefore, from equations (4.30) and (4.28) we obtain:

$$\dot{e}_c = -K_4 e_c \tag{4.31}$$

The above equation is an asymptotically stable system and shows that the velocity vector of the robot satisfies $v \to v_c$ as $t \to \infty$ or:

$$\lim_{t\to\infty} e_4 = 0 \tag{4.32}$$

$$\lim_{t\to\infty} e_5 = 0 \tag{4.33}$$

The Lyapunov function candidate to prove the stability of the system (4.23) with the controller (4.25) is:

$$V = K_x\left({e_x}^2 + {e_y}^2\right) + \frac{2k_x}{k_y}(1 - cose_\theta) + \frac{1}{k_4}\left({e_4}^2 + \frac{K_x}{k_y K_\theta v_r}{e_5}^2\right) \tag{4.34}$$

It can easily be seen that the above function is a Lyapunov function because $V \geq 0$ and $V = 0$ only if $e_p = 0$ and $e_c = 0$ which is considered as the equilibrium point of the system. The derivative of the above Lyapunov function is:

$$\dot{V} = 2K_x\dot{e}_x e_x + 2K_x\dot{e}_y e_y + \frac{2k_x}{k_y}\dot{e}_\theta sine_\theta + \frac{2}{k_4}\dot{e}_4 e_4 + \frac{2K_x}{k_4 k_y K_\theta v_r}\dot{e}_5 e_5 \tag{4.35}$$

Substituting equation (4.31) in the above equation we have:

$$\dot{V} = 2K_x\dot{e}_x e_x + 2K_x\dot{e}_y e_y + \frac{2k_x}{k_y}\dot{e}_\theta sine_\theta - 2{e_4}^2 - \frac{2K_x}{k_y K_\theta v_r}{e_5}^2 \tag{4.36}$$

Using equation (4.8) we have:

$$\begin{aligned}\dot{V} = {} & 2K_x e_x\left[\omega e_y - v + v_r cose_\theta\right] + 2K_x e_y[-\omega e_x + v_r sine_\theta] \\ & + \frac{2k_x}{k_y}sine_\theta[\omega_r - \omega] - 2{e_4}^2 - \frac{2K_x}{k_y K_\theta v_r}{e_5}^2\end{aligned} \tag{4.37}$$

Substituting equation (4.29) in (4.37) we have:

$$\begin{aligned}\dot{V} = {} & 2K_x e_x\left[\omega e_y - v + v_r cose_\theta\right] + 2K_x e_y[-\omega e_x + v_r sine_\theta] \\ & + \frac{2k_x}{k_y}sine_\theta[\omega_r - \omega] - 2(v - v_r cose_\theta - K_x e_x)^2 \\ & - \frac{2K_x}{k_y K_\theta v_r}(\omega - \omega_r - K_y v_r e_y - K_\theta v_r sine_\theta)^2\end{aligned} \tag{4.38}$$

Simplifying and reshaping the above equation, we have:

$$\begin{aligned}\dot{V} = {} & -(v - v_r cose_\theta)^2 - {k_x}^2{e_x}^2 - \frac{K_x}{k_y K_\theta v_r}(\omega - \omega_r - K_y v_r e_y)^2 \\ & - \frac{K_x K_\theta}{k_y}v_r sin^2 e_\theta\end{aligned} \tag{4.39}$$

Using equation (4.29) again, we obtain:

$$\dot{V} = -{k_x}^2{e_x}^2 - \frac{K_x K_\theta}{k_y}v_r sin^2 e_\theta - (e_4 + K_x e_x)^2 - \frac{K_x}{k_y K_\theta v_r}(e_5 + K_\theta v_r sine_\theta)^2 \tag{4.40}$$

By taking a look at equation (4.40) one can easily find out that the derivative of the proposed Lyapunov function is negative $\dot{V} \leq 0$ and therefore, the entire error $e = \left[{e_p}^T {e_c}^T\right]^T$ is bounded.

Considering the fact that $e_c \to 0$ as $t \to \infty$ then the limit of the Lyapunov function derivate (4.38) as $t \to \infty$ will be:

$$\begin{aligned}\lim_{t\to\infty} \dot{V} = \lim_{t\to\infty} \Bigg(& -{k_x}^2 {e_x}^2 - \frac{K_x K_\theta}{k_y} v_r sin^2 e_\theta - (e_4 + K_x e_x)^2 \\ & - \frac{K_x}{k_y K_\theta v_r} (e_5 + K_\theta v_r sine_\theta)^2 \Bigg) \\ & = \lim_{t\to\infty} ({k_x}^2 {e_x}^2 + \frac{K_x K_\theta}{k_y} v_r sin^2 e_\theta)\end{aligned} \tag{4.41}$$

Since the derivative of the Lyapunov function is negative $\dot{V} \leq 0$ and we know that V does not increase and converge to a constant value, $\dot{V} \to 0$ as $t \to \infty$. Therefore the limit in equation (4.41) will be:

$$\lim_{t\to\infty} \dot{V} = \lim_{t\to\infty} ({k_x}^2 {e_x}^2 + \frac{K_x K_\theta}{k_y} v_r sin^2 e_\theta) = 0 \tag{4.42}$$

From the above equation we have:

$$\lim_{t\to\infty} e_x = 0 \tag{4.43}$$

$$\lim_{t\to\infty} e_\theta = 0 \tag{4.44}$$

Equations (4.8) and (4.44) will give:

$$\lim_{t\to\infty} (\omega_r - \omega) = 0 \tag{4.45}$$

Using equation (4.29), (4.44) and (4.45) and knowing that $e_c \to 0$ as $t \to \infty$ we have:

$$\lim_{t\to\infty} (\omega - \omega_r - K_y v_r e_y - K_\theta v_r sine_\theta) = \lim_{t\to\infty} \left(-K_y v_r e_y\right) = 0 \tag{4.46}$$

Therefore:

$$\lim_{t\to\infty} e_y = 0 \tag{4.47}$$

According to equations (4.32), (4.33), (4.44), (4.45) and (4.47) the equilibrium point $e = \left[{e_p}^T {e_c}^T\right]^T = 0$ is uniformly asymptotically stable.

The above stability analysis shows that the backstepping controller (4.25) along with the nonlinear feedback (4.21) will make the complete nonholonomic mobile robot system

(4.19) and (4.20) track the reference trajectory or it makes the tracking error system stable. The tracking performance depends on both kinematic based controller gains $K = (K_x, K_y, K_\theta)$ and the backstepping controller gain K_4. The backstepping controller simulation results are shown in the next section.

4.2.2 The backstepping controller with nonlinear feedback simulation results

The backstepping controller structure shown in figure (4.8) is used to construct the Matlab Simulink block diagram to simulate the system response with this controller. The Matlab Simulink block diagram for this control structure is shown in Figure 4-9:

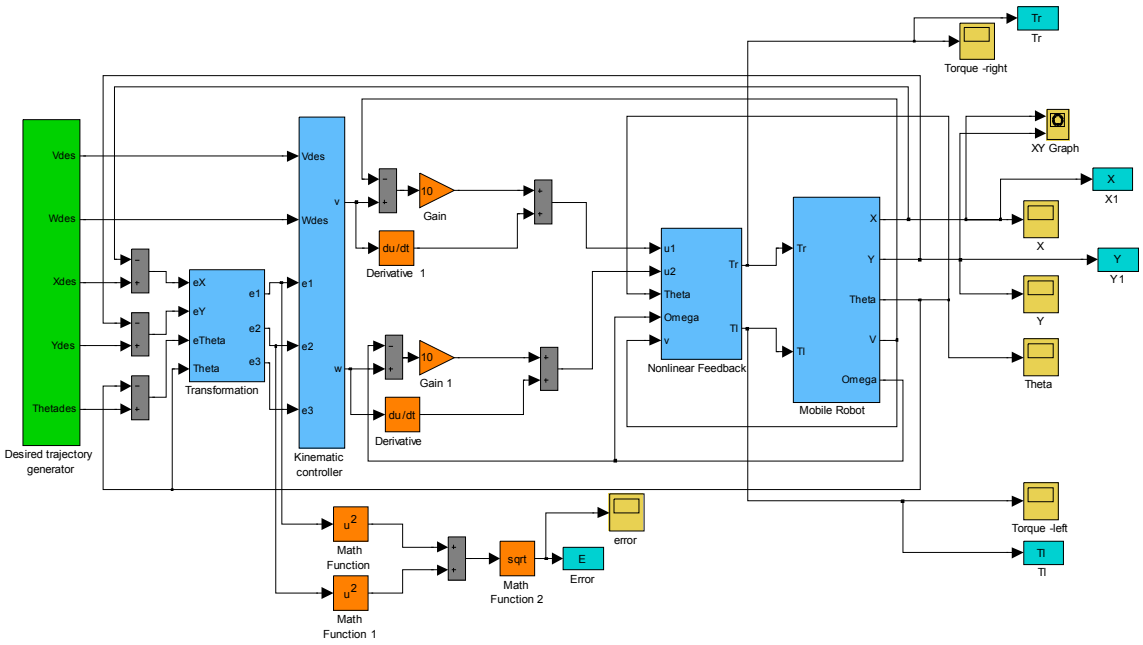

Figure 4-9: the Matlab Simulink block diagram to simulate the backstepping controller response

Using the above block diagram we can test the system response to different reference trajectories with different controller gains. The simulation is done with a fixed step solver with the step size of 0.1 seconds. The solver type is ode4 (Runge-Kutta). The robot tracking response with different initial positions and controller gains are shown Figures 4-10 and 4-11:

$$\textit{The robot initial states: } x = 3, y = 1, \theta = \frac{3pi}{2}$$

$$\textit{The backstepping controller gains: } k_x = 1, k_y = 55, k_\theta = 15, k_4 = 10$$

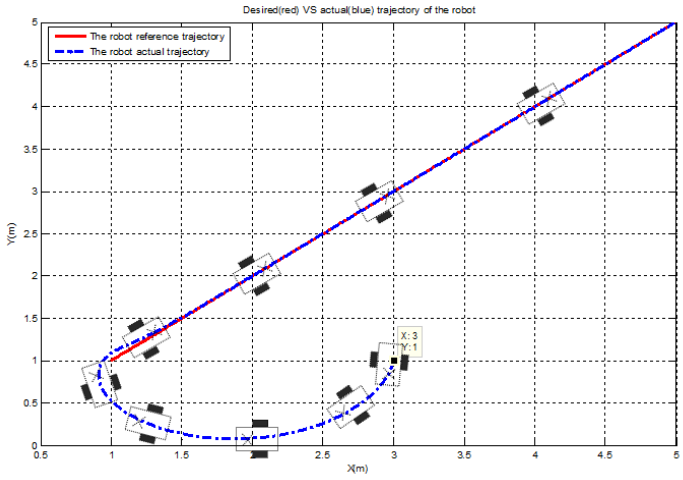

Figure 4-10: the robot actual and reference trajectory in x-y plane/Robot initial states(x, y, θ): (3, 1, and 3pi/2)

The above figure shows the trajectory of the robot when its initial location is far from the trajectory. The motion of the robot is because of its nonholonomic constraint which imposes the no-lateral motion. The controller gains are tuned to have the fastest trajectory tracking response. The trajectory error which is calculated using equation (4.18) shows the controller performance more clearly. The trajectory error response of the above controller and initial position case is shown in the Figure 4-11:

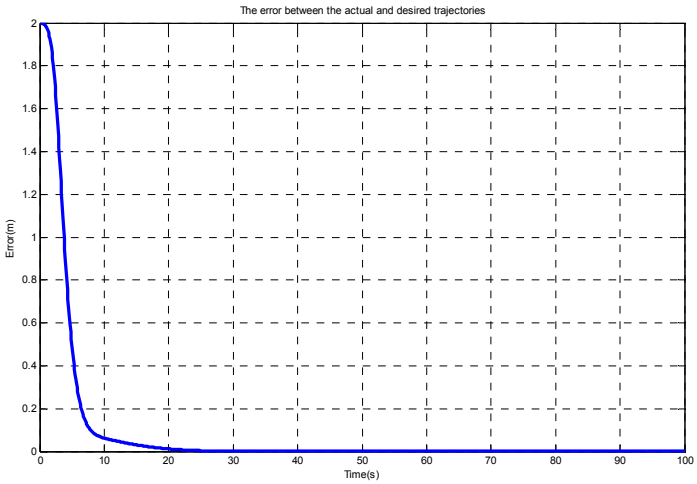

Figure 4-11: the robot trajectory error /Robot initial states(x, y, θ): (3, 1, and 3pi/2)

The above trajectory error response shows that the controller makes the system asymptotically stable. In order to analyze the system performance, the time response of the robot states $q_c = [x_c \quad y_c \quad \theta_c]^T$ and the reference states $q_r = [x_r \quad y_r \quad \theta_r]^T$ are shown in Figures 4-12, 4-13 and 4-14:

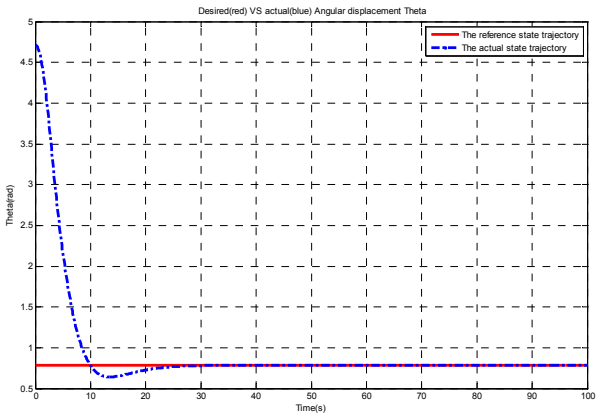

Figure 4-12: the robot angular displacement response with time/ Robot initial states(x, y, θ): (3, 1, and 3pi/2)

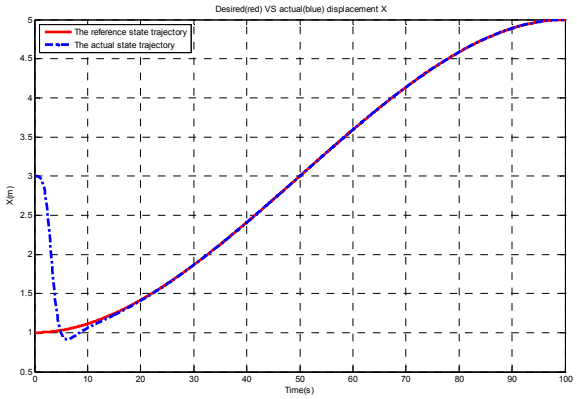

Figure 4-13: the robot linear displacement in x-direction response with time/ Robot initial states(x, y, θ): (3, 1, and 3pi/2)

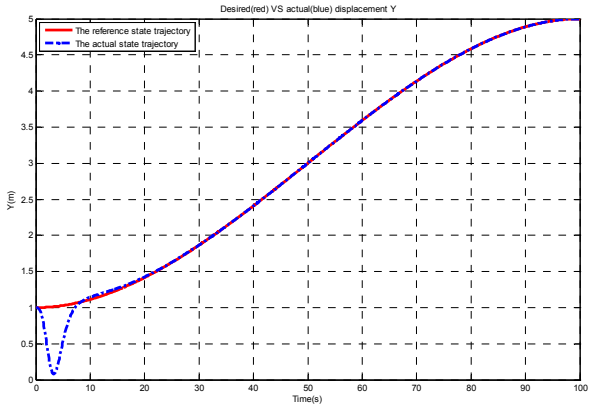

Figure 4-14: the robot linear displacement in y-direction response with time/ Robot initial states(x, y, θ): (3, 1, and 3pi/2)

The above three figures show that the system states reach their reference trajectories in about 20 seconds which shows a relatively fast tracking response for the controller. The torque needed from the robot wheel motors to provide such a tracking response is shown in Figures 4-15 and 4-16:

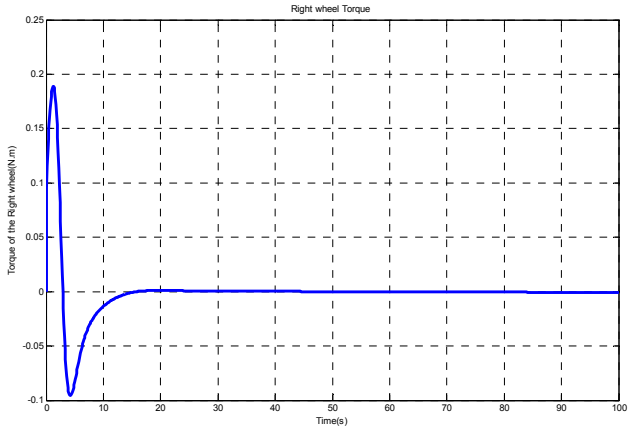

Figure 4-15: the robot right wheel torque response with time

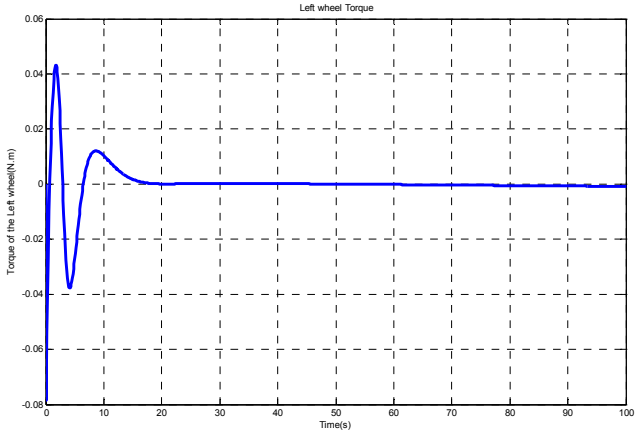

Figure 4-16: the robot left wheel torque response with time

The above two figures show that the torque needed for such a trajectory tracking response can be provided by the system actuators. Therefore the designed controller is applicable to the real platform. The system response with the robots some other different initial positions are shown in the following figures to clarify the performance of the controller:

The robot initial states: $x = 3, y = 1, \theta = 0$

The backstepping controller gains: $k_x = 1, k_y = 55, k_\theta = 15, k_4 = 10$

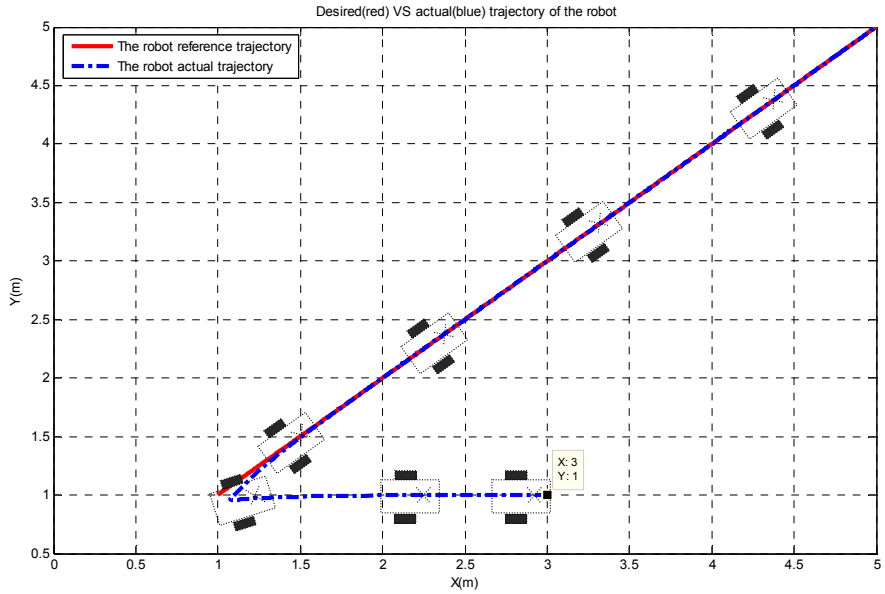

Figure 4-17: the robot actual and reference trajectory in x-y plane/Robot initial states(x, y, θ): (3, 1, and 0)

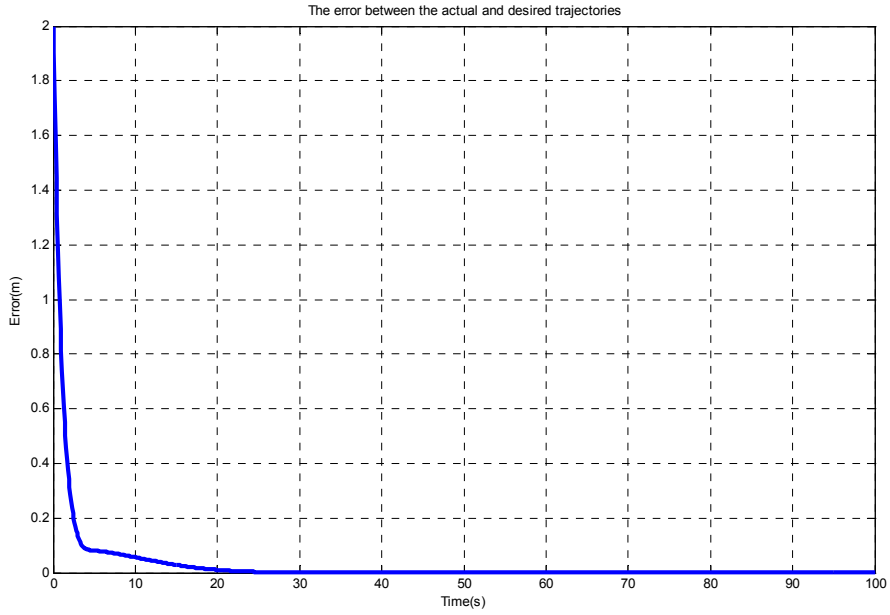

Figure 4-18: the robot trajectory error /Robot initial states(x, y, θ): (3, 1, and 0)

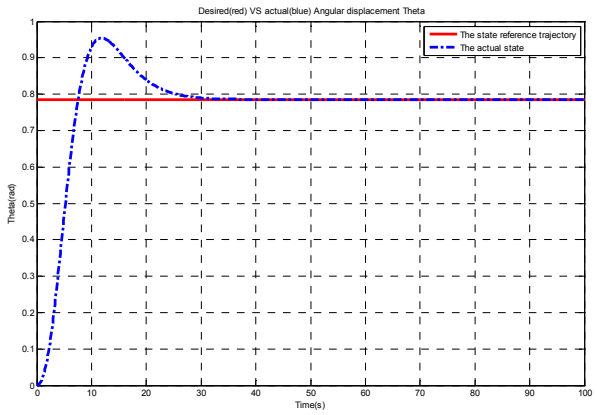

Figure 4-19: the robot angular displacement response with time/ Robot initial states(x, y, θ): (3, 1, and 0)

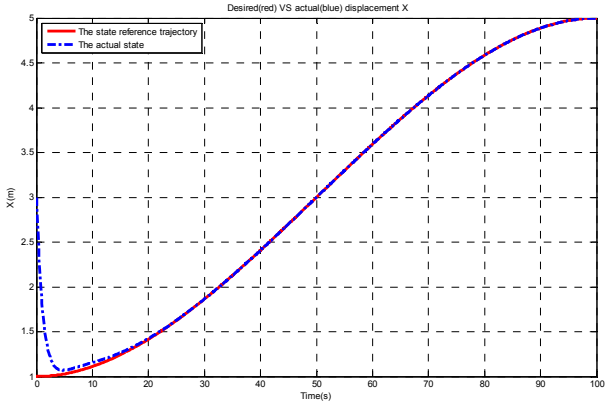

Figure 4-20: the robot linear displacement in x-direction response with time/ Robot initial states(x, y, θ): (3, 1, and 0)

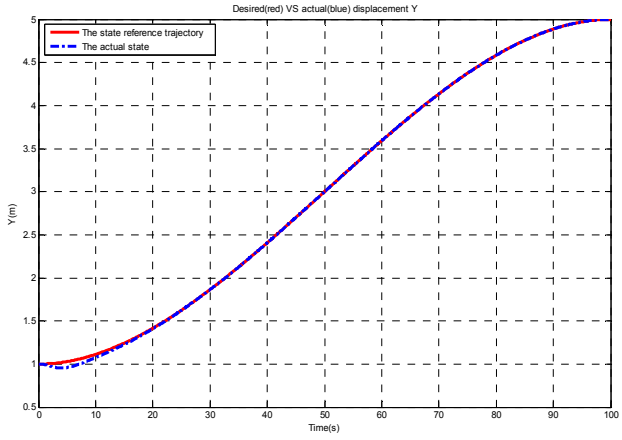

Figure 4-21: the robot linear displacement in y-direction response with time/ Robot initial states(x, y, θ): (3, 1, and 0)

Different robot trajectory tracking responses with different initial conditions and different trajectory types are shown in the following figures:

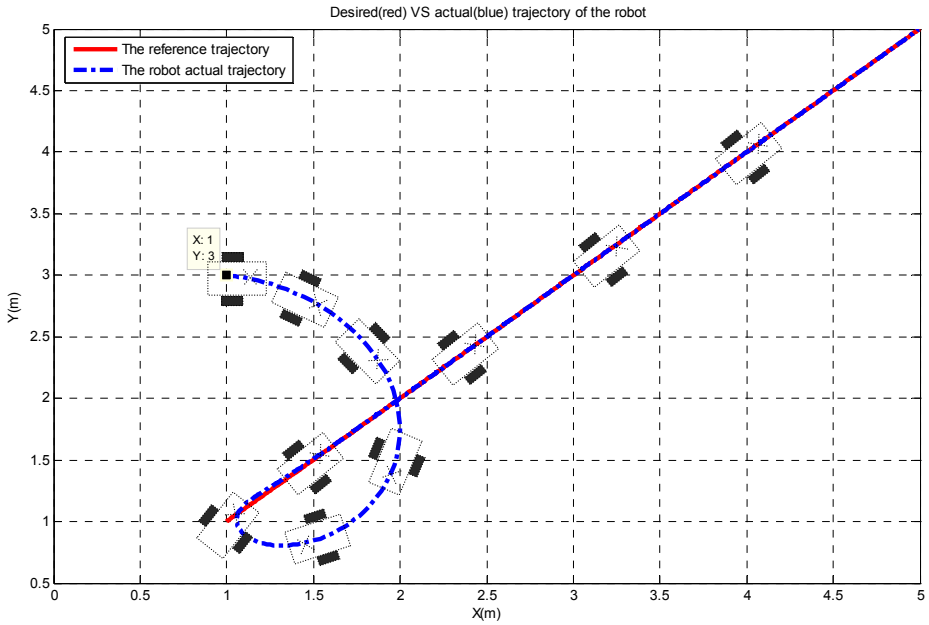

Figure 4-22: the robot actual and reference trajectory in x-y plane/Robot initial states(x, y, θ): (1, 3, and 0)

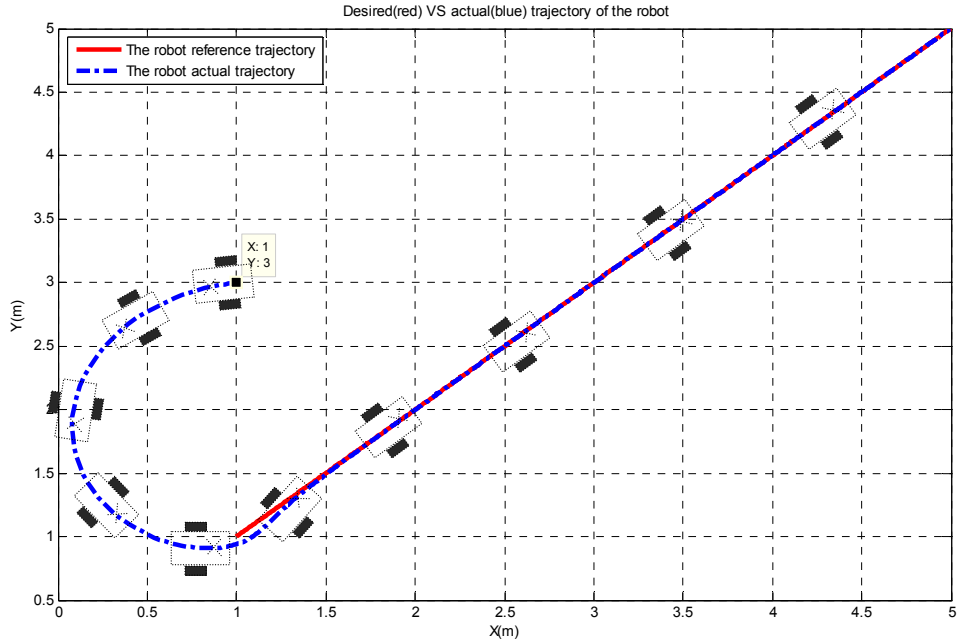

Figure 4-23: the robot actual and reference trajectory in x-y plane/Robot initial states(x, y, θ): (1, 3, and pi)

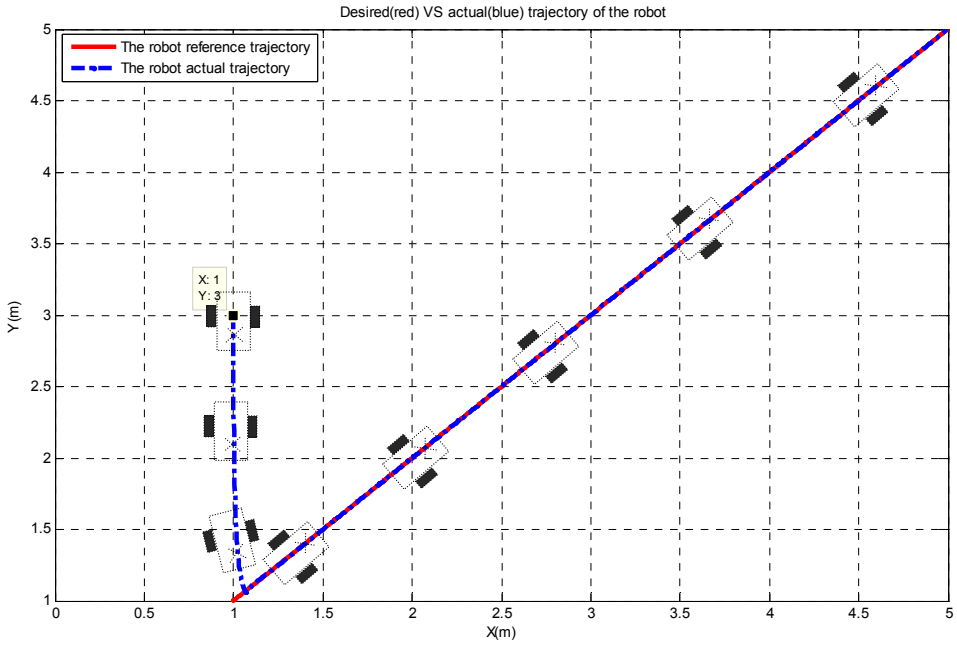

Figure 4-24: the robot actual and reference trajectory in x-y plane/Robot initial states(x, y, θ): (1, 3, and 3pi/2)

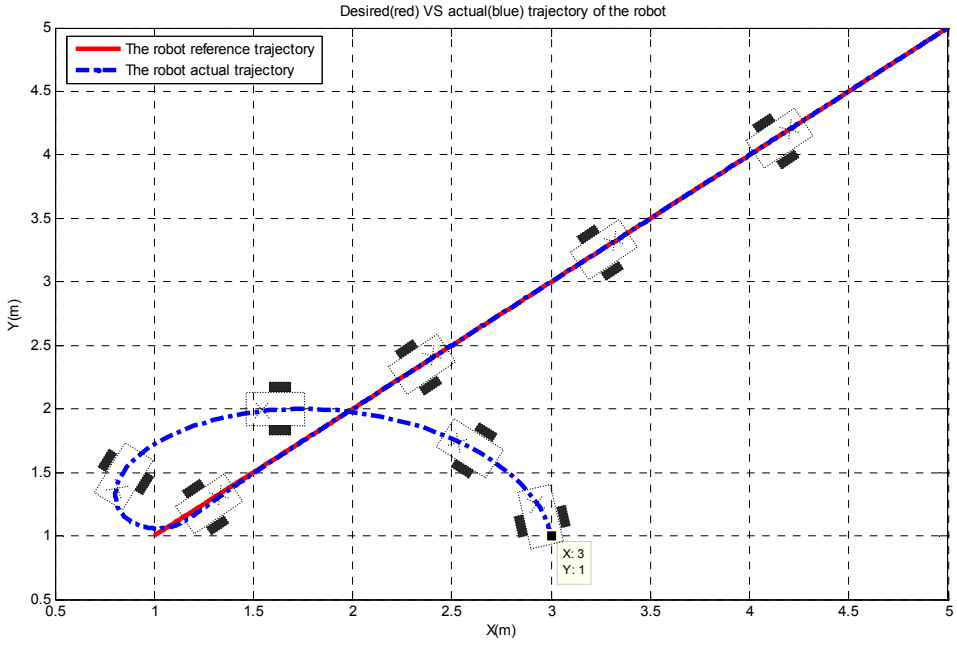

Figure 4-25: the robot actual and reference trajectory in x-y plane/Robot initial states(x, y, θ): (3, 1, and pi/2)

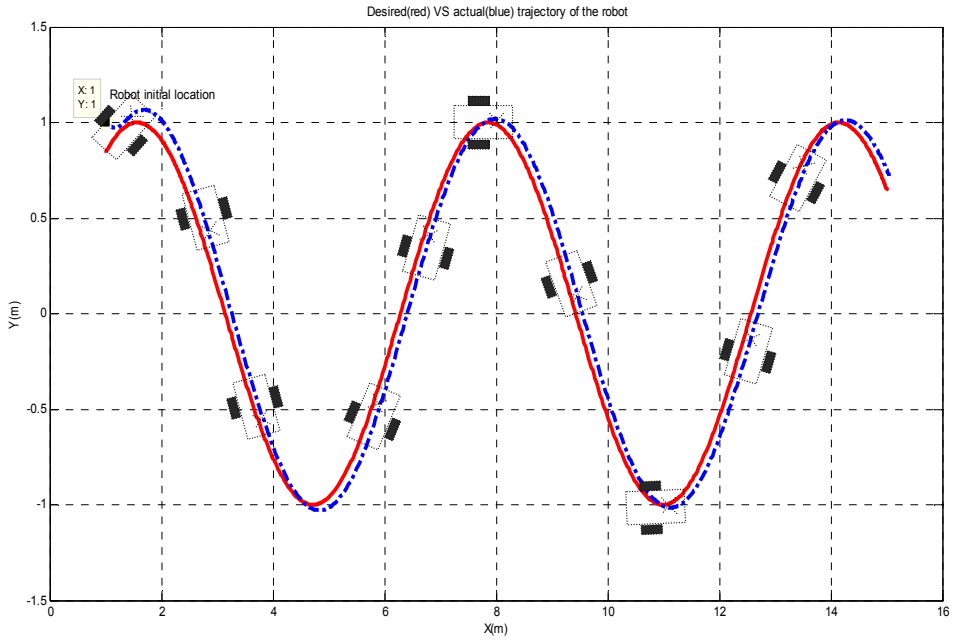

Figure 4-26: the robot actual and reference trajectory in x-y plane/Robot initial states(x, y, θ): (1, 1, and 0) sinosoidal reference traject0ry

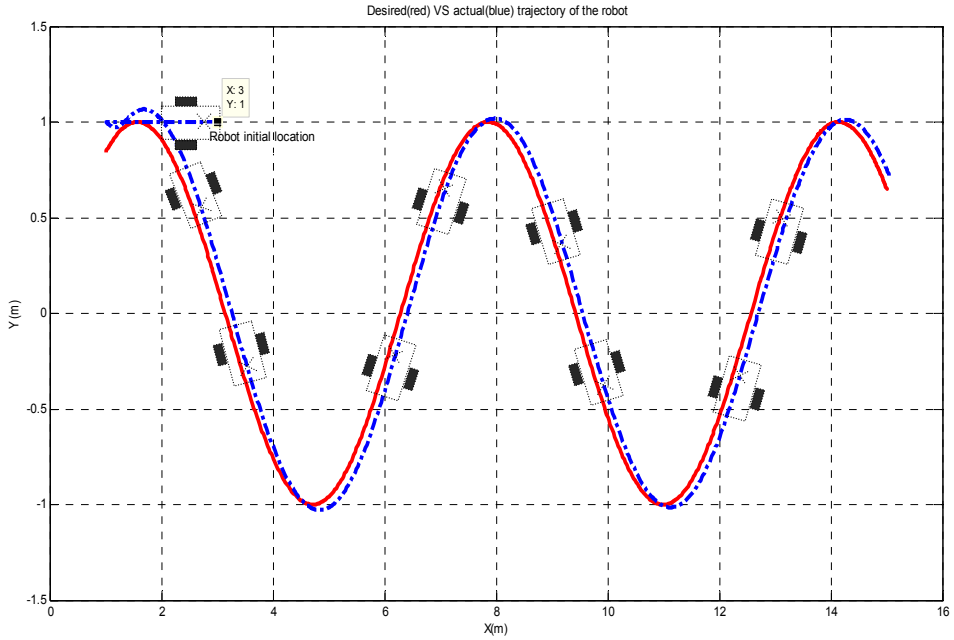

Figure 4-27: the robot actual and reference trajectory in x-y plane/Robot initial states(x, y, θ): (3, 1, and 0) sinusoidal reference traject0ry

As it can be seen from the above the system with the backstepping controller has a smooth response and is asymptotically stable regardless of the initial conditions and the type of the reference trajectory. The drawbacks of the backstepping controller are as follows:

- The controller needs a precise knowledge of the system dynamics and cants deal with model uncertainties.
- The controller cannot deal with unmodeled unstructured disturbances.
- The controller needs a large amount of gain tuning for each reference trajectory

The effect of the model uncertainties and disturbances such as wheel friction will be discussed in the next section.

4.2.3 The effect of model uncertainties and disturbances

In the previous section, we considered the trajectory tracking control of the mobile robot with the assumption of having a precise model without disturbance. Model imprecision may come from actual uncertainty about the plant for example unknown plant parameters or from purposeful choice of simplified representation of the systems dynamics like modeling friction as linear or neglecting structural modes in a reasonably rigid mechanical system. From a control point of view, modeling inaccuracies can be classified into two major kinds:

- Structured of parametric uncertainties
- Unstructured uncertainties or unmodeled dynamics

The first kind corresponds to inaccuracies on the terms actually included in the model, while the second kind corresponds to inaccuracies (or underestimation of) the system order.

Both kinds of modeling inaccuracies exist in the modeling of the nonholonomic mobile robot system. We have structured uncertainty because we don't exactly know all the system parameters and their precise values. The unstructured and unmodeled uncertainties are because of the friction between the ground and the robot wheels. We want to analyze the effect of these two kinds of uncertainties in the system and see how crucial their effect is on the system tracking performance.

First we analyze the effect of having friction in the system which is considered in the category of the unmodeled dynamics. Figures 4-28 and 4-29 show a typical mobile robot trajectory tracking with the backstepping controller without any friction disturbance and uncertainty:

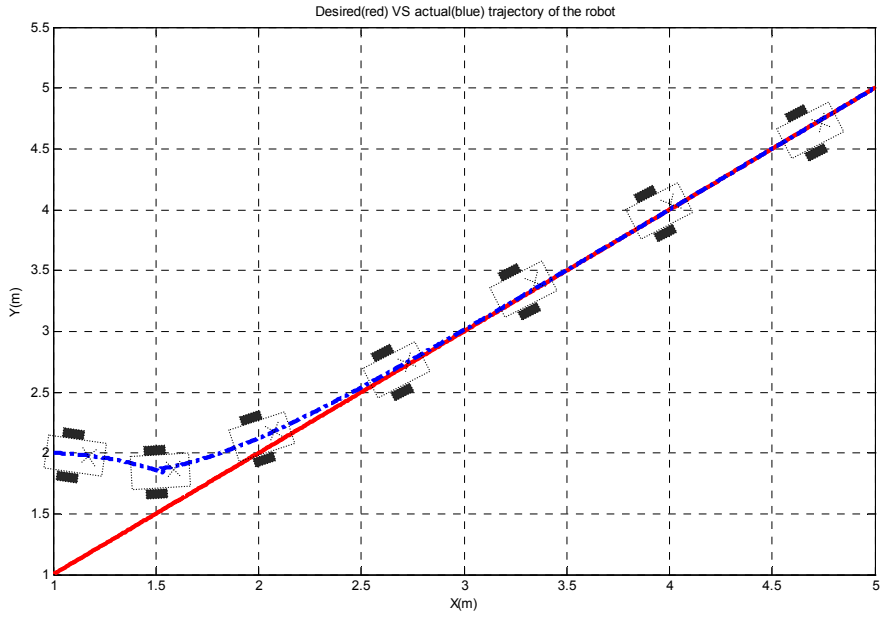

Figure 4-28: mobile robot trajectory tracking without friction and uncertainty

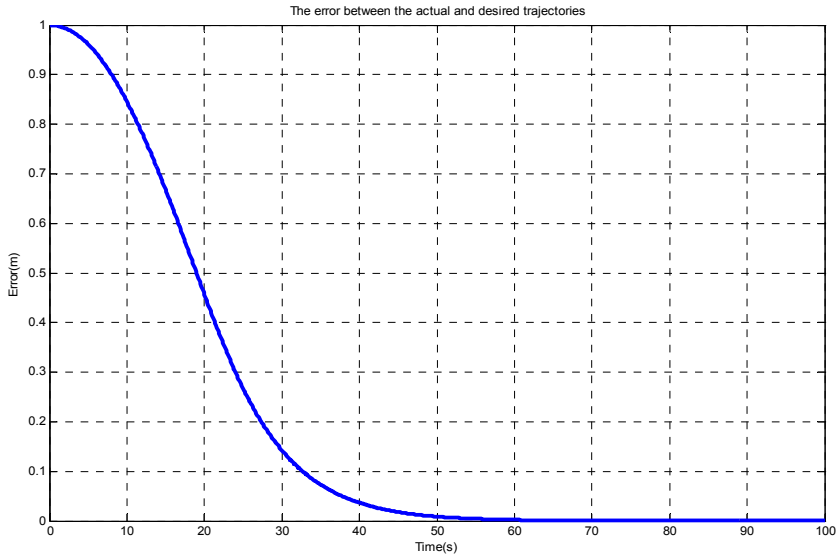

Figure 4-29: mobile robot trajectory tracking error without friction and uncertainty

In order to analyze the effect of the friction in the system, we add two different friction torques which are proportional to the kinetic friction coefficient between the robot wheel and ground, to the system and see the effect. Figures 4-30 and 4-31 show the robot trajectory tracking in existence of a 0.5 N.m friction torque for the right wheel and 0.1 N.m friction torque for the left wheel:

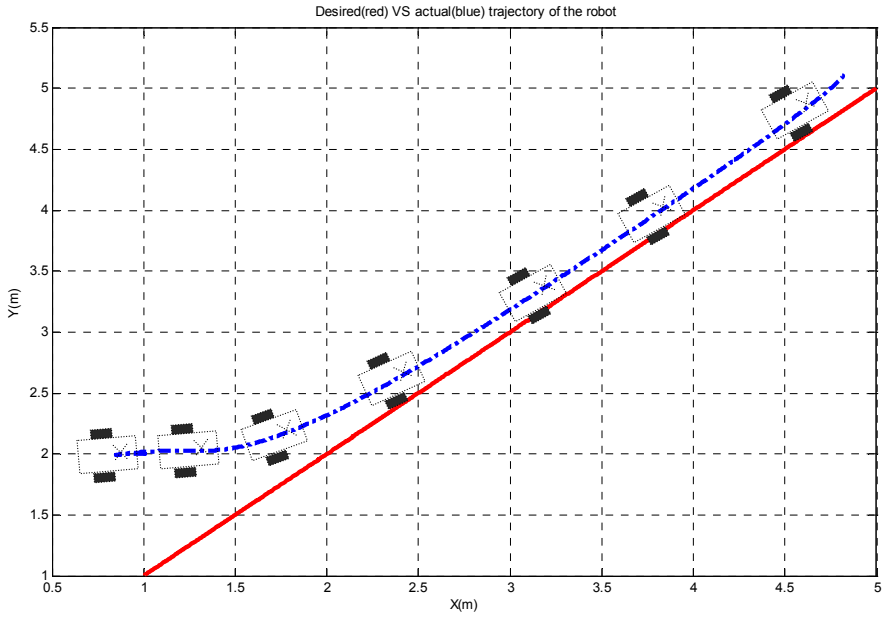

Figure 4-30: mobile robot trajectory tracking with 0.5 N.m right wheel friction torque and 0.1 N.m left wheel friction torque

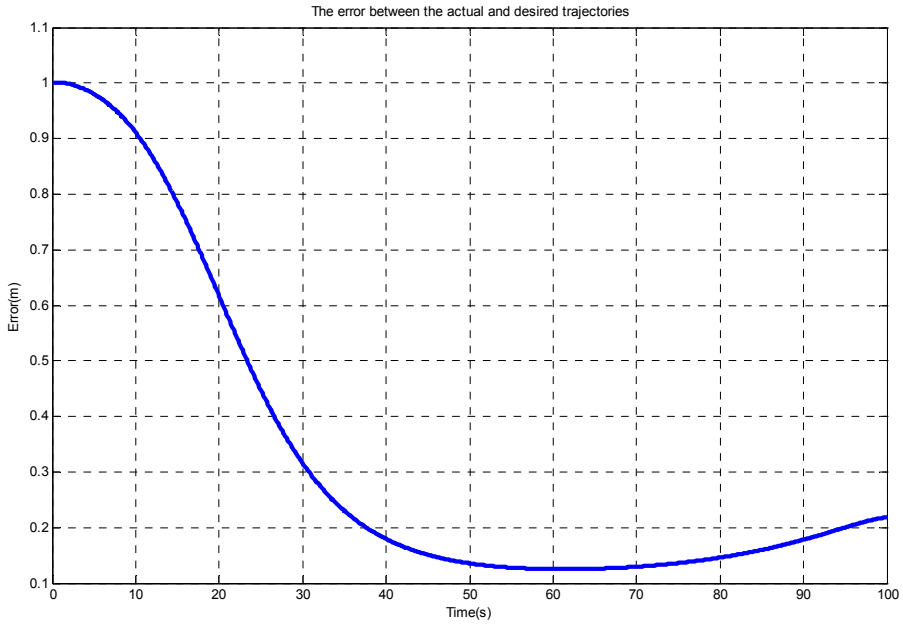

Figure 4-31: mobile robot trajectory tracking error with 0.5 N.m right wheel friction torque and 0.1 N.m left wheel friction torque

Figures 4-32 and 4-33 show the robot trajectory tracking in existence of a 0.5 N.m friction torque for the left wheel and 0.1 N.m friction torque for the right wheel:

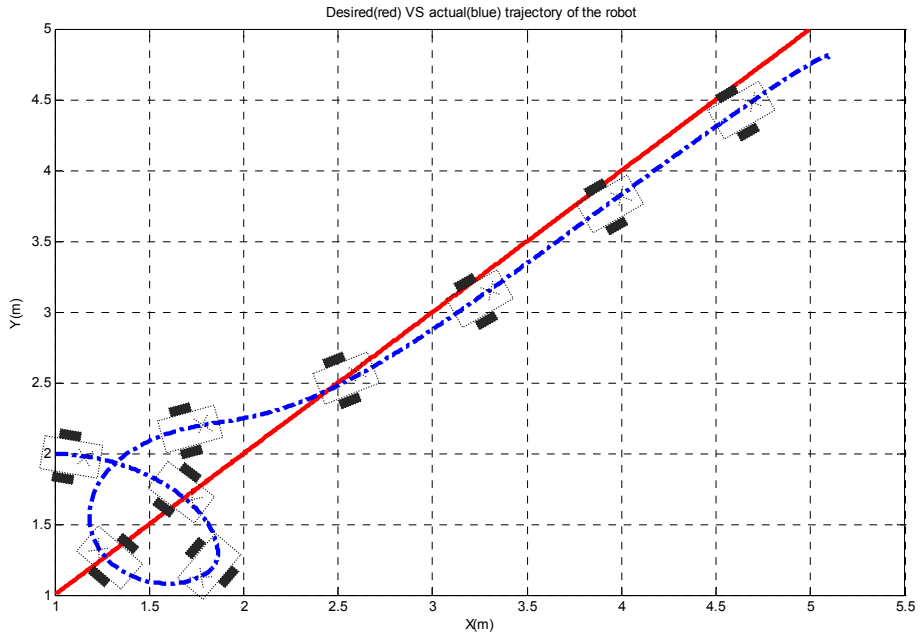

Figure 4-32: mobile robot trajectory tracking with 0.5 N.m left wheel friction torque and 0.1 N.m right wheel friction torque

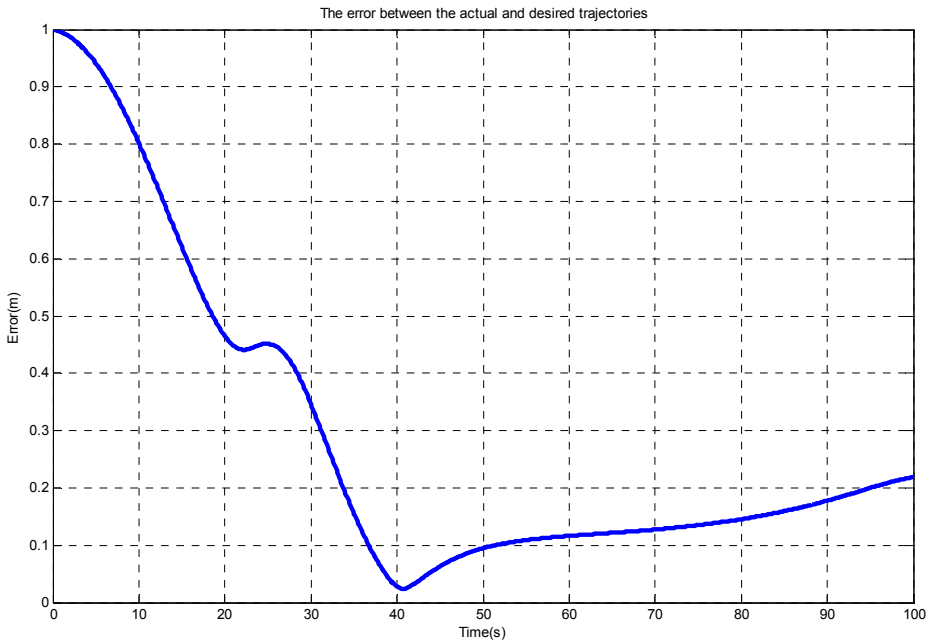

Figure 4-33: mobile robot trajectory tracking error with 0.5 N.m left wheel friction torque and 0.1 N.m right wheel friction torque

As it can be seen from the above four figures, the robot cannot have the desired trajectory tracking performance in existence of friction between the wheels and ground. The friction in the real application is not constant and cannot be modeled if the robot is supposed to work in any kind of work environment. Two major complementary approaches to deal with this kind of model uncertainty are *robust control* and *Adaptive control.* The Neural Network controller proposed in the next section is considered a strong adaptive control method that will deal with unmodeled uncertainties and disturbances using its learning and function approximation abilities. An introduction on NN control theory, its application in the case of mobile robot trajectory tracking control and the details of the proposed Neural-Backstepping controller will be discussed in the next section.

4.3 THE NEURAL NETWORK CONTROLLER DESIGN

Before we start to explain the details of the design of a neural network controller for the trajectory tracking of this mobile robot system, a brief background on neural networks which includes different NN topologies, properties, weight selection and training and NN application in control problems is presented in the next section.

4.3.1 Background on neural networks

4.3.1.1 Neural network topologies and overview

Artificial NN are modeled on biological process for information processing, including specifically the nervous system and its basic unit, the neuron. Signals are propagated in the form of potential differences between the inside and outside of cells. The mathematical model of a neuron is depicted in Figure 4-34:

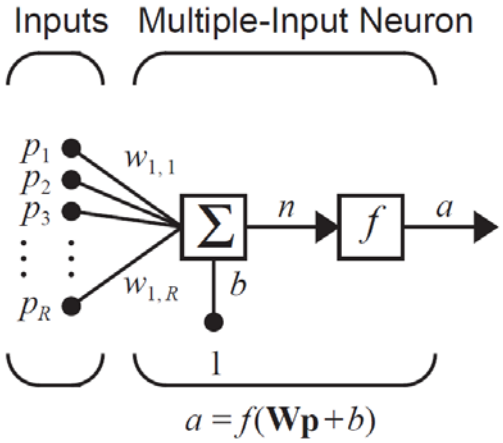

Figure 4-34: Mathematical model of a neuron

The input weights w_j, the firing threshold b (also called the bias), the summation of the weighted inputs and the nonlinear activation function f are shown in the above figure. If

the cell inputs are *n* signals at the time instant k, $x_1(k), x_2(k), x_3(k), \dots x_n(k)$ and the output is the scalar *y (k)*, the mathematical equation of the neuron can be written as follows:

$$y(k) = f(\sum_{j=1}^{n} w_j x_j(k) + b) \tag{4.48}$$

The activation functions are selected specific to the application though some common choices are as follows:

$$Linear\ threshold: \quad y = x \tag{4.49}$$

$$sigmoid\ (logistic\ curve): \quad y = \frac{1}{1 + e^{-x}} \tag{4.50}$$

$$Symmetric\ sigmoid: \quad y = \frac{1 - e^{-x}}{1 + e^{-x}} \tag{4.51}$$

$$Hyperbolic\ tangent: \quad y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4.52}$$

The intent of the activation function is to model the nonlinear behavior of the cell where there is no output below a certain values of the argument. Sigmoid functions are a general class of monotonically nondecreasing functions taking on bounded values. For many NN training algorithms including back-propagation algorithm, the derivative of the activation function is needed, so the selected activation function should be differentiable. Figure 4-35 shows a one layer NN consisting of s cells:

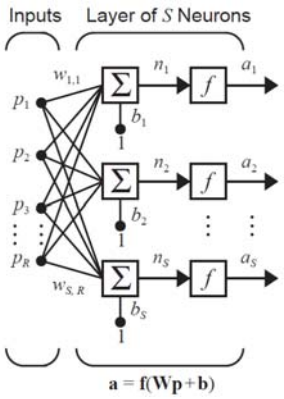

Figure 4-35: One-layer neural network with s neurons

The equation for this network is given by:

$$y_l(k) = f(\sum_{j=1}^{n} w_{lj} x_j(k) + b_l) \tag{4.53}$$

It is convenient to write the weights and threshold in matrix and vector forms. By defining the matrix of weights and thresholds as

$$W^T = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ w_{L1} & w_{L2} & \cdots & w_{Ln} \end{bmatrix}$$
$$b_w = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_L \end{bmatrix} \tag{4.54}$$

We can write the output vector $y(t) = [y_0 \quad y_1 \quad y_2 \quad \cdots \quad y_n]^T$ as:

$$y(t) = f(W^T x + b_w) \tag{4.55}$$

A two layer NN which has two layers of neurons with one layer having *L* neurons feeding the second one having *m* neurons, is shown in Figure 4-36:

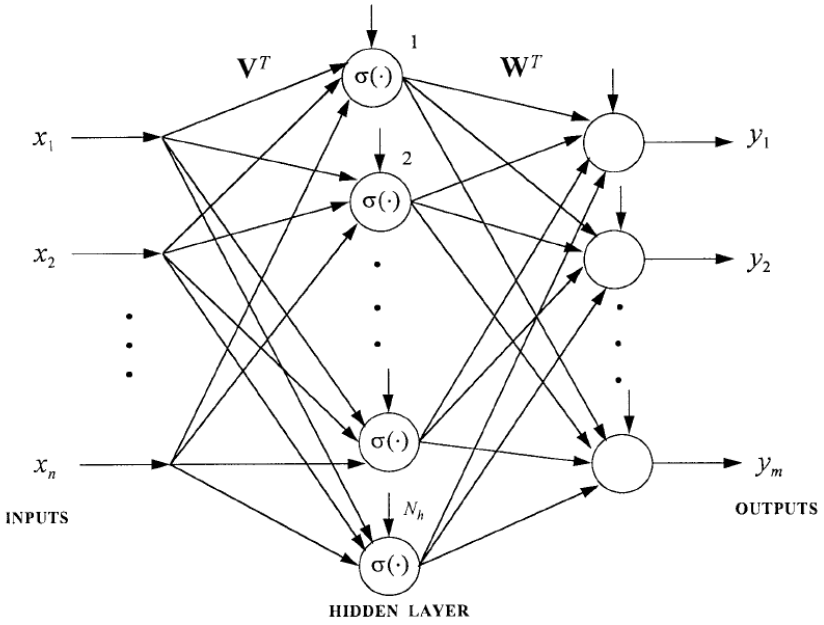

Figure 4-36: a two layer neural network

The first layer is called the hidden layer and the second layer is called the output layer. A neural network with multiple layers is called a multilayer perceptron and its computational power is enhanced significantly over the one layer network. The output of the above two layer network is given by the following equation:

$$y_i = \sigma(\sum_{l=1}^{L} w_{il}\sigma(\sum_{j=1}^{n} v_{lj}x_j + v_{l0}) + w_{i0}) \tag{4.56}$$

If we define the output of the hidden layer as z_l then we can write the above equation as follows:

$$z_l = \sigma(\sum_{j=1}^{n} v_{lj}x_j + v_{l0}) \quad l = 1,2,\dots,L \tag{4.57}$$

$$y_i = \sigma(\sum_{l=1}^{L} w_{il}z_l + w_{i0}) \quad i = 1,2,\dots,m \tag{4.58}$$

The above two equations can be written in the matrix form like equations (4.54) and (4.55). It is important to mention that the input to hidden layer weights will be selected randomly while the hidden to output layer weights will be tuned using appropriate training methods. This will minimize the computational complexity of using NN in the control applications.

4.3.1.2 Neural network properties

Neural networks are complex nonlinear distributed systems, and as a result they have a wide range of applications. Two of the most important properties of the neural networks are: classification (for pattern recognition) and function approximation. One fundamental in NN closed-loop control applications is the universal function approximation property of NN having at least two layers. (One layer NNs do not have a universal function approximation ability).

The basic universal approximation theory says that any smooth function *f(x)* can be approximated arbitrarily closely on a compact set using a two layer NN with appropriate weights. Specifically, let *f(x)* be a general smooth function. Then given a compact set *S* and a positive number ϵ_N, there exists a two layer NN such that:

$$f(x) = W^T\sigma(V^Tx) + \epsilon \tag{4.59}$$

With $\|\epsilon\| = \epsilon_N$ for all x in the compact set S, for some sufficiently large value L of hidden layer neurons. The value ϵ is called the NN function approximation error and it decreases as the number of hidden layer neurons L increase. On the other hand, on the compact set S, as S becomes larger, the required L generally increases correspondingly. The neural network acting as a function approximator is shown in figure 4-37:

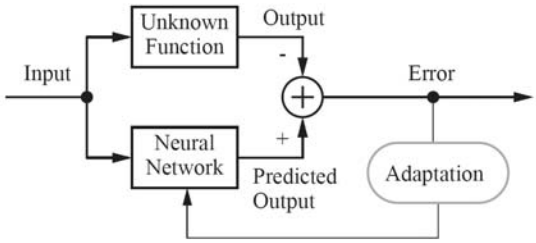

Figure 4-37: the function approximation structure for neural networks

Even though the above theorem says that there exists an NN that approximates any function *f(x)*, it should be noted that it does not show how to determine the required weights. It is in fact not an easy task to determine the weights so that an NN does indeed approximate a given function *f(x)* closely enough. In the next section we shall show how to accomplish this using backpropagation tuning.

4.3.1.3 Neural network training

As it is mentioned in the previous section, for an NN to function as desired, it is necessary to determine suitable weights and thresholds that ensure performance as desired. For years this was a problem specifically for a multilayer NN, where it was not known to relate the portion of the error to the weights of the different layers. Today these problems have for the most part been solved and there are very good algorithms for NN weight selection and tuning. The backpropagation training algorithm is one of the famous algorithms because of its simplicity and power. The derivation of this method is based on the gradient decent method which is not included in this text. The backpropagation algorithm for a two layer neural network shown in Figure 4-38 using sigmoid activation function is explained as follows:

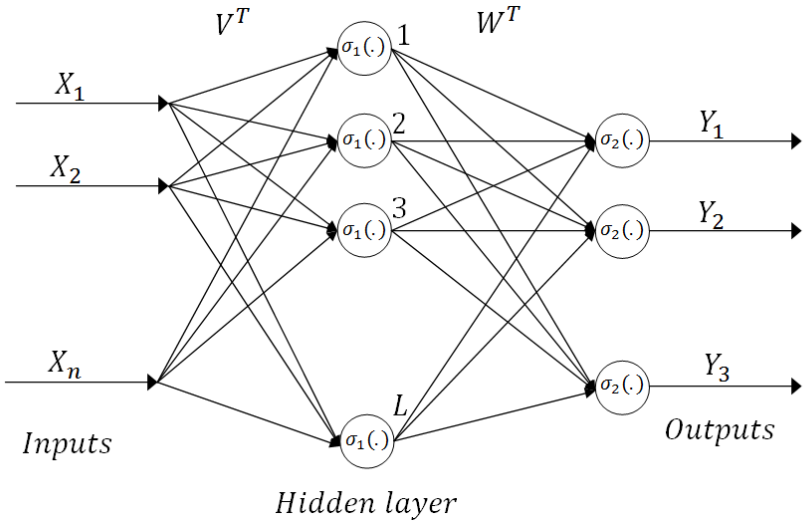

Figure 4-38: two layer neural network

The output of the above two layer neural network is given by the following equation:

$$y_i = \sigma\left(\sum_{l=1}^{L} W_{il}\sigma\left(\sum_{j=1}^{n} V_{lj}x_j + v_{l0}\right) + w_{i0}\right) \quad i = 1,2,\dots,m \tag{4.60}$$

The derivation is greatly simplified by defining some intermediate parameters.
In the figure, we can call the layer of weights V_{lj} the first layer and the layer of weights W_{il} the second layer. The input to layer one is x_j. Define the input to layer two as:

$$z_l = \sigma\left(\sum_{j=1}^{n} V_{lj}x_j + v_{l0}\right) \quad l = 1,2,\dots,L \tag{4.61}$$

The thresholds can more easily be dealt with by defining $x_0 = 1$ and $z_0 = 1$. Then one can say:

$$y_i = \sigma\left(\sum_{l=1}^{L} W_{il}\, z_l\right) \tag{4.62}$$

$$z_l = \sigma\left(\sum_{j=1}^{n} V_{lj}x_j\right) \tag{4.63}$$

Define the output of layers two and one, respectively as:

$$u_i^2 = \sum_{l=0}^{L} W_{il}\, z_l \tag{4.64}$$

$$u_l^1 = \sum_{j=0}^{n} V_{lj}x_j \tag{4.65}$$

Then we can write:

$$y_i = \sigma\left(u_i^2\right) \tag{4.66}$$

$$z_l = \sigma\left(u_l^1\right) \tag{4.67}$$

In deriving the backpropagation algorithm we shall have the occasion to differentiate the activation functions. Note therefore that:

$$\frac{\partial y_i}{\partial W_{il}} = \acute{\sigma}(u_i^2) z_l \tag{4.68}$$

$$\frac{\partial y_i}{\partial z_l} = \acute{\sigma}(u_i^2) W_{il} \tag{4.69}$$

$$\frac{\partial z_l}{\partial V_{lj}} = \acute{\sigma}(u_l^1) x_j \tag{4.70}$$

$$\frac{\partial z_l}{\partial x_j} = \acute{\sigma}(u_l^1) V_{lj} \tag{4.71}$$

Where $\acute{\sigma}()$ is the derivative of the activation function. The backpropagation algorithm is a weight tuning algorithm based on the gradient descent method:

$$W_{il}(k+1) = W_{il}(k) - \eta \frac{\partial E(k)}{\partial W_{il}} \tag{4.72}$$

$$V_{lj}(k+1) = V_{lj}(k) - \eta \frac{\partial E(k)}{\partial V_{lj}} \tag{4.73}$$

Let there be prescribed an input vector X and an associated desired output vector Y for the network. Define the lease square NN output error as:

$$E(k) = \frac{1}{2} e^T(k) e(k) = \frac{1}{2} \sum_{l=1}^{m} e_l(k)^2 \tag{4.74}$$

$$e_i(k) = Y_i(k) - y_i(k) \tag{4.75}$$

Where $y_i(k)$ is calculated using the forward propagation equation for the neural network with the components of the input pattern X_j as the NN inputs $x_i(k)$.

The required gradients of the cost function $E(k)$ with respect to the weights can easily be determined using chain rule. Specifically for the second layer gains we have:

$$\frac{\partial E}{\partial W_{il}} = \frac{\partial E}{\partial u_i^2} \times \frac{\partial u_i^2}{\partial W_{il}} = \left[\frac{\partial E}{\partial e_i} \times \frac{\partial e_i}{\partial y_i} \times \frac{\partial y_i}{\partial u_i^2}\right] \times \frac{\partial u_i^2}{\partial W_{il}} \tag{4.76}$$

Using the above equations we have:

$$\frac{\partial E}{\partial u_i^2} = -1 \times \acute{\sigma}\left(u_i^2\right) \times e_i = -\acute{\sigma}\left(u_i^2\right) e_i \tag{4.77}$$

$$\frac{\partial E}{\partial W_{il}} = e_i \times -1 \times \acute{\sigma}\left(u_i^2\right) \times z_l = -z_l \acute{\sigma}\left(u_i^2\right) e_i \tag{4.78}$$

Similarly for the first layer weights we have:

$$\frac{\partial E}{\partial V_{lj}} = \frac{\partial E}{\partial u_l^1} \times \frac{\partial u_l^1}{\partial V_{lj}} = \left[\sum_{i=1}^{m} \frac{\partial E}{\partial u_i^2} \times \frac{\partial u_i^2}{\partial z_l} \times \frac{\partial z_l}{\partial u_l^1}\right] \times \frac{\partial u_l^1}{\partial V_{lj}} \tag{4.79}$$

Therefore we have:

$$\frac{\partial E}{\partial V_{lj}} = \left[\sum_{i=1}^{m} \left(-\acute{\sigma}\left(u_i^2\right) e_i\right) \times W_{il} \times \acute{\sigma}\left(u_l^1\right)\right] \times x_j \tag{4.80}$$

$$\frac{\partial E}{\partial V_{lj}} = -x_j \acute{\sigma}\left(u_l^1\right) \sum_{i=1}^{m} \left(-\acute{\sigma}\left(u_i^2\right) e_i\right) \times W_{il} \tag{4.81}$$

The above equations can be considerably simplified by introducing the notation of a backward recursion through the network. Thus define the back propagated error for layers two and one, respectively as:

$$\delta_i^2 = -\frac{\partial E}{\partial u_i^2} = \acute{\sigma}\left(u_i^2\right) e_i \tag{4.82}$$

$$\delta_l^1 = -\frac{\partial E}{\partial u_l^1} = \acute{\sigma}\left(u_l^1\right) \sum_{i=1}^{m} \delta_i^2 \times W_{il} \tag{4.83}$$

Assuming the sigmoid activation functions used the back propagated errors can be computed as:

$$\delta_i^2 = y_i(1 - y_i)e_i \tag{4.84}$$

$$\delta_l^1 = z_l(1 - z_l)\sum_{i=1}^{m} \delta_i^2 \times W_{il} \tag{4.85}$$

Combining the above equations, one can write the backpropagation algorithm formulas as follows:

$$W_{il}(k+1) = W_{il}(k) - \eta z_l \delta_i^2 \tag{4.86}$$

$$V_{lj}(k+1) = V_{lj}(k) - \eta x_j \delta_l^1 \tag{4.87}$$

In many applications the NN has no activation function in the output layer. Then one must use simply $\delta_i^2 = e_i$ in the equations for backpropagation.

The back propagation general formulas and procedures are shown in the diagram of Figure 4-39:

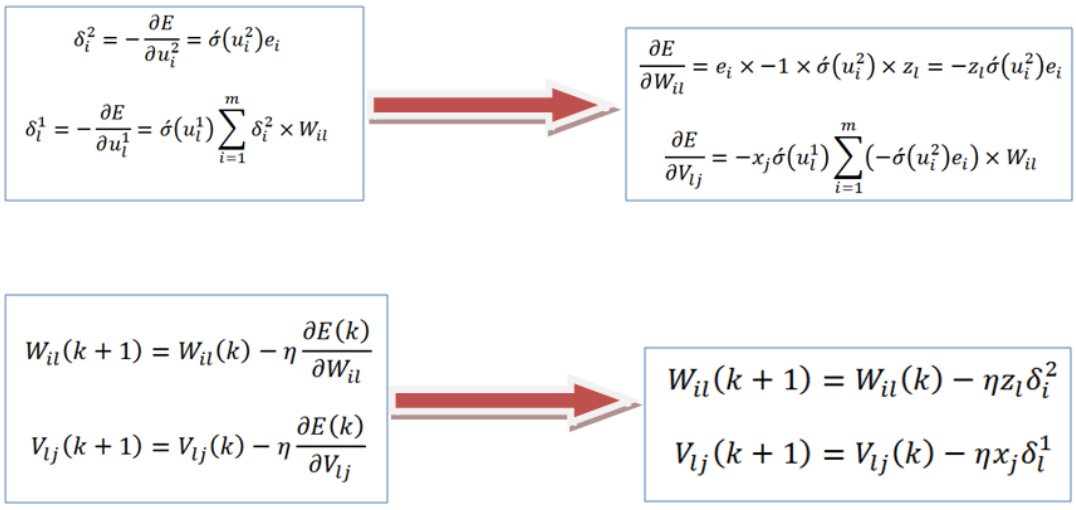

Figure 4-39: the backpropagation algorithm block diagram

An improved version of the above algorithm which is based on the gradient decent method is given by the momentum gradient algorithm:

$$V(k+1) = \beta V(k) + \eta(1 - \beta)Xe^T(k) \tag{4.88}$$

With the positive momentum parameter $\beta < 1$ and a positive learning rate < 1 . β is generally selected near one for example 0.95. Momentum adds a memory effect so that the NN in effect responds not only to the local gradient, but also to the recent trends in the error surface. Without momentum, the network may get stuck in the local minima but adding momentum can help the NN ride through the local minima. The above algorithm is the method we use to train the required neural network in the application of the mobile robot trajectory tracking controller.

4.3.1.4 Neural network development and implementation

In order to construct a neural network for the approximation of a general multi input multi output function, one should have an algorithm to initialize the neural network, use the backpropagation algorithm to train the network and use the approximation error of the neural network to show its performance. The algorithm used to construct the neural network and implement it through a suitable software package for function approximation is shown Figure 4-40:

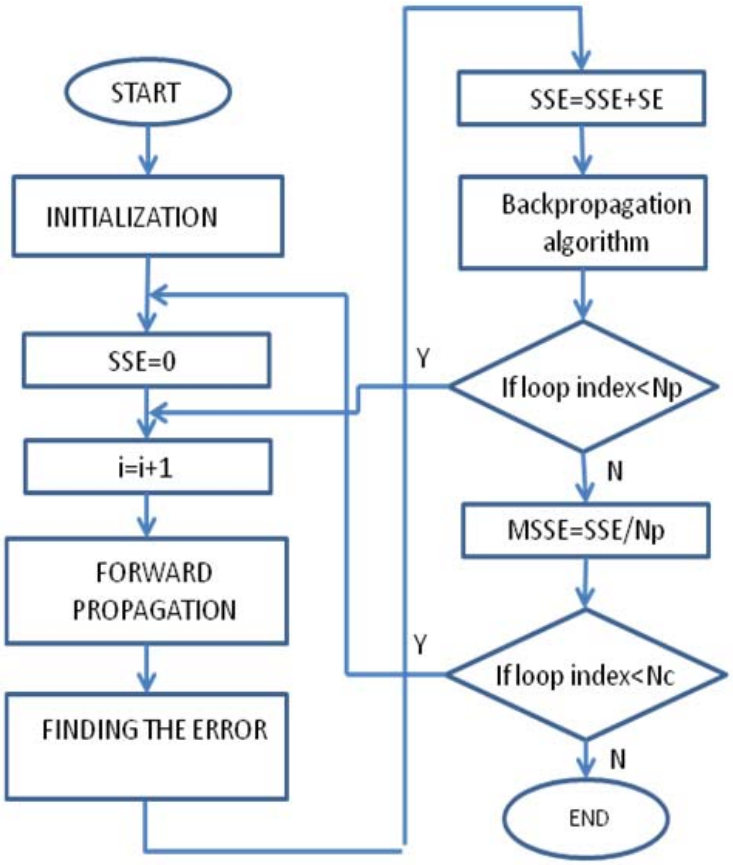

Figure 4-40: The general algorithm for implementation of neural network training using backpropagation

Different blocks in the above flow chart are responsible for a specific task which will be explained in the remaining part of this section. A typical two layer neural network with two inputs and one output which uses the above algorithm for its training is shown Figure 4-41:

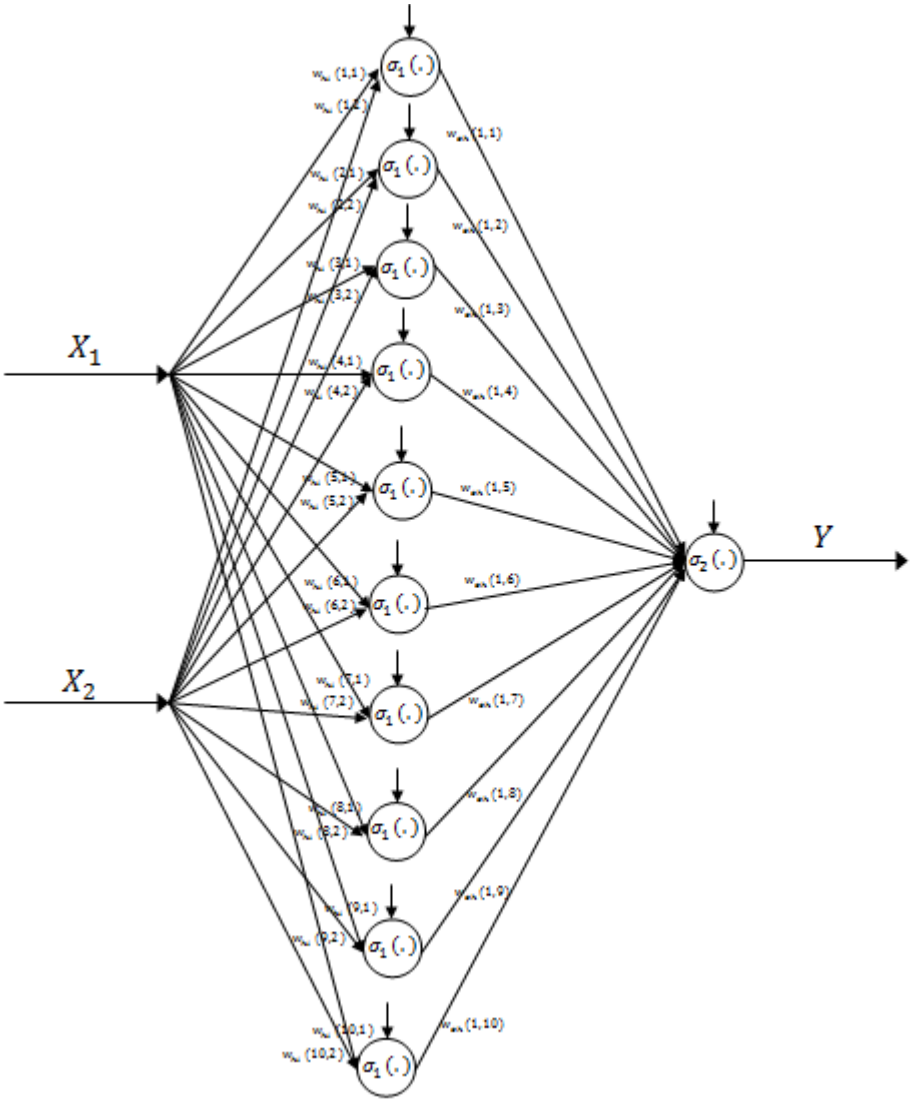

Figure 4-41: a two layer neural network with two inputs and on output

As it can be seen from figure (4-40), the first part of the algorithm is to initialize the network which means to define the number of inputs, number of outputs, and number of neurons in the hidden layer and the learning and momentum rates. The following terminology is used in this work to define different parameters of the implemented neural networks:

N_i: the number of inputs to the neural network

N_o: the number of outputs of the neural network

N_h: the number of neurons in the hidden layer

η: the learning rate

β: the momentum rate

N_p: the number input data patterns

N_C: the number cycles to train the neural network

Whi: the hidden layer to input layer weights

Woh: the output layer to hidden layer weights

The next part of the network initialization is to define the activation function to be used for the hidden layer and output layer functions. The next step is to construct the learning and training part which consists of the following parts:

- A loop for the number of cycles we want to repeat the same set of input data to train the network and reach the desirable error. The number of repetitions of this loop is N_c and the loop index is j.
- A loop for the number of patterns of data we have in each cycle. This loop will read all of the data in one cycle. The number of repetitions of this loop is N_p and the loop index is i.

As it can be seen from the flow chart of figure (4-40), the loop which reads the data patterns in each cycle is inside another loop which repeats the whole data for a certain number of cycles. The first thing we need to do before performing the back-propagation for each data pattern is to do the forward propagation and find the error between the neural network output and the actual function. The forward propagation is done using the following equations:

$$y_i = \sigma(\sum_{l=1}^{L} woh_{il}\sigma(\sum_{j=1}^{n} whi_{lj}x_j + v_{l0}) + w_{i0}) \tag{4.89}$$

$$z_l = \sigma(\sum_{j=1}^{n} whi_{lj}x_j + v_{l0}) \quad l = 1,2,\dots,L \tag{4.90}$$

$$y_i = \sigma(\sum_{l=1}^{L} woh_{il} z_l + w_{i0}) \quad i = 1,2,\dots,m \tag{4.91}$$

The above equations are for the general case of having i outputs, L hidden layer neurons and n inputs. For the case of having one output, two inputs and 10 hidden layer neurons, for examples, the forward propagation equations will be as follows:

$$\begin{gathered} z_l = \sigma(whi_{l1} x_1 + whi_{l1} x_1 + v_{l0}) \quad l = 1,2,\dots,10 \\ Y = \sigma(\sum_{l=1}^{L} woh_l z_l + w_0) \end{gathered} \tag{4.92}$$

After performing the forward propagation, we can find the error for each input data pattern and find the sum of error squared according to the following equation:

$$SSE = \sum_{i=1}^{Np} {E_p}^2 \tag{4.93}$$

Np is the number of input data patterns in one cycle and E_p is the approximation error corresponding to each pattern. By the end of the data pattern loop, we have one SSE for one cycle of input data. The next step is to find the mean sum squared error based on the following equation:

$$MSSE = \frac{\sum_{i=1}^{Np} {E_p}^2}{Np} \tag{4.94}$$

By finishing the data cycles loop, the calculations for the entire cycles and patterns are done and we can check the performance of the network by looking at the MSSE curve and comparing the outputs of the neural network and the real function. The Matlab code which is developed to impellent the above algorithm is included in the appendix. An example of a two input one output function to be approximated using this algorithm is as follows:

Function to be approximated: $f(x_1, x_2) = \sin(pi * x_1)\cos(pi * x_2)$

The activation function: $f(x) = \frac{1-e^{-x}}{1+e^{-x}}\ log - sigmoid$

The number of hidden layers: $N_h = 10$

The learning rate: $\eta = 0.1$

The momentum rate: $\beta = 0.95$

The number of cycles: $N_C = 100$

The output of the neural network and the real output of the function is shown Figures 4-42 and 4-43:

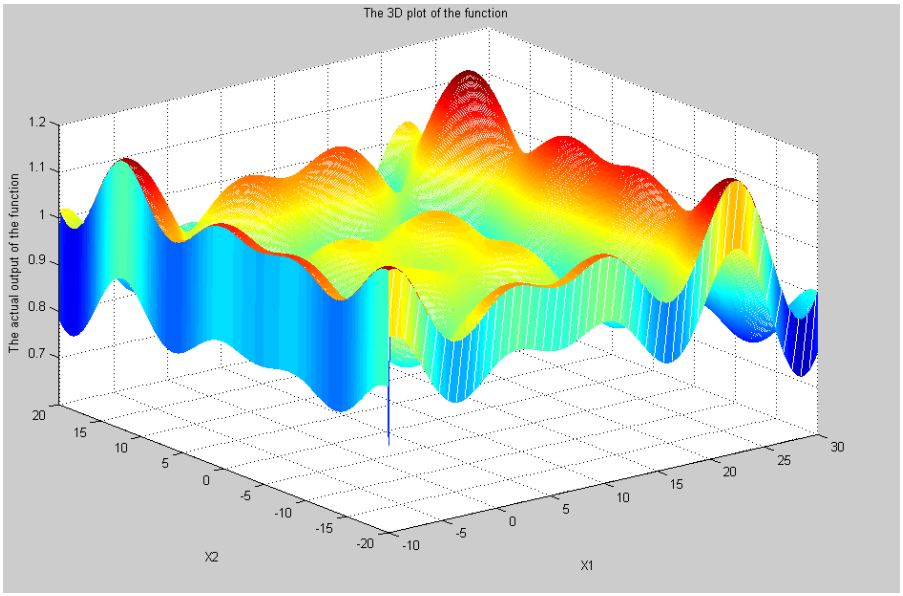

Figure 4-42: the real output of the function

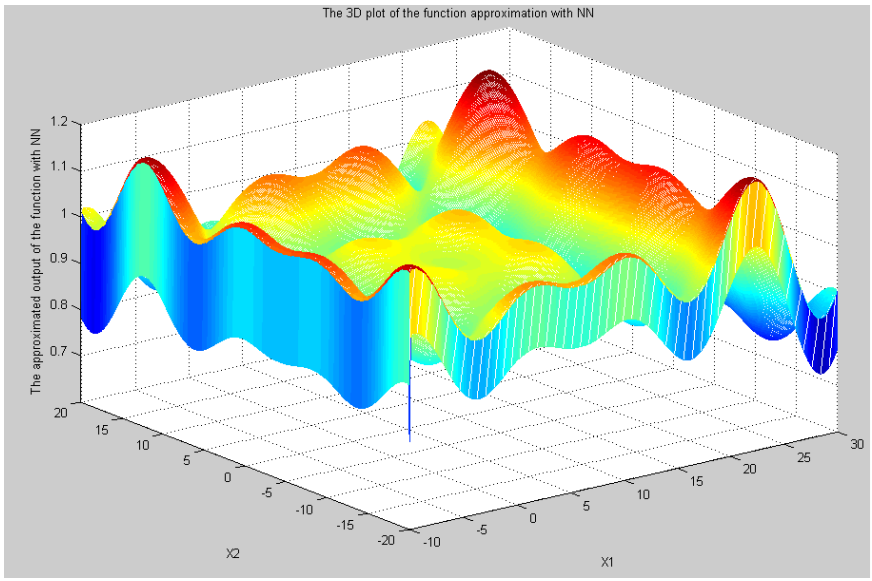

Figure 4-43: the neural network output

Perfect approximation performance of the algorithm can be seen from the above two figures. The above neural network implementation algorithm will be used in designing the controllers which need the approximation property of a neural network. Designing

trajectory tracking controllers using neural networks will be explained in detail in next sections of this chapter.

4.3.2 The NN inverse model trajectory tracking controller

A control structure that integrates the backstepping controller and a neural network computed torque controller for nonholonomic mobile robots is proposed in this section. A combined kinematic/torque control law is developed using backstepping and the stability is proved using Lyapunov approach. The NN controller proposed in this work can deal with unmodeled bounded disturbances and unstructured unmodeled dynamics in the vehicle. The trajectory tracking controller structure using neural networks inverse model is shown in Figure 4-44:

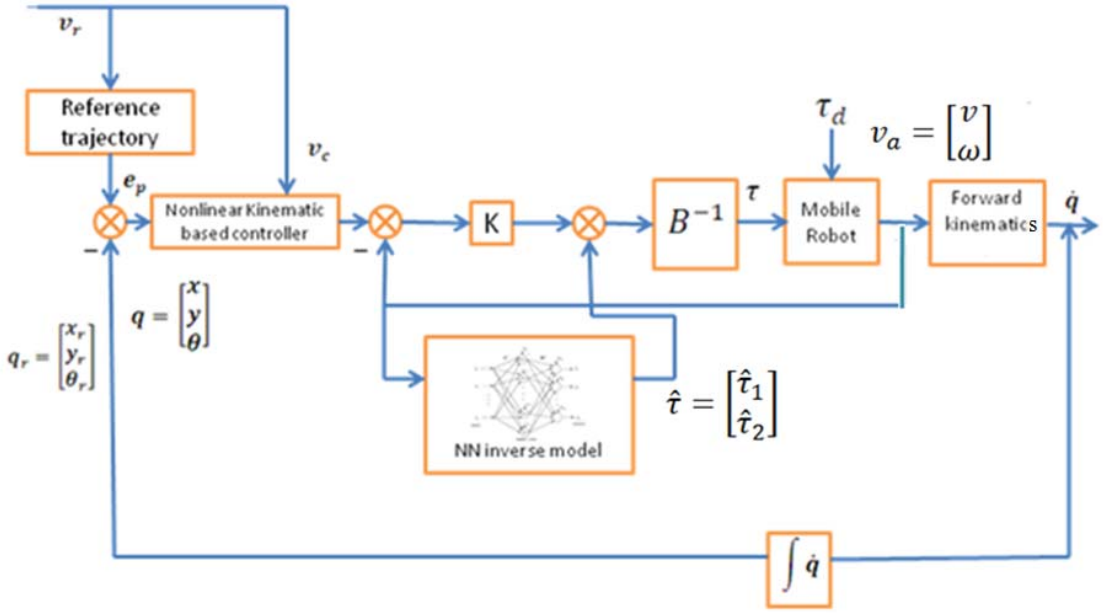

Figure 4-44: the neural network inverse model controller structure

The traditional backstepping controller structure is shown in Figure 4-45 again for the comparison purpose:

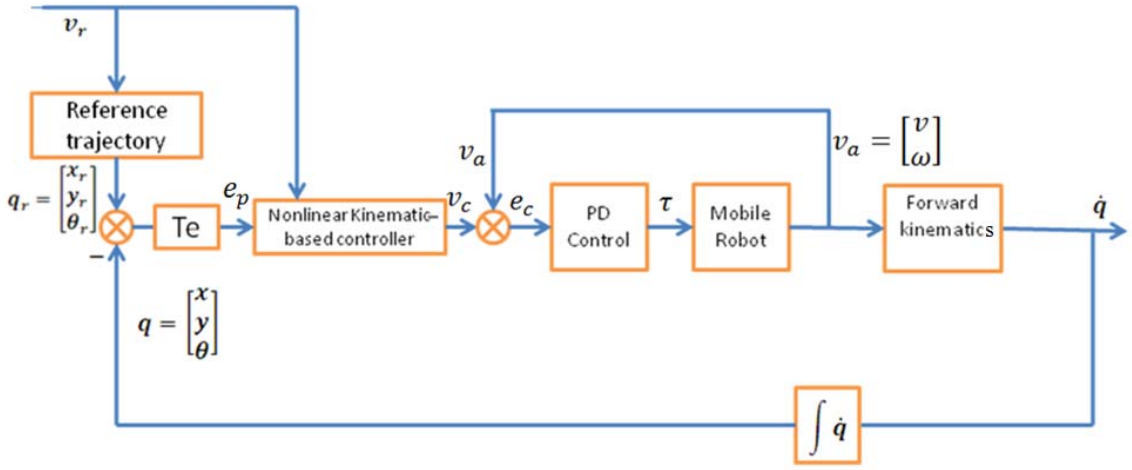

Figure 4-45: the backstepping controller structure

Comparing figures (4-44) and (4-45), one can find out that in the proposed neural network inverse model control structure, no knowledge about the dynamics of the robot is assumed and the function of the neural network is to learn the vehicle dynamics online and reconstruct the dynamic model of the system. Therefore, converting the kinematic based controller output $\boldsymbol{v_c}$ to a suitable torque τ for the robot despite having unknown dynamical parameters or bounded unknown disturbances, is done using the neural network.

Given the desired velocity $v_c(t)$ which is the output of the kinematic controller equation (4.7), the trajectory tracking error will be:

$$e_c = v_c - v \tag{4.89}$$

Differentiating the above equation and using the mobile robot dynamic equation (3.80):

$$\dot{e}_c = \dot{v}_c - \dot{v} \tag{4.90}$$

$$\bar{M}(q)\dot{v}(t) + \bar{V}_m(q,\dot{q})v(t) + \bar{F}(\dot{q}) + \bar{G}(q) + \bar{\tau}_d = \bar{B}(q)\tau \tag{4.91}$$

Multiplying equation (4.90) by $\bar{M}(q)$, we have:

$$\bar{M}(q)\dot{e}_c = \bar{M}(q)\dot{v}_c - \bar{M}(q)\dot{v} = \bar{M}(q)\dot{v}_c + \bar{V}_m(q,\dot{q})v(t) + \bar{F}(\dot{q}) + \bar{\tau}_d - \bar{B}(q)\tau \tag{4.92}$$

$$\bar{\mathrm{M}}(\mathrm{q})\dot{e}_c = -\bar{V}_m(q,\dot{q})e_c - \bar{B}(q)\tau + \bar{\tau}_d + f(v, v_c, \dot{v}_c) \tag{4.93}$$

Equation (4.93) is the mobile robot dynamic equation in terms of the velocity tracking error e_c. The function *f* is the function containing the mobile robot nonlinearities:

$$f(v,v_c,\dot{v}_c)=M(q)\dot{v}_c+V_m(q,\dot{q})v_c+F(\dot{q}) \tag{4.94}$$

As it can be seen from the above equation, f is a function of $v, v_c, \dot{v}_c$ which are all measurable. This function contains all the mobile robot parameters such as masse, moments of inertia and friction coefficient. In the experiment, this function is partially known, therefore a suitable control action for the robot can be written as follows:

$$\bar{\tau} = \hat{f} + K_4 e_c \tag{4.95}$$

Where K_4 is the same diagonal positive definite gains used in the backstepping controller (4.25) and $\hat{f}$ is an estimate of the robot nonlinear function f and is provided by the neural network inverse model. Substituting the above control rule in the equation (4.93) we have:

$$\overline{\mathrm{M}}(\mathrm{q})\dot{e}_c = -\overline{V}_m(q,\dot{q})e_c - \hat{f} - K_4 e_c + \bar{\tau}_d + f(v, v_c, \dot{v}_c) \tag{4.96}$$

We can define the function approximation error as follows:

$$\tilde{f} = f - \hat{f} \tag{4.97}$$

Substituting equation (4.76) in (4.75) and simplifying it we have:

$$\overline{M}(q)\dot{e}_c = -(\overline{V}_m(q,\dot{q}) + K_4)e_c + \tilde{f} + \bar{\tau}_d \tag{4.98}$$

At this stage, the control design problem is how to design the gain matrix K_4 and the estimate $\tilde{f}$ so that both the error signal e_c and the control signal are bounded. In computing the estimate $\widehat{f}$, several methods including adaptive control can be used. Here we use the neural network function approximation property to do the estimate which can always be accomplished due to equation (4.59). The important issue here is that we should design the controller in a way that both the error signal $e_c(t)$ and the control signal stay bounded. The derivation of the inverse model neural network learning algorithm to make the error signal $e_c(t)$ zero will be explained in detail in the next section.

4.3.2.1 The NN inverse model learning algorithm derivation

The following definitions and terminology will be used to derive the learning algorithm of the neural network inverse model controller according to figure (1):

$$\tau = \begin{bmatrix}\tau_1 \\ \tau_2\end{bmatrix}, \quad \bar{\tau} = \begin{bmatrix}\bar{\tau}_1 \\ \bar{\tau}_2\end{bmatrix} \tag{4.99}$$

$$v_a = \begin{bmatrix} v \\ \omega \end{bmatrix} \quad , \hat{\tau} = \begin{bmatrix} \hat{\tau}_1 \\ \hat{\tau}_2 \end{bmatrix} \tag{4.100}$$

$$e_c = v_c - v = \begin{bmatrix} v_c \\ \omega_c \end{bmatrix} - \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} e_{cv} \\ e_{c\omega} \end{bmatrix} \tag{4.101}$$

The neural network topology that we want to use for the purpose of torque computation is shown in Figure 4-46:

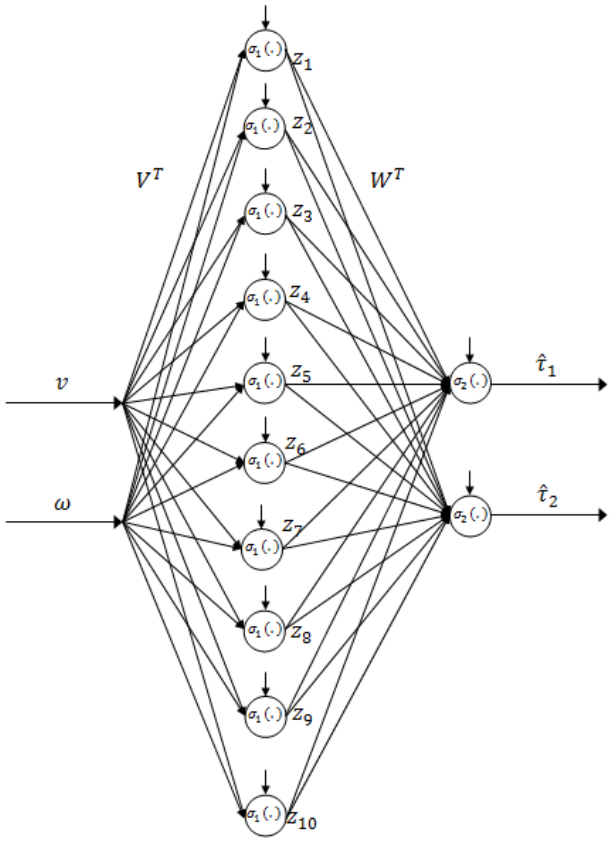

Figure 4-46: the inverse model neural network

The following performance criterion is chosen to make the error $e_c(t)$ zero:

$$E = \frac{1}{2}({e_{cv}}^2 + {e_{c\omega}}^2) \tag{4.102}$$

The derivative of this cost function with respect to the neural network gains is:

$$\frac{\partial E}{\partial W} = e_{cv}\frac{\partial e_{cv}}{\partial W} + e_{c\omega}\frac{\partial e_{c\omega}}{\partial W} = {e_C}^T\frac{\partial e_C}{\partial W} \tag{4.103}$$

$$\frac{\partial E}{\partial V} = e_{cv}\frac{\partial e_{cv}}{\partial V} + e_{c\omega}\frac{\partial e_{c\omega}}{\partial V} = {e_C}^T\frac{\partial e_C}{\partial V} \tag{4.104}$$

The backpropagation algorithm will be used to learning structure of this neural network:

$$W(k+1) = W(k) - \eta\frac{\partial E(k)}{\partial W} \tag{4.105}$$

$$V(k+1) = V(k) - \eta\frac{\partial E(k)}{\partial V} \tag{4.106}$$

Substituting the derivatives of equations (4.103) and (4.104) in the above back propagation algorithm, we have:

$$W(k+1) = W(k) - \eta({e_C}^T\frac{\partial e_C}{\partial W}) \tag{4.107}$$

$$V(k+1) = V(k) - \eta({e_C}^T\frac{\partial e_C}{\partial V}) \tag{4.108}$$

The derivate of the errors with respect to the network gains are as follows:

$$\frac{\partial e_C}{\partial W} = \frac{\partial e_C}{\partial v_a}\times\frac{\partial v_a}{\partial W} \tag{4.109}$$

Using the chain rule we have:

$$\frac{\partial e_C}{\partial W} = \frac{\partial e_C}{\partial v_a}\times\frac{\partial v_a}{\partial \bar{\tau}}\times\frac{\partial \bar{\tau}}{\partial W} \tag{4.110}$$

From the control structure shown in figure (1) we have:

$$K\times e_C + \hat{\tau} = \bar{\tau} \tag{4.111}$$

Substituting the above terms in equations (4.110), we have:

$$\frac{\partial e_C}{\partial W} = \frac{\partial e_C}{\partial v_a} \times \frac{\partial v_a}{\partial \bar{\tau}} \times \frac{\partial (K \times e_C + \hat{\tau})}{\partial W} \tag{4.112}$$

Expanding the last term in the above equation, we have:

$$\frac{\partial e_C}{\partial W} = \frac{\partial e_C}{\partial v_a} \times \frac{\partial v_a}{\partial \bar{\tau}} \times \left(K \frac{\partial (e_C)}{\partial W} + \frac{\partial \hat{\tau}}{\partial W} \right) \tag{4.113}$$

The first term in the above equation is:

$$\frac{\partial e_C}{\partial v_a} = \frac{\partial (v_c - v_a)}{\partial v_a} = -1 \tag{4.114}$$

Substituting (4.114) into (4.113) we have:

$$\frac{\partial e_C}{\partial W} = -1 \times \frac{\partial v_a}{\partial \bar{\tau}} \times \left(K \frac{\partial (e_C)}{\partial W} + \frac{\partial \hat{\tau}}{\partial W} \right) = -\frac{\partial v_a}{\partial \bar{\tau}} \times K \frac{\partial (e_C)}{\partial W} - \frac{\partial v_a}{\partial \bar{\tau}} \times \frac{\partial \hat{\tau}}{\partial W} \tag{4.115}$$

Simplifying the above equation will yield:

$$\frac{\partial e_C}{\partial W} \left(1 + K \frac{\partial v_a}{\partial \bar{\tau}} \right) = -\frac{\partial v_a}{\partial \bar{\tau}} \times \frac{\partial \hat{\tau}}{\partial W} \tag{4.116}$$

$$\frac{\partial e_C}{\partial W} = \frac{-\frac{\partial v_a}{\partial \bar{\tau}} \times \frac{\partial \hat{\tau}}{\partial W}}{1 + K \frac{\partial v_a}{\partial \bar{\tau}}} \tag{4.117}$$

Using the same procedure we can have the derivative of the errors with respect to the gains of the first layer. Therefore the equations that we need to train the network are as follows:

$$\frac{\partial e_C}{\partial W} = \frac{-Jac \times \frac{\partial \hat{\tau}}{\partial W}}{1 + K \times Jac} \tag{4.118}$$

$$\frac{\partial e_C}{\partial V} = \frac{-Jac \times \frac{\partial \hat{\tau}}{\partial V}}{1 + K \times Jac} \tag{4.119}$$

In which:

$$e_c = \begin{bmatrix} e_{cv} \\ e_{c\omega} \end{bmatrix} \quad , v_a = \begin{bmatrix} v \\ \omega \end{bmatrix} \quad , \hat{\tau} = \begin{bmatrix} \hat{\tau}_1 \\ \hat{\tau}_2 \end{bmatrix} \quad , \bar{\tau} = \begin{bmatrix} \bar{\tau}_1 \\ \bar{\tau}_2 \end{bmatrix} \tag{4.120}$$

$$Jac = \frac{\partial v_a}{\partial \bar{\tau}} = Jacobian\ matrix = \begin{bmatrix} \frac{\partial v}{\partial \bar{\tau}_1} & \frac{\partial \omega}{\partial \bar{\tau}_1} \\ \frac{\partial v}{\partial \bar{\tau}_2} & \frac{\partial \omega}{\partial \bar{\tau}_2} \end{bmatrix} \tag{4.121}$$

$$\frac{\partial \hat{\tau}}{\partial W} = \begin{bmatrix} \frac{\partial \hat{\tau}_1}{\partial W} \\ \frac{\partial \hat{\tau}_2}{\partial W} \end{bmatrix} \quad , \quad \frac{\partial \hat{\tau}}{\partial V} = \begin{bmatrix} \frac{\partial \hat{\tau}_1}{\partial V} \\ \frac{\partial \hat{\tau}_2}{\partial V} \end{bmatrix} \tag{4.122}$$

$$\frac{\partial e_C}{\partial W} = \begin{bmatrix} \frac{\partial e_{cv}}{\partial W} \\ \frac{\partial e_{c\omega}}{\partial W} \end{bmatrix} \quad , \quad \frac{\partial e_C}{\partial V} = \begin{bmatrix} \frac{\partial e_{cv}}{\partial V} \\ \frac{\partial e_{c\omega}}{\partial V} \end{bmatrix} \tag{4.123}$$

$$K = \begin{bmatrix} k_4 & 0 \\ 0 & k_4 \end{bmatrix} \tag{4.124}$$

Substituting equations (4.120) to (4.124) in equation (4.118) and (4.119) we have:

$$\begin{bmatrix} \frac{\partial e_{cv}}{\partial W} \\ \frac{\partial e_{c\omega}}{\partial W} \end{bmatrix} = \frac{-\begin{bmatrix} \frac{\partial v}{\partial \bar{\tau}_1} & \frac{\partial \omega}{\partial \bar{\tau}_1} \\ \frac{\partial v}{\partial \bar{\tau}_2} & \frac{\partial \omega}{\partial \bar{\tau}_2} \end{bmatrix} \times \begin{bmatrix} \frac{\partial \hat{\tau}_1}{\partial W} \\ \frac{\partial \hat{\tau}_2}{\partial W} \end{bmatrix}}{I + \begin{bmatrix} k_4 & 0 \\ 0 & k_4 \end{bmatrix} \begin{bmatrix} \frac{\partial v}{\partial \bar{\tau}_1} & \frac{\partial \omega}{\partial \bar{\tau}_1} \\ \frac{\partial v}{\partial \bar{\tau}_2} & \frac{\partial \omega}{\partial \bar{\tau}_2} \end{bmatrix}} \tag{4.125}$$

$$\begin{bmatrix} \frac{\partial e_{cv}}{\partial V} \\ \frac{\partial e_{c\omega}}{\partial V} \end{bmatrix} = \frac{-\begin{bmatrix} \frac{\partial v}{\partial \bar{\tau}_1} & \frac{\partial \omega}{\partial \bar{\tau}_1} \\ \frac{\partial v}{\partial \bar{\tau}_2} & \frac{\partial \omega}{\partial \bar{\tau}_2} \end{bmatrix} \times \begin{bmatrix} \frac{\partial \hat{\tau}_1}{\partial V} \\ \frac{\partial \hat{\tau}_2}{\partial V} \end{bmatrix}}{I + \begin{bmatrix} k_4 & 0 \\ 0 & k_4 \end{bmatrix} \begin{bmatrix} \frac{\partial v}{\partial \bar{\tau}_1} & \frac{\partial \omega}{\partial \bar{\tau}_1} \\ \frac{\partial v}{\partial \bar{\tau}_2} & \frac{\partial \omega}{\partial \bar{\tau}_2} \end{bmatrix}} \tag{1.126}$$

The derivative terms $\frac{\partial \hat{\tau}}{\partial W}$ and $\frac{\partial \hat{\tau}}{\partial V}$ in equations (4.118) and (4.119) can be found from the neural network governing equations and will be explained in the next section.

4.3.2.1.1 The neural network derivative terms calculation

Using the backpropagation formulation and the following figures terminology, the derivatives for the second layer are:

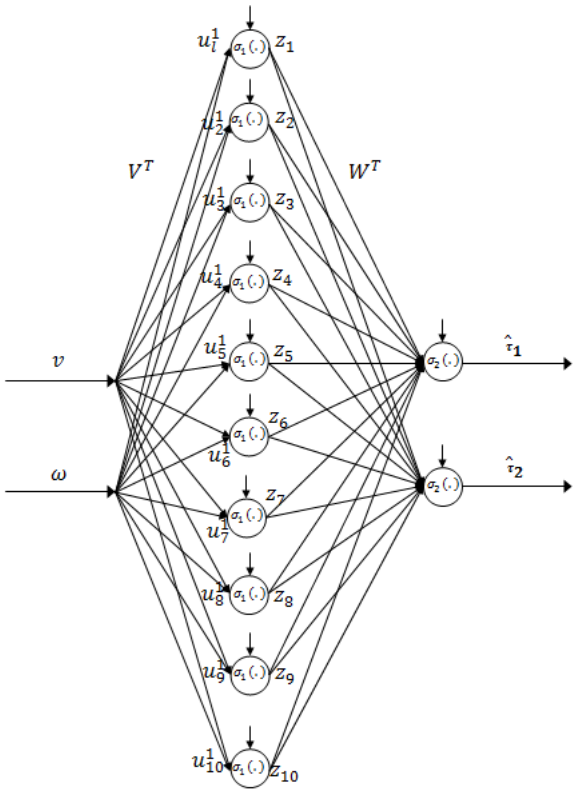

Figure 4-47: the inverse model neural network details

Assuming a linear activation function for the output layer, we have:

$$\hat{\tau}_1 = \sum_{l=1}^{N_h} z_l \times W_{1l} \quad , \quad l = 1,2,\dots,N_h \tag{4.127}$$

$$\hat{\tau}_2 = \sum_{l=1}^{N_h} z_l \times W_{2l} \quad , \quad l = 1,2,\dots,N_h \tag{4.128}$$

According to the above equations for the NN output, we have:

$$\frac{\partial \hat{\tau}_1}{\partial W_{1l}} = z_l, \quad \frac{\partial \hat{\tau}_2}{\partial W_{2l}} = z_l \tag{4.129}$$

The output of each neuron in the hidden layer is z_l and can be computed as based on the network inputs, first layer weights and hidden layer activation function as follows:

$$z_l = \sigma\left(u_l^1\right) \tag{4.130}$$

$$u_l^1 = \sum_{j=0}^{n} V_{lj} x_j = V_{l1} v + V_{l2} \omega \quad j = 1,2\,, \;\; l = 1,2,\dots,N_h \tag{4.131}$$

According to equations (4.130) to (4.131), the required derivatives are:

$$\frac{\partial \hat{\tau}_1}{\partial W_{1l}} = \sigma(V_{l1} v + V_{l2} \omega), \quad \frac{\partial \hat{\tau}_2}{\partial W_{2l}} = \sigma(V_{l1} v + V_{l2} \omega) \tag{4.132}$$

The derivatives for the first layer are:

$$\frac{\partial \hat{\tau}_1}{\partial V_{lj}} = \frac{\partial \hat{\tau}_1}{\partial z_l} \times \frac{\partial z_l}{\partial V_{lj}} \tag{4.133}$$

According to equation (4.127), the first derivative in the above equation is:

$$\frac{\partial \hat{\tau}_1}{\partial z_l} = W_{1l} \tag{4.134}$$

The output of the hidden layer neuron is:

$$z_l = \sigma\left(u_l^1\right) = \sigma\left(\sum_{j=0}^{n} V_{lj} x_j\right) \tag{4.135}$$

Therefore, the second derivative in equation (4.133) is:

$$\frac{\partial z_l}{\partial V_{lj}} = \acute{\sigma}\left(u_l^1\right) x_j \tag{4.136}$$

Therefore, the derivative with respect to the second layer weights is:

$$\frac{\partial \hat{\tau}_1}{\partial V_{lj}} = W_{1l} \times \acute{\sigma}\left(u_l^1\right) x_j = W_{1l} \times \acute{\sigma}(V_{l1} v + V_{l2} \omega) \times x_j \tag{4.137}$$

$$\frac{\partial \hat{\tau}_2}{\partial V_{lj}} = W_{2l} \times \acute{\sigma}\left(u_l^1\right) x_j = W_{2l} \times \acute{\sigma}(V_{l1} v + V_{l2} \omega) \times x_j \tag{4.138}$$

4.3.2.1.2 The Jacobian matrix calculation

The Jacobian can be computed based on the robot dynamic equations. The mobile robot dynamic equation is:

$$\bar{M}(q)\dot{v}(t) + \bar{V}_m(q,\dot{q})v(t) + \bar{F}(\dot{q}) + \bar{G}(q) + \bar{\tau}_d = \bar{B}(q)\tau \tag{4.139}$$

$$\bar{B}(q)^{-1}\bar{\tau} = \tau \tag{4.140}$$

$$\begin{aligned} v &= [v \quad \omega]^T \\ \dot{v} &= [\dot{v} \quad \dot{\omega}]^T \end{aligned} \tag{4.141}$$

The simplified dynamic equation for this mobile robot is as follows:

$$\bar{M}(q)\dot{v}(t) + \bar{V}_m(q,\dot{q})v(t) = \bar{\tau} \tag{4.142}$$

$$\begin{bmatrix} \bar{M}(q)_{11} & \bar{M}(q)_{12} \\ \bar{M}(q)_{21} & \bar{M}(q)_{22} \end{bmatrix} \begin{bmatrix} \dot{v} \\ \dot{\omega} \end{bmatrix} + \begin{bmatrix} \bar{V}_m(q,\dot{q})_{11} & \bar{V}_m(q,\dot{q})_{12} \\ \bar{V}_m(q,\dot{q})_{21} & \bar{V}_m(q,\dot{q})_{22} \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \bar{\tau}_1 \\ \bar{\tau}_2 \end{bmatrix} \tag{4.143}$$

Expanding the above equation, we have:

$$\bar{M}(q)_{11}\dot{v} + \bar{M}(q)_{12}\dot{\omega} + \bar{V}_m(q,\dot{q})_{11} v + \bar{V}_m(q,\dot{q})_{12}\omega = \bar{\tau}_1 \tag{4.144}$$

$$\bar{M}(q)_{21}\dot{v} + \bar{M}(q)_{22}\dot{\omega} + \bar{V}_m(q,\dot{q})_{21} v + \bar{V}_m(q,\dot{q})_{22}\omega = \bar{\tau}_2 \tag{4.144}$$

For the Jacobian matrix, we are interested to find $\frac{\partial v}{\partial \bar{\tau}}$. From equation (4.142) we have:

$$\dot{v}(t) = -\bar{M}(q)^{-1}\bar{V}_m(q,\dot{q})v(t) + \bar{M}(q)^{-1}\bar{\tau} \tag{4.145}$$

Expanding the above equation, we have:

$$\begin{aligned} \begin{bmatrix} \dot{v} \\ \dot{\omega} \end{bmatrix} = &- \begin{bmatrix} \bar{M}(q)^{-1}{}_{11} & \bar{M}(q)^{-1}{}_{12} \\ \bar{M}(q)^{-1}{}_{21} & \bar{M}(q)^{-1}{}_{22} \end{bmatrix} \begin{bmatrix} \bar{V}_m(q,\dot{q})_{11} & \bar{V}_m(q,\dot{q})_{12} \\ \bar{V}_m(q,\dot{q})_{21} & \bar{V}_m(q,\dot{q})_{22} \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \\ &+ \begin{bmatrix} \bar{M}(q)^{-1}{}_{11} & \bar{M}(q)^{-1}{}_{12} \\ \bar{M}(q)^{-1}{}_{21} & \bar{M}(q)^{-1}{}_{22} \end{bmatrix} \begin{bmatrix} \bar{\tau}_1 \\ \bar{\tau}_2 \end{bmatrix} \end{aligned} \tag{4.146}$$

From equation (4.145) we have:

$$v(t) = \int_0^t -\bar{M}(q)^{-1}\bar{V}_m(q,\dot{q})v(t)dt + \int_0^t \bar{M}(q)^{-1}\bar{\tau}dt \tag{4.147}$$

Expanding the above equation, we have:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \int_0^t -\begin{bmatrix} \bar{M}(q)^{-1}{}_{11} & \bar{M}(q)^{-1}{}_{12} \\ \bar{M}(q)^{-1}{}_{21} & \bar{M}(q)^{-1}{}_{22} \end{bmatrix}\begin{bmatrix} \bar{V}_m(q,\dot{q})_{11} & \bar{V}_m(q,\dot{q})_{12} \\ \bar{V}_m(q,\dot{q})_{21} & \bar{V}_m(q,\dot{q})_{22} \end{bmatrix}\begin{bmatrix} v \\ \omega \end{bmatrix} dt + \int_0^t \begin{bmatrix} \bar{M}(q)^{-1}{}_{11} & \bar{M}(q)^{-1}{}_{12} \\ \bar{M}(q)^{-1}{}_{21} & \bar{M}(q)^{-1}{}_{22} \end{bmatrix}\begin{bmatrix} \bar{\tau}_1 \\ \bar{\tau}_2 \end{bmatrix} dt \tag{4.148}$$

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \int_0^t -\begin{bmatrix} \bar{M}(q)^{-1}{}_{11} & \bar{M}(q)^{-1}{}_{12} \\ \bar{M}(q)^{-1}{}_{21} & \bar{M}(q)^{-1}{}_{22} \end{bmatrix}\begin{bmatrix} \bar{V}_m(q,\dot{q})_{11} & \bar{V}_m(q,\dot{q})_{12} \\ \bar{V}_m(q,\dot{q})_{21} & \bar{V}_m(q,\dot{q})_{22} \end{bmatrix}\begin{bmatrix} v \\ \omega \end{bmatrix} dt + \int_0^t \begin{bmatrix} \bar{M}(q)^{-1}{}_{11} \times \bar{\tau}_1 + \bar{M}(q)^{-1}{}_{12} \times \bar{\tau}_2 \\ \bar{M}(q)^{-1}{}_{21} \times \bar{\tau}_1 + \bar{M}(q)^{-1}{}_{22} \times \bar{\tau}_2 \end{bmatrix} dt \tag{4.149}$$

According to the above equation, the required derivatives are as follows:

$$\frac{\partial v}{\partial \bar{\tau}_1} = \int_0^t \bar{M}(q)^{-1}{}_{11}dt \tag{4.150}$$

$$\frac{\partial v}{\partial \bar{\tau}_2} = \int_0^t \bar{M}(q)^{-1}{}_{12}dt \tag{4.151}$$

$$\frac{\partial \omega}{\partial \bar{\tau}_1} = \int_0^t \bar{M}(q)^{-1}{}_{21}dt \tag{4.152}$$

$$\frac{\partial \omega}{\partial \bar{\tau}_2} = \int_0^t \bar{M}(q)^{-1}{}_{22}dt \tag{4.153}$$

According to the mobile robot dynamic model, $\bar{M}(q)$ is as follows:

$$\bar{M}(q) = \begin{bmatrix} m & 0 \\ 0 & I_C + ma^2 \end{bmatrix} \tag{4.154}$$

$$\bar{M}(q)^{-1} = \frac{1}{m(I_C + ma^2)}\begin{bmatrix} I_C + ma^2 & 0 \\ 0 & m \end{bmatrix} = \begin{bmatrix} \frac{1}{m} & 0 \\ 0 & \frac{1}{I_C + ma^2} \end{bmatrix} \tag{4.155}$$

According to equation (4.155) the components of the Jacobian matrix are as follows:

$$\begin{aligned} Jac = \frac{\partial v_a}{\partial \bar{\tau}} = Jacobian\ matrix &= \begin{bmatrix} \frac{\partial v}{\partial \bar{\tau}_1} & \frac{\partial \omega}{\partial \bar{\tau}_1} \\ \frac{\partial v}{\partial \bar{\tau}_2} & \frac{\partial \omega}{\partial \bar{\tau}_2} \end{bmatrix} \\ &= \begin{bmatrix} \int_0^t \bar{M}(q)^{-1}{}_{11} dt & \int_0^t \bar{M}(q)^{-1}{}_{21} dt \\ \int_0^t \bar{M}(q)^{-1}{}_{12} dt & \int_0^t \bar{M}(q)^{-1}{}_{22} dt \end{bmatrix} \\ &= \begin{bmatrix} \int_0^t \frac{1}{m} dt & 0 \\ 0 & \int_0^t \frac{1}{I_C + ma^2} dt \end{bmatrix} = \begin{bmatrix} \frac{1}{m} t & 0 \\ 0 & \frac{1}{I_C + ma^2} t \end{bmatrix} \end{aligned} \tag{4.156}$$

Substituting equation (4.155), (4.138), (4.139) and (4.130) in equations (4.126) and (4.127) we have:

$$\frac{\partial e_C}{\partial W} = \begin{bmatrix} \frac{\partial e_{cv}}{\partial W} \\ \frac{\partial e_{c\omega}}{\partial W} \end{bmatrix} = \frac{-Jac \times \begin{bmatrix} z_l \\ z_l \end{bmatrix}}{I + \begin{bmatrix} k_4 & 0 \\ 0 & k_4 \end{bmatrix} \times Jac} \tag{4.157}$$

$$\frac{\partial e_C}{\partial V} = \begin{bmatrix} \frac{\partial e_{cv}}{\partial V} \\ \frac{\partial e_{c\omega}}{\partial V} \end{bmatrix} = \frac{-Jac \times \begin{bmatrix} W_{1l} \times \acute{\sigma}(V_{l1} v + V_{l2} \omega) \times x_j \\ W_{2l} \times \acute{\sigma}(V_{l1} v + V_{l2} \omega) \times x_j \end{bmatrix}}{I + \begin{bmatrix} k_4 & 0 \\ 0 & k_4 \end{bmatrix} \times Jac} \tag{4.158}$$

The above derivatives will be used in the main error equations (4.103) and (4.104):

$$\frac{\partial E}{\partial W} = {e_C}^T \frac{\partial e_C}{\partial W} = {e_C}^T \times \frac{-Jac \times \begin{bmatrix} z_l \\ z_l \end{bmatrix}}{I + \begin{bmatrix} k_4 & 0 \\ 0 & k_4 \end{bmatrix} \times Jac} \tag{4.159}$$

$$\frac{\partial E}{\partial V} = {e_C}^T \frac{\partial e_C}{\partial V} = {e_C}^T \times \frac{-Jac \times \begin{bmatrix} W_{1l} \times \acute{\sigma}(V_{l1} v + V_{l2} \omega) \times x_j \\ W_{2l} \times \acute{\sigma}(V_{l1} v + V_{l2} \omega) \times x_j \end{bmatrix}}{I + \begin{bmatrix} k_4 & 0 \\ 0 & k_4 \end{bmatrix} \times Jac} \tag{4.160}$$

Substituting the above two equations in (4.102) and (4.103) will yield the learning algorithm for the neural network:

$$W(k+1) = W(k) - \Delta W \tag{4.161}$$

$$V(k+1) = V(k) - \Delta V \tag{4.162}$$

In which:

$$\Delta W = \eta \times {e_C}^T \times \frac{-Jac \times \begin{bmatrix} z_l \\ z_l \end{bmatrix}}{I + \begin{bmatrix} k_4 & 0 \\ 0 & k_4 \end{bmatrix} \times Jac} \tag{4.163}$$

$$\Delta V = \eta \times {e_C}^T \times \frac{-Jac \times \begin{bmatrix} W_{1l} \times \acute{\sigma}(V_{l1} v + V_{l2} \omega) \times x_j \\ W_{2l} \times \acute{\sigma}(V_{l1} v + V_{l2} \omega) \times x_j \end{bmatrix}}{I + \begin{bmatrix} k_4 & 0 \\ 0 & k_4 \end{bmatrix} \times Jac} \tag{4.164}$$

Using equations (4.161) to (4.164) we can have a neural network that produces $\hat{\tau} = [\hat{\tau}_1 \quad \hat{\tau}_2]^T$ torques to make the error signal $e_c = [e_{cv} \quad e_{c\omega}]^T$ zero. The most important step that should be taken before using the above learning algorithm is to train the inverse model neural network so that its approximation error will be minimal. Different stages of offline and online training with different input signals to excite all the frequency modes of the model has been done and will be explained in the next section. Training the inverse model neural network, we can use implement it in the control structure of figure (4.1).

4.3.2.2 The inverse model neural network development and training

The following steps are taken to train the neural network and make it ready for online approximation of the system model:

- Offline training of the neural network with sinusoidal and chirp inputs with different frequencies to excite all the modal frequencies of the system.

- Optimizing the neural network parameters to reach the best approximation performance possible for a chirp input which contains all the frequencies of the system.
- Testing the online learning performance of the neural network with the network weights obtained from the offline training and comparing it with the real robot model outputs.

The Simulink block diagram shown in Figure 4-48 is used to generate the required data to perform the offline training of the network with the chirp input:

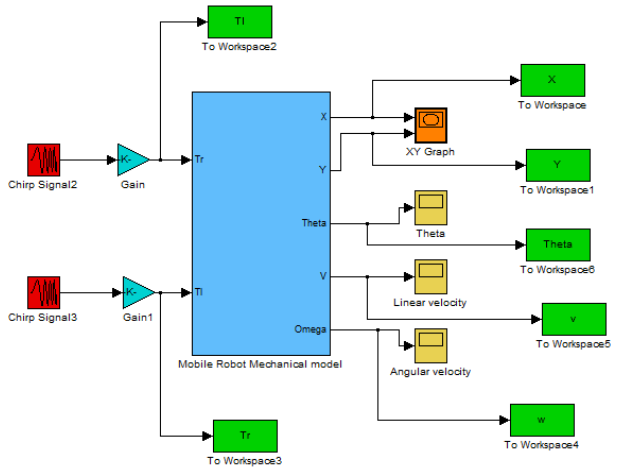

Figure 4-48: the Simulink block diagram to generate the required data for the offline training of the inverse model neural network

The input chirp signals used to generate the data are shown in figure 4-49:

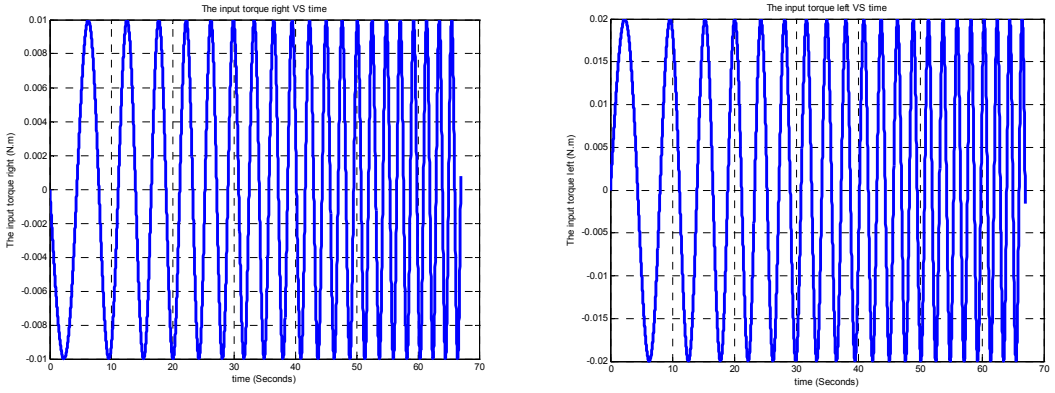

Figure 4-49: The input chirp signals to the robot model

The frequencies of the above chirp signal vary between 0.1 to 1 hertz. The parameters of the neural network which should be used for the approximation of the inverse model should be optimized to make guarantee the best performance possible. A comprehensive analysis on the effect of each neural network parameter has been done on the inverse model neural network with the above chirp input. The results and details of the inverse model neural network parameter optimization are shown in next section.

4.3.2.2.1 The inverse model neural network parameter optimization

The following neural network parameters should be optimized to have the best approximation performance for the neural network:

$$N_h: Number\ of\ neurons\ in\ the\ hidden\ layer\ of\ the\ Neural\ Network$$

$$\eta: The\ learning\ rate\ for\ the\ back-propagation\ algorithm$$

$$\beta: The\ momentum\ rate\ for\ the\ back-propagation\ algorithm$$

The first parameter that is been analyzed is the number of neurons in the hidden layer N_h. Note that once we want to investigate the effect of one parameter, we make the others fixed and change the required parameter. The effect of changing N_h on the learning performance of the neural network is shown in Figure 4-50:

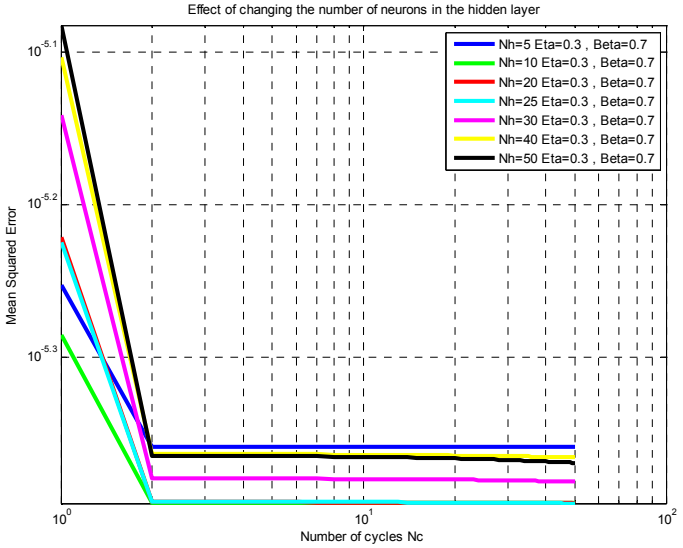

Figure 4-50: the effect of number of neurons in the hidden layer on the learning performance of the neural network

Note that the above analysis is been done for 75 cycles of learning for each value of N_h . According to the above figure, the best learning performance is related to $N_h = 0, 20, 25$.

In order to choose the best value between the above three values we should focus on the final section of the graph to see which one will reach the smallest error. The zoomed graph of the above figure is shown in Figure 4-51:

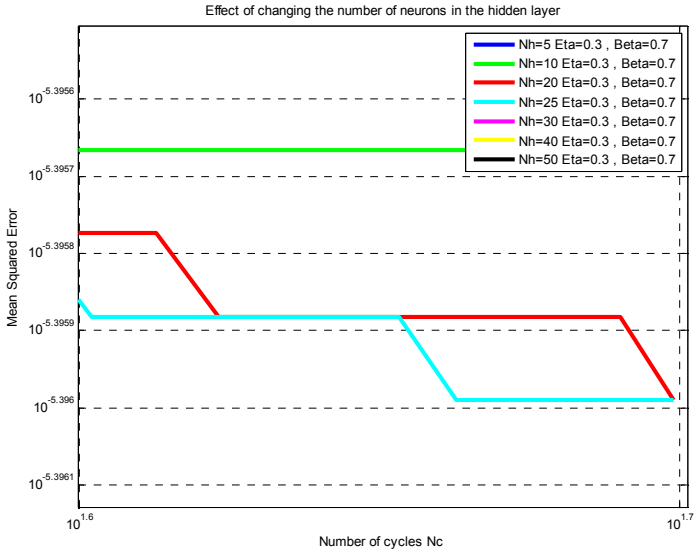

Figure 4-51: the zoomed graph of the Nh analysis graph

As it can be seen from the above figure, the smallest error is related to $N_h = 25$. Therefore the best value to be chosen for the number of neurons in the hidden layer is:

$$N_h = 25$$

The next comparison analysis should be done on the learning rate and momentum rate values. The first analysis is done when:

$$\beta + \eta = 1 \tag{4.165}$$

The effect of changing the learning rate and momentum rate values on the learning performance of the neural network is shown in Figure 4-52:

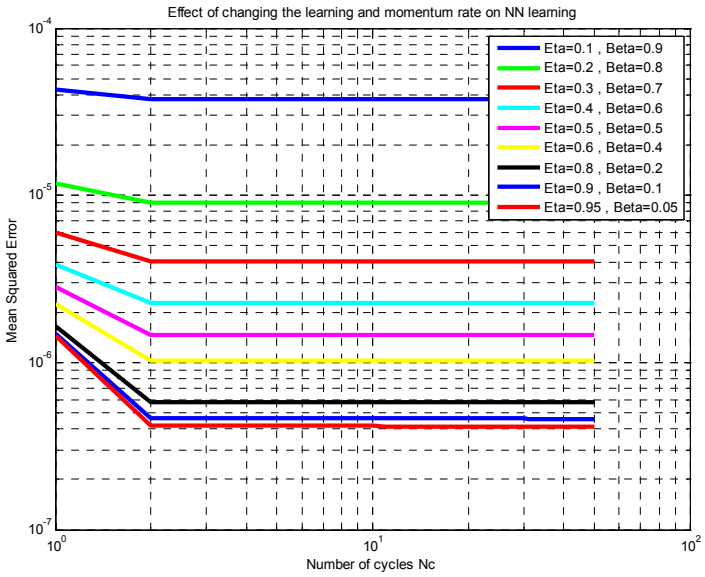

Figure 4-52: the effect of the learning and momentum rate on the learning performance when the sum of the values is one

According to the above figure, the best values for the learning rate and momentum rate when we are following equation (4.165) are:

$$\eta = 0.95, \beta = 0.05$$

The next step is to investigate the effect of changing the learning rate and momentum rate without following equation (4.165) which is shown in Figure 4-53:

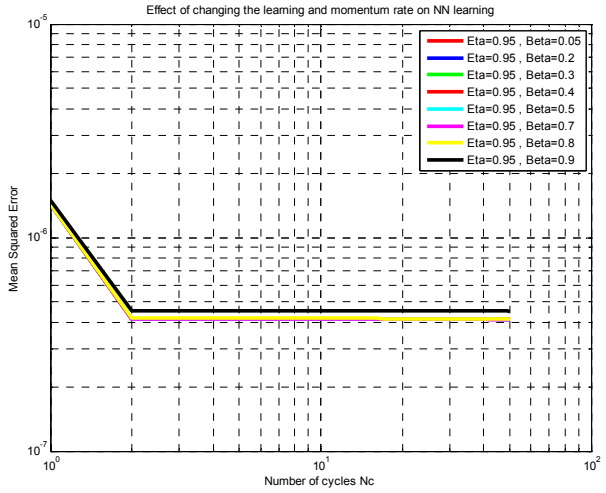

Figure 4-53: the effect of changing the learning rate and the momentum rate on the learning performance

In order to choose the best value between the above values we should focus on the final section of the graph to see which one will reach the smallest error. The zoomed graph of the above figure is shown in Figure 4-54:

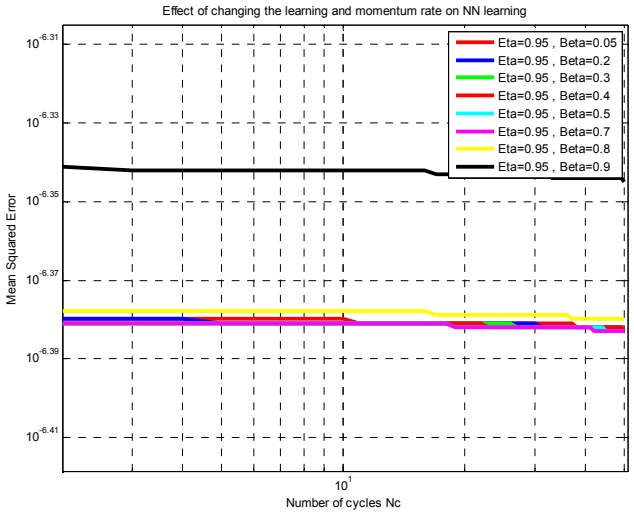

Figure 4-54: the effect of changing the learning rate and the momentum rate on the learning performance (zoomed plot)

According to the above figure, the best values for the learning rate and momentum rate are:

$$\eta = 0.95, \beta = 0.7$$

The above performance analysis shows that the best neural network parameters to approximate the inverse model are as follows:

$$N_h = 25$$

$$\eta = 0.95, \beta = 0.7$$

The above parameters are used to perform the offline training of the inverse model neural network using the generated input chirp signals shown in figure (4-49). The number of cycles and the used activation functions for the offline training are as follows:

$$N_c = 200 : \textit{The number of cycles the set of input data will repeat}$$

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

$$: \textit{The hyper tangent activation function for the hidden layer}$$

$$f_1(x) = x : \textit{The linear activation function for the output layer}$$

The Matlab code used to perform the offline training with the above parameters is included in the appendix. The neural network mean square error after 100 cycles of training is shown in Figure 4-55:

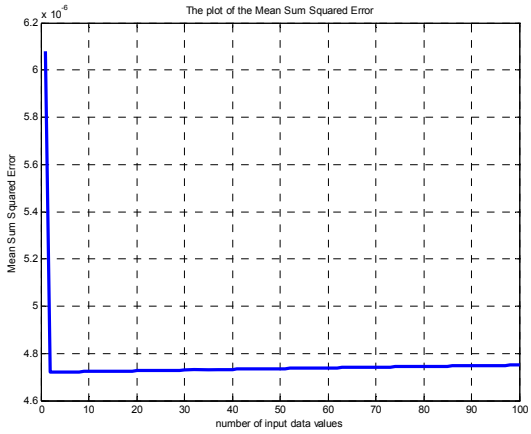

Figure 4-55: the mean squared error for 100 cycles of training (chirp input)

The final values of the mean squared error, corresponding to the above learning is:

$$MSE_{final} = 4.7 \times 10^{-6}$$

The mean square error is at the minimum value after 100 cycles of training which shows that the number of cycles used to train the network is enough and the approximation performance should be satisfactory. The neural network outputs and the inverse model's output are shown in Figures 4-56 and 4-57:

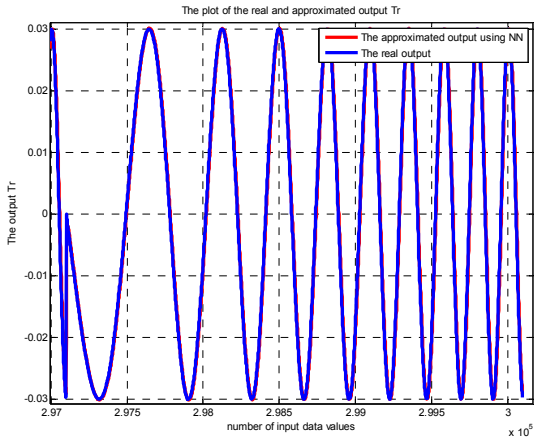

Figure 4-56: The output Tr of the model (BLUE) and the NN (RED) for the last cycle of learning (Chirp input)

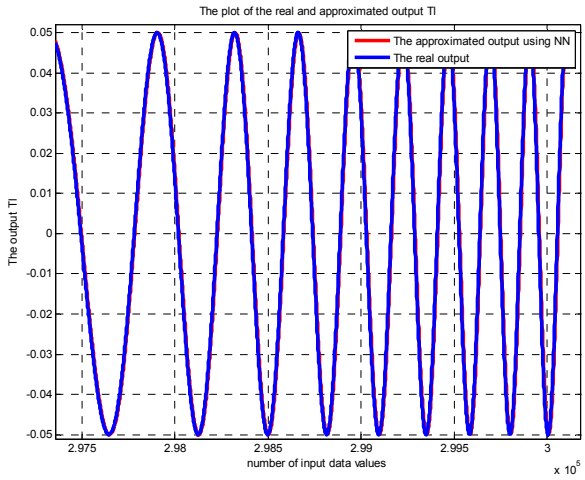

Figure 4-57: The output Tl of the model (BLUE) and the NN (RED) for the last cycle of learning (Chirp input)

Perfect estimation is achieved after this number of offline training as it can be seen from the above figures. The next step is to do the offline training using the chirp signal as the input to the robot system. The Simulink block diagram shown in Figure 4-58 is used to generate the required data to do the offline training using the sinusoidal input:

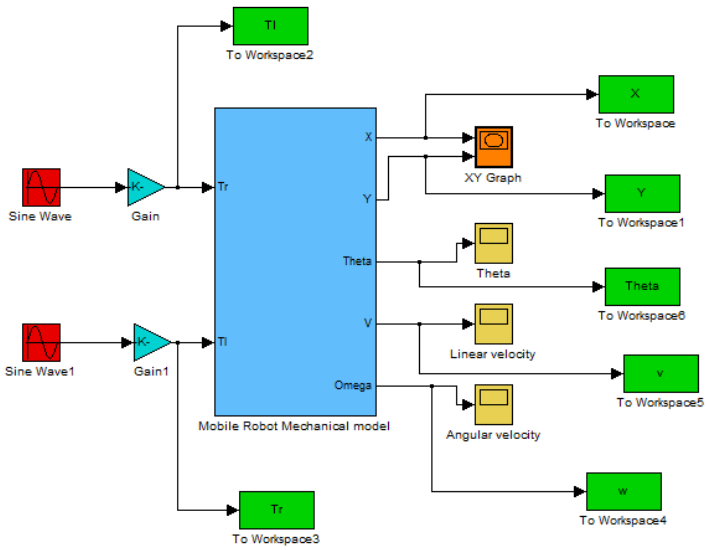

Figure 4-58: the Simulink block diagram used to generate the required data for offline training with sinusoidal input

The input sinusoidal signals used to generate the data are shown in Figure 4-59:

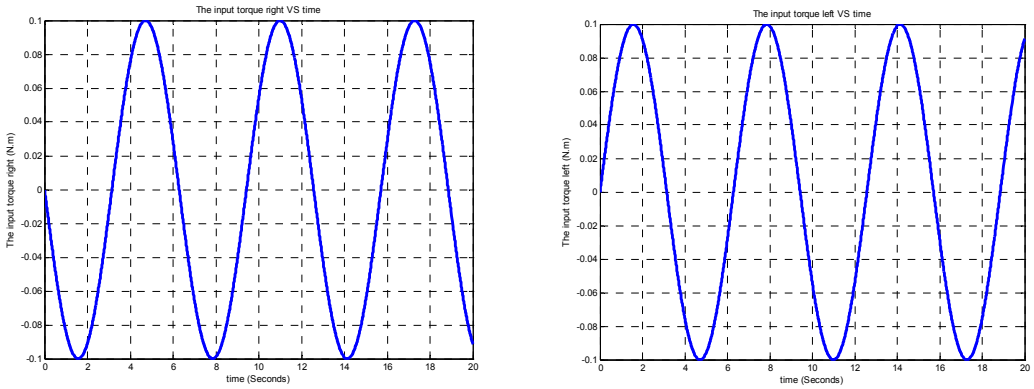

Figure 4-59: The sinusoidal input to the robot system

The same neural network structure and parameters are used to perform the offline training on the above data. The only difference is that the trained network with the chirp input is used to do the training with the sinusoidal input. The neural network mean square error after 100 cycles of training is shown in Figure 4-60:

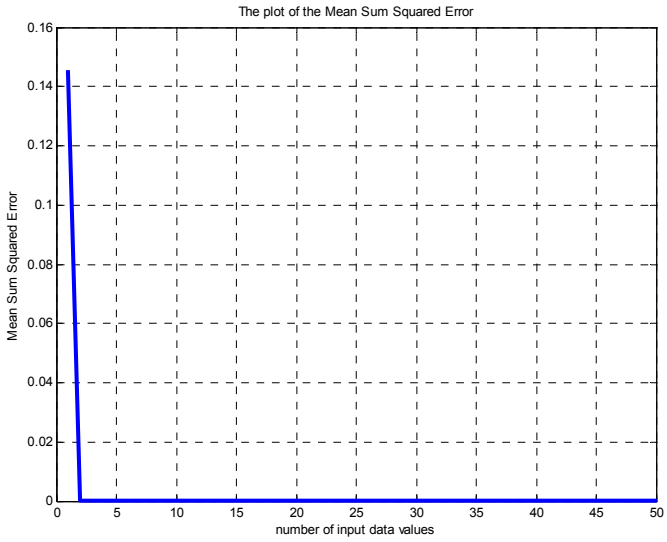

Figure 4-60: the mean squared error for 100 cycles of training (sinusoidal input)

The final values of the mean squared error, corresponding to the above learning is:

$$MSE_{final} = 3.44 \times 10^{-6}$$

The mean square error is at the minimum value after 100 cycles of training which shows that the number of cycles used to train the network is enough and the approximation

performance should be satisfactory. The neural network outputs and the inverse model's output are shown in Figures 4-61 and 4-62:

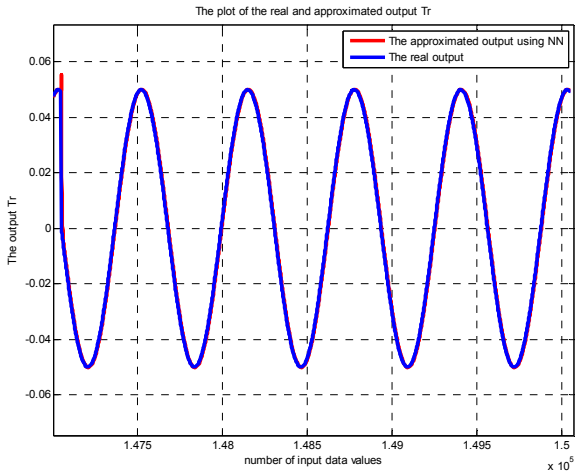

Figure 4-61: The output Tr of the model (BLUE) and the NN (RED) for the last cycle of learning (Sinusoidal input)

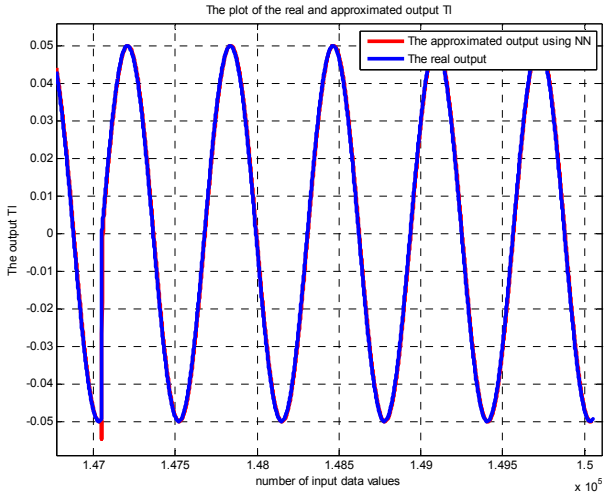

Figure 4-62: The output Tl of the model (BLUE) and the NN (RED) for the last cycle of learning (Sinusoidal input)

Perfect estimation is achieved after this number of offline training as it can be seen from the above figures. The next step of the neural network preparation for the direct modeling purpose is to test the offline trained network in online learning and check if it can result a precise approximation when it is learning the system online. The Simulink model shown in Figure 4-63 is used to check the online learning performance of the trained neural network and compare it with the actual outputs of the robot system:

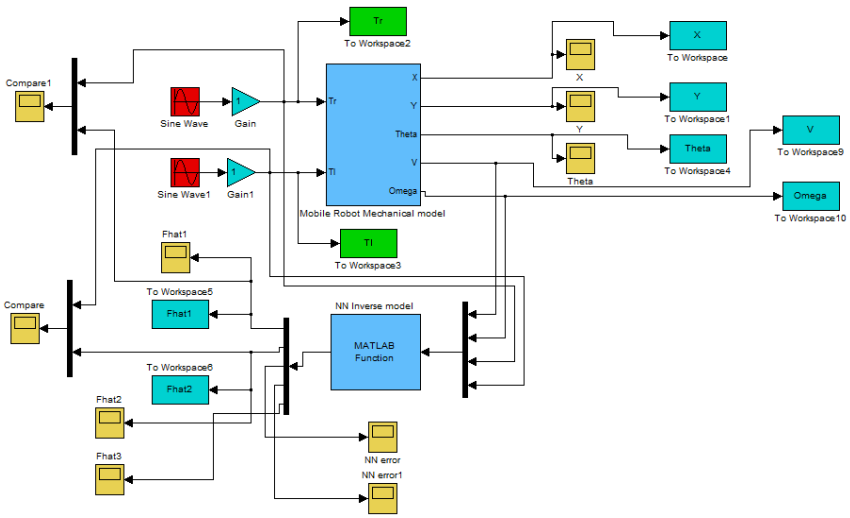

Figure 4-63: The Simulink block diagram used to check the online learning performance of the inverse model neural network

The above Simulink block diagram sends the same sinusoidal input signal to the robot model and the inverse model neural network and compares their outputs. The same neural network parameters used for offline training are used for the online training. The only difference is that the weights used in the online neural network are the ones obtained from the comprehensive offline training explained above. The robot inverse model outputs and the online neural network outputs are shown in Figures 4-64 and 4-47 to check the approximation performance:

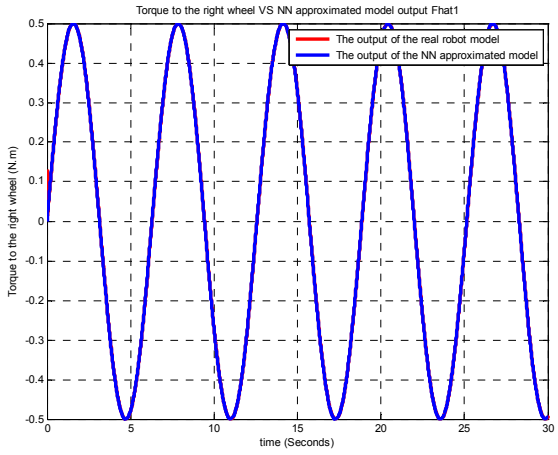

Figure 4-64: the inverse model output Tr VS NN model output Tr (online training)

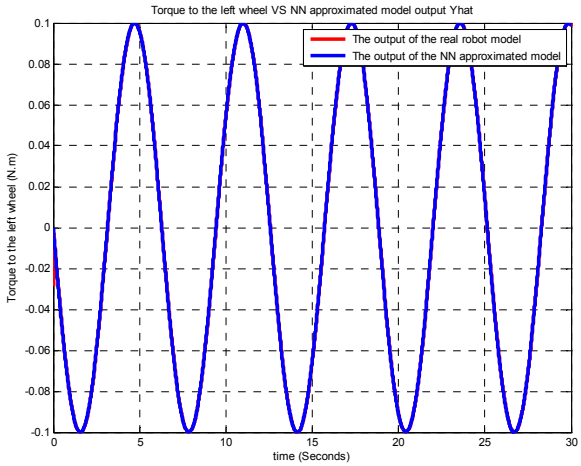

Figure 4-65: the inverse model output Tl VS NN model output Tl (online training)

As it can be seen from the above figures, the neural network is well trained and can perfectly approximate mobile robot model online. This trained network can be used to generate $\hat{\tau} = [\hat{\tau}_1 \quad \hat{\tau}_2]^T$ torques to make the error signal $e_c = [e_{cv} \quad e_{c\omega}]^T$ zero.

4.3.2.3 Comparative analysis for NN inverse model controller and backstepping controller

The simulation results of the neural network inverse model controller can be divided into the following two categories:

- The normal trajectory tracking results without any external disturbance on the robot
- The trajectory tracking in presence of an external disturbance on the robot motor torques, velocities and position.

The first part of the simulation results is allocated for comparison study between the normal trajectory tracking between backstepping controller and neural network inverse model controller. The Simulink block diagram used to simulate the performance of the NN inverse model controller is shown in Figure 4-66:

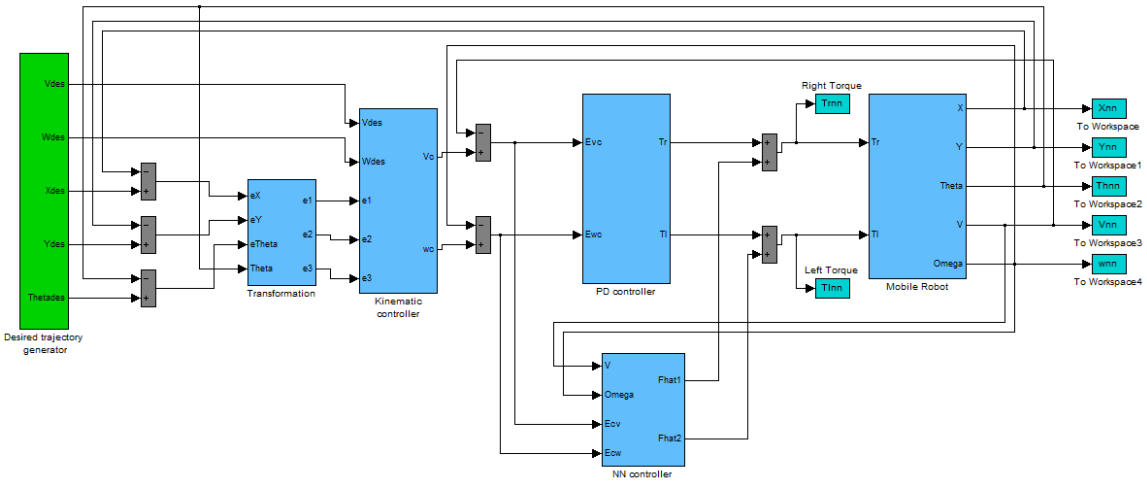

Figure 4-66: the Simulink block diagram to simulate the performance of the NN inverse model controller

The NN controller block in the above block diagram is the inverse model controller which produces torques to compensate for the mobile robot nonlinear dynamics. The details of this block are shown in Figure 4-67:

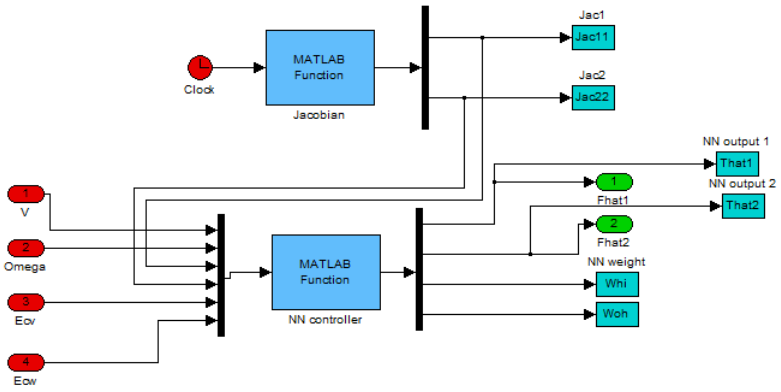

Figure 4-67: the NN inverse model block details

The Jacobian block and the NN controller block use two Matlab functions which implement the corresponding equations in Matlab and are included in the appendix. The neural network which is used in the NN controller block in the online inverse model neural network which is developed in the previous section and uses the weights obtained from comprehensive offline training. The Simulink block diagram used to simulate the backstepping controller is the same block diagram used in the previous section. A trajectory tracking comparison with a linear reference trajectory with any external disturbance is shown in Figures 4-68:

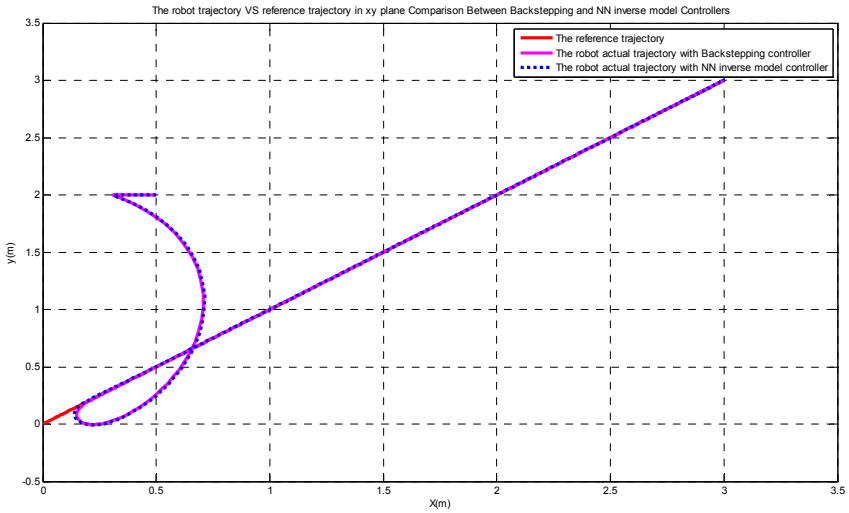

Figure 4-68: The robot trajectory in x-y plane comparison between Backstepping and NN inverse model controller

The robot trajectories are almost the same for both controllers because the backstepping controller alone can produce the required control inputs to the system when there is no disturbance in the system. The main purpose of the NN inverse controller is to make sure that the robot velocities will follow the velocities produces by the backstepping controller. The velocity errors of the system which shows the effect of the velocity control part of the system are shown in Figure 4-69:

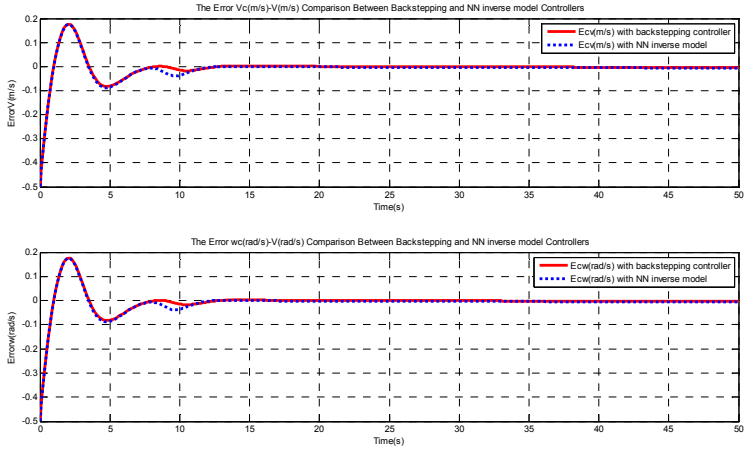

Figure 4-69: The output velocities Error response with time comparison between Backstepping and NN inverse model controller

The effect of the neural network velocity control is not very obvious in the above plots because it is working in addition to a PD controller which is enough for velocity control of a system without disturbance. The neural network performance and functionality can be check by looking at the change of its weights during the trajectory and relating them to the velocity error of the system which is the performance index of the neural network. The time response of the two different weights of the neural network is shown in Figure 4-70:

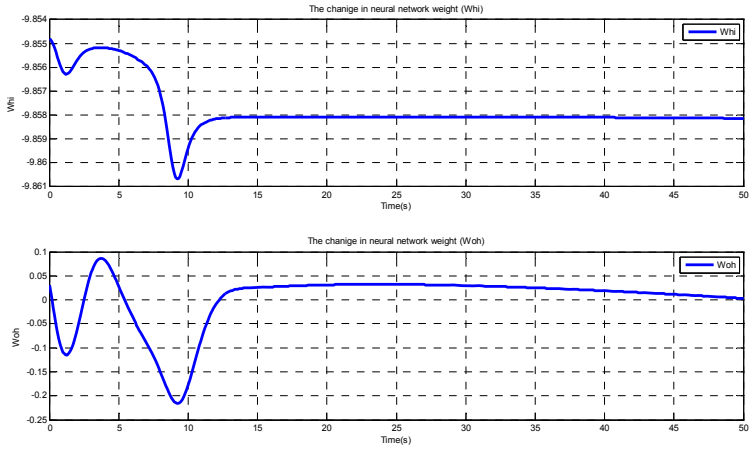

Figure 4-70: The Neural Network weights response with time

Looking at the above time response, one can find out that the neural network is learning and the gains will become steady and constant after the velocity errors of the system become zero. The other trajectory and robot initial condition that will give a better view on the controller performance is shown in Figure 4-71:

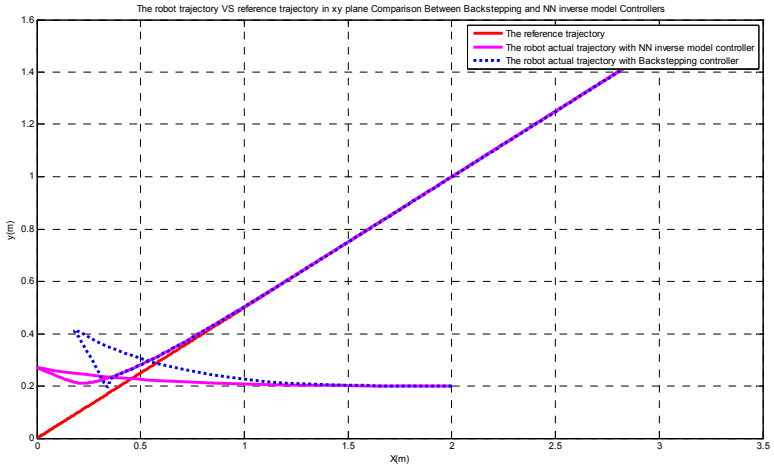

Figure 4-71: The robot trajectory in x-y plane comparison between Backstepping and NN inverse model controller

The time response of the two sample neural network weights is shown in Figure 4-72:

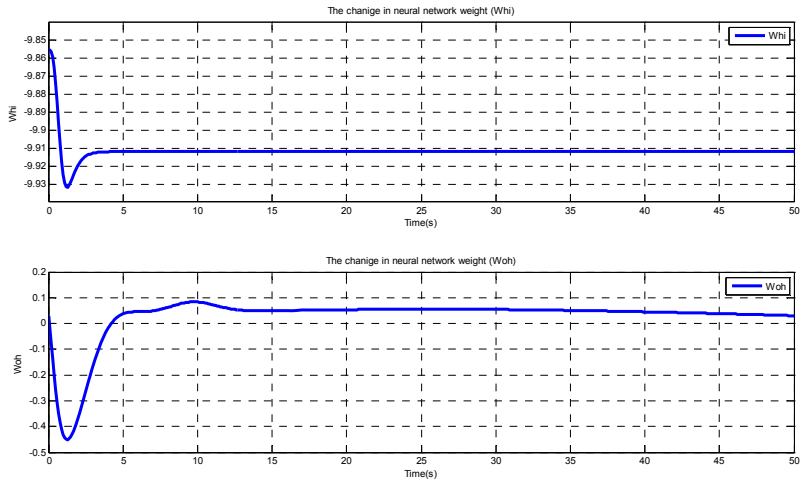

Figure 4-72: The Neural Network weights response with time

The learning performance of the neural network can be seen in the above figure. The next step of the performance analysis of this proposed controller is to introduce disturbance to the system and compare the performance with the backstepping controller. The disturbance is always in the shape of a torque to the robot wheel which can be caused from external forces like friction or it can be in the shape of velocity disturbance. The effect of both of the above mentioned disturbances will be shown in the following disturbance analysis of the system. The Simulink block diagram used to add disturbance to the system is shown in Figure 4-73:

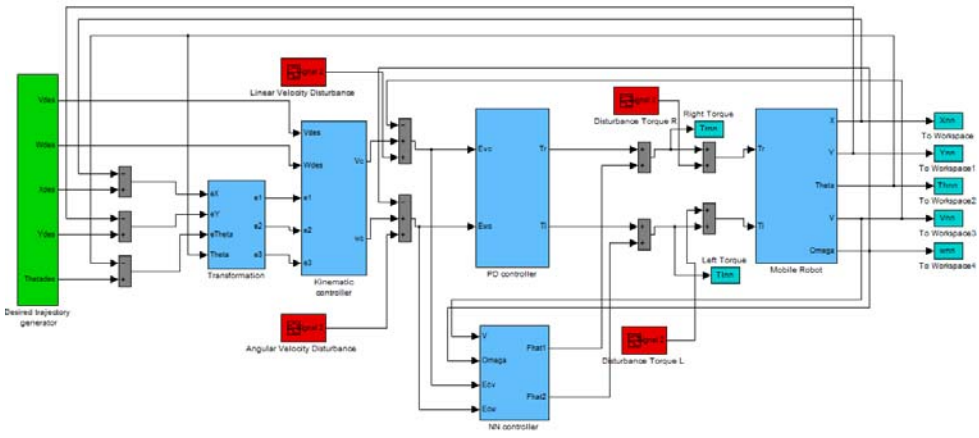

Figure 4-73: the Simulink block diagram of the NN inverse controller with disturbance

All the torque and velocity disturbances are added to the system with the NN inverse model controller as it can be seen from the above figure. The disturbances are generated

from the Matlab Simulink signal generator block. The backstepping controller Simulink block diagram with disturbance is shown in Figure 4-74:

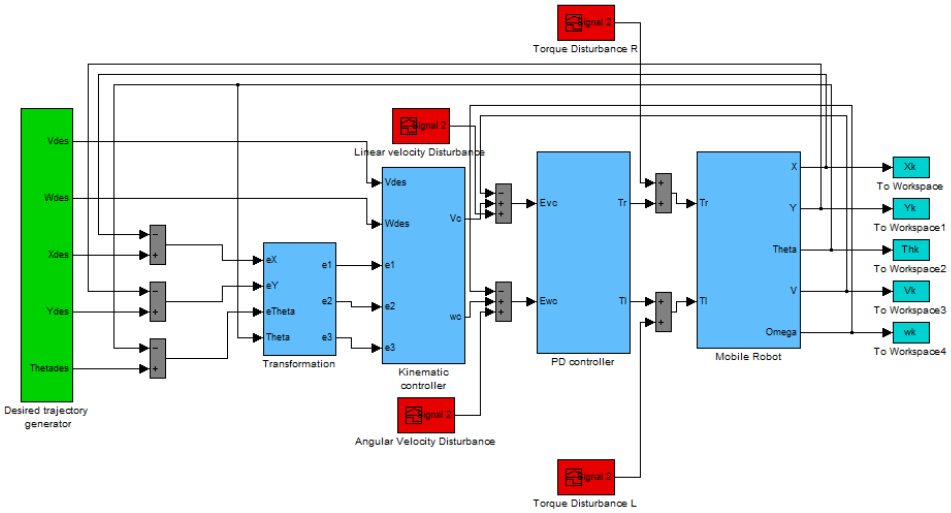

Figure 4-74: The Simulink block diagram of the backstepping controller with disturbance

The tracking and disturbance rejection performance of the NN inverse model controller in comparison with the backstepping controller is shown in Figure 4-75:

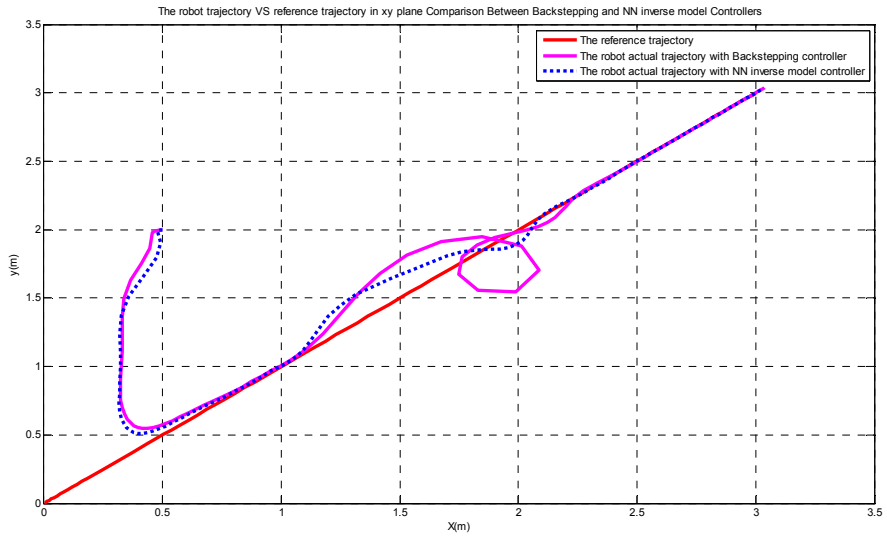

Figure 4-75: the robot trajectory in X-Y plane with disturbance

As it can be seen from the above figure, the NN inverse controller shows its disturbance rejection ability when it is compared to the backstepping controller. The disturbance rejection ability of this controller comes from the fact that the disturbances are occurring

in the velocity control loop and the neural network is acting on the velocity errors of the system. The system states errors are shown in Figure 4-76:

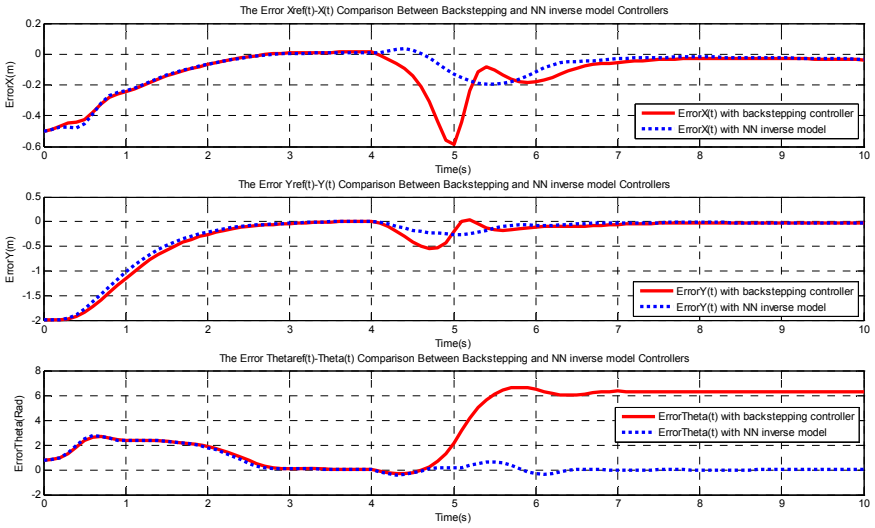

Figure 4-76: the system states errors with disturbance

The system velocity error time responses are shown in Figure 4-77:

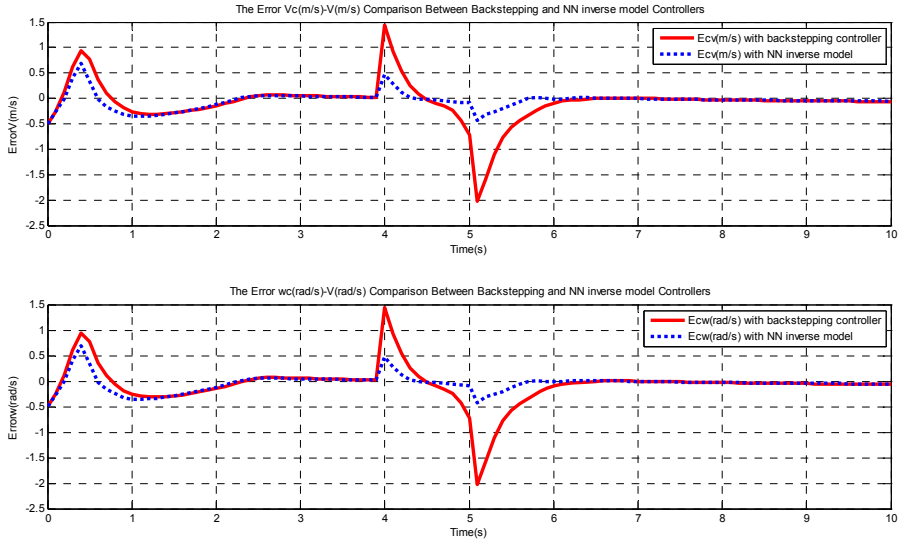

Figure 4-77: the velocity errors of the system with disturbance

The neural network weights time responses which are shown in Figure 4-78 can clear out the neural networks action on the system:

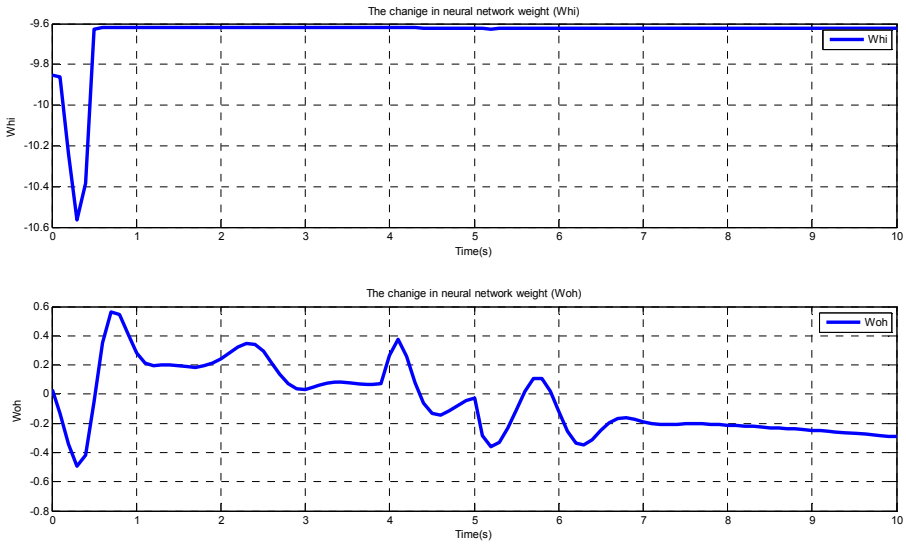

Figure 4-78: The Neural Network weights response with time with disturbance

The above comparison analysis between the neural network inverse model controller shows that this proposed controller has the following advantages over the backstepping controller:

- This controller can deal with unmodeled bounded disturbances and unstructured unmodeled dynamics in the vehicle which are a part of any environment that robot wants to perform the trajectory tracking.
- No knowledge of the system dynamics and parameters is needed in order to compensate for the nonlinear terms in the system. Especially when a readymade robot platform with unknown parameters and inner layers is used for the tracking task.

The other trajectory tracking controller that can be designed to improve the performance of the above controllers is the one that can adapt itself and the backstepping controller gains to each trajectory and robot location. Note that the gains of the backstepping controller in all of the above simulation performances are tuned according to each trajectory and robot initial location. In order to eliminate the backstepping controller gain tuning step which is a very time consuming and inefficient approach, an adaptive gain tuning controller using the neural network direct model in designed and implemented in the next section of this section.

4.3.3 The NN-based adaptive backstepping controller

The kinematic based controller or the so called backstepping controller proposed by Kanayama in 1992 is a stable tracking control rule for a nonholonomic mobile robot and

is explained thoroughly in section (4.1) of this chapter. This controller's structure is shown in Figure 4-79:

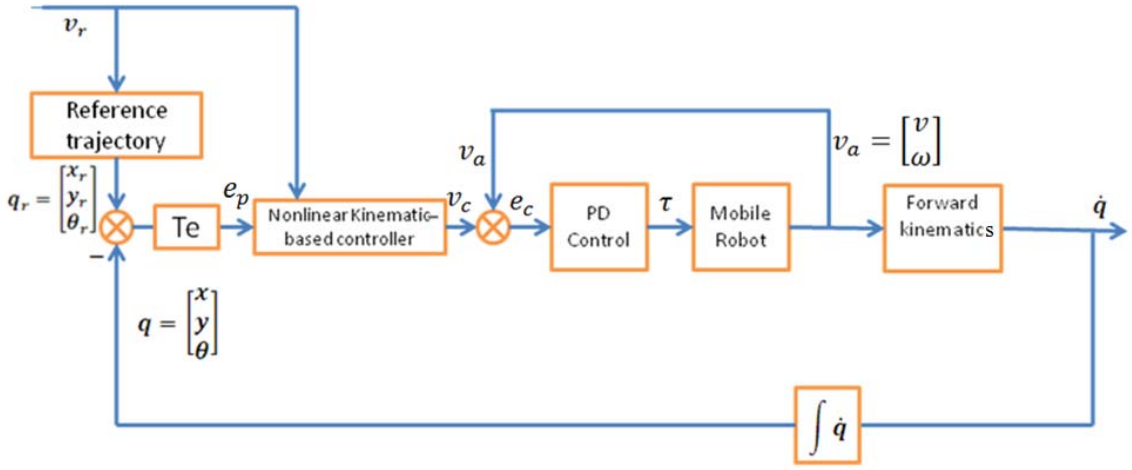

Figure 4-79: The backstepping controller structure

The input error to this controller is defined as follows:

$$e_p = \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = T_e(q_r - q) = T_e \times e_r$$
$$\begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix} \quad (4.165)$$

The control algorithm that calculates the robot velocity input vector v_c is shown in the following equation:

$$v_c = \begin{bmatrix} v_r cose_\theta + K_x e_x \\ \omega_r + K_y v_r e_y + K_\theta v_r sine_\theta \end{bmatrix}$$
$$v_c = f(e_p, v_r, K)$$
$$K = (K_x, K_y, K_\theta) \quad (4.166)$$

The additional PD controller block is to make sure that the robot velocities follow the input reference velocities and has the following equation:

$$\tau = K_P e_c + K_D \frac{de_c}{dt} \quad (4.167)$$

In which:

$$K_P = \begin{bmatrix} K_{P_R} & 0 \\ 0 & K_{P_L} \end{bmatrix} \quad (4.168)$$

As it can be seen in the above equation, the controller has three gains which are mentioned to be positive constant values in the other references. The disadvantage of the above controller with constant gains is that it needs a careful gain tuning for each different reference trajectory and it will not give zero trajectory error with a smooth tracking. The proposed control algorithm provides the backstepping controller with adaptive gains which are variable and change according to the reference trajectory. The NN-based adaptive backstepping controller structure is shown in Figure 4-80:

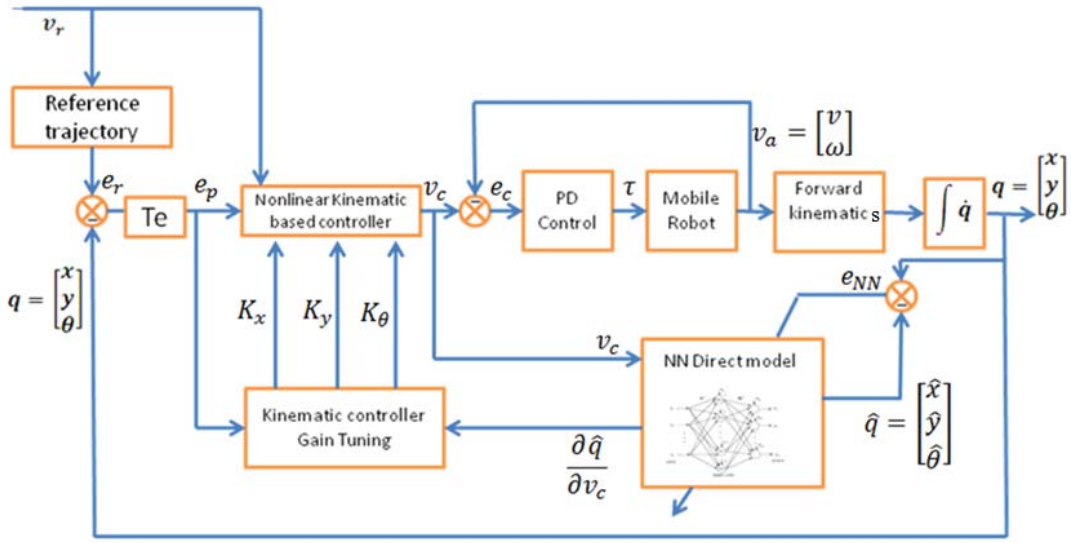

Figure 4-80: The NN-based adaptive backstepping controller structure

The above control structure adapts the kinematic based controller gains to minimize the following cost function:

$$J = \frac{1}{2}\sum \gamma_x {e_x}^2 + \gamma_y {e_y}^2 + \gamma_\theta {e_\theta}^2 \tag{4.169}$$

The kinematic model based controller gains are considered part of the above cost function and are optimized and updated according to the gradient descent method. The kinematic controller gains are represented by the set $\alpha = [K_x \quad K_y \quad K_\theta]$. The partial derivative of the cost function with respect to α is:

$$\frac{\partial J}{\partial \alpha} = \gamma_x e_x \frac{\partial e_x}{\partial \alpha} + \gamma_y e_y \frac{\partial e_y}{\partial \alpha} + \gamma_\theta e_\theta \frac{\partial e_\theta}{\partial \alpha} = \gamma {e_p}^T \frac{\partial e_p}{\partial \alpha}$$

$$\gamma = \begin{bmatrix} \gamma_x & 0 & 0 \\ 0 & \gamma_y & 0 \\ 0 & 0 & \gamma_\theta \end{bmatrix} \tag{4.170}$$

Substituting equation (4.165) in the above equation, we have:

$$e_p = \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = T_e\big(q_r - q\big)$$

$$\frac{\partial J}{\partial \alpha} = \gamma {e_p}^T \frac{\partial \big(T_e(q_r - q)\big)}{\partial \alpha} = -\gamma \times {e_p}^T \times T_e \times \frac{\partial q}{\partial \alpha} \tag{4.171}$$

Using the chain rule, the derivative $\frac{\partial q}{\partial \alpha}$ can be written as:

$$\frac{\partial q}{\partial \alpha} = \frac{\partial q}{\partial v_c} \times \frac{\partial v_c}{\partial \alpha} \tag{4.172}$$

The first derivative in the above equation $\frac{\partial q}{\partial v_c}$ can be defined as the Jacobian matrix with the system velocity inputs, as follows:

$$\frac{\partial q}{\partial v_c} = Jac_v = \begin{bmatrix} \frac{\partial x}{\partial v_c} & \frac{\partial x}{\partial \omega_c} \\ \frac{\partial y}{\partial v_c} & \frac{\partial y}{\partial \omega_c} \\ \frac{\partial \theta}{\partial v_c} & \frac{\partial \theta}{\partial \omega_c} \end{bmatrix} = \begin{bmatrix} Jac_{v_{11}} & Jac_{v_{12}} \\ Jac_{v_{21}} & Jac_{v_{22}} \\ Jac_{v_{31}} & Jac_{v_{32}} \end{bmatrix} \tag{4.173}$$

The second derivative in equation (4.172) $\frac{\partial v_c}{\partial \alpha}$ is calculated according to equation (4.166) as follows:

$$\frac{\partial v_c}{\partial \alpha} = \begin{bmatrix} \frac{\partial v_c}{\partial K_x} & \frac{\partial v_c}{\partial K_y} & \frac{\partial v_c}{\partial K_\theta} \\ \frac{\partial \omega_c}{\partial K_x} & \frac{\partial \omega_c}{\partial K_y} & \frac{\partial \omega_c}{\partial K_\theta} \end{bmatrix} = \begin{bmatrix} e_x & 0 & 0 \\ 0 & v_r e_y & v_r sine_\theta \end{bmatrix} \tag{4.174}$$

Therefore, the required derivative in equation (4.172) can be found by substituting equations (4.173) and (4.174) in (4.172):

$$\frac{\partial q}{\partial \alpha} = Jac_v \times \begin{bmatrix} e_x & 0 & 0 \\ 0 & v_r e_y & v_r sine_\theta \end{bmatrix} \tag{4.175}$$

The derivative of the cost function with respect to the controller gains $\frac{\partial J}{\partial \alpha}$ is:

$$\frac{\partial J}{\partial \alpha} = \begin{bmatrix} \frac{\partial J}{\partial K_x} & \frac{\partial J}{\partial K_y} & \frac{\partial J}{\partial K_\theta} \end{bmatrix} = -\gamma \times {e_p}^T \times T_e \times Jac_v \times \begin{bmatrix} e_x & 0 & 0 \\ 0 & v_r e_y & v_r sine_\theta \end{bmatrix} \tag{4.176}$$

In which:

$$\gamma = \begin{bmatrix} \gamma_x & 0 & 0 \\ 0 & \gamma_y & 0 \\ 0 & 0 & \gamma_\theta \end{bmatrix} \tag{4.177}$$

$${e_p}^T = [e_x \quad e_y \quad e_\theta] \tag{4.178}$$

$$T_e = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.179}$$

$$Jac_v = \begin{bmatrix} \frac{\partial x}{\partial v_c} & \frac{\partial x}{\partial \omega_c} \\ \frac{\partial y}{\partial v_c} & \frac{\partial y}{\partial \omega_c} \\ \frac{\partial \theta}{\partial v_c} & \frac{\partial \theta}{\partial \omega_c} \end{bmatrix} = \begin{bmatrix} Jac_{v11} & Jac_{v12} \\ Jac_{v21} & Jac_{v22} \\ Jac_{v31} & Jac_{v32} \end{bmatrix} \tag{4.180}$$

Therefore, the kinematic controller gains will change and adapt to make the cost function zero according to the gradient descent method as follows:

$$K_x = K_x + \Delta K_x \tag{4.181}$$

$$K_y = K_y + \Delta K_y \tag{4.182}$$

$$K_\theta = K_\theta + \Delta K_\theta \tag{4.183}$$

In which the change in the gains is computed according to the following equations:

$$\Delta K_x = -\eta_{K_x} \frac{\partial J}{\partial K_x} \tag{4.184}$$

$$\Delta K_y = -\eta_{K_y} \frac{\partial J}{\partial K_y} \tag{4.185}$$

$$\Delta K_\theta = -\eta_{K_\theta} \frac{\partial J}{\partial K_\theta} \tag{4.186}$$

The parameters η_{K_x} , η_{K_y} and η_{K_θ} are the learning rates of the gradient descent algorithm. The important part of calculating the derivatives of the equation (4.175) is how to compute the Jacobian matrix of the system. The Jacobian matrix can be calculated using the exact equations of the system or it can be provided by the neural network direct model. Calculating the Jacobian matrix based on the above two methods and their advantages and restrictions will be explained in the following two sections.

4.3.3.1 Jacobian calculation using the system's exact equations

The Jacobian matrix can be calculated using the robots governing dynamic and kinematic equations. The derivative $\frac{\partial q}{\partial v_c}$ can be expanded as follows:

$$Jac_v = \frac{\partial q}{\partial v_c} = \frac{\partial q}{\partial v_a} \times \frac{\partial v_a}{\partial \tau} \times \frac{\partial \tau}{\partial e_c} \times \frac{\partial e_c}{\partial v_c} \tag{4.187}$$

To calculate the first derivative in the above equation, we use the robots kinematic equation as follows:

$$\dot{q} = S v_a = \begin{bmatrix} cos\theta & -asin\theta \\ sin\theta & acos\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{4.188}$$

Integrating the above equation over time, we have:

$$q = \int \dot{q} dt = \int S v_a dt \tag{4.189}$$

Therefore the required derivative is:

$$\frac{\partial q}{\partial v_a} = \int Sdt = \int \begin{bmatrix} cos\theta & -asin\theta \\ sin\theta & acos\theta \\ 0 & 1 \end{bmatrix} dt \tag{4.190}$$

The second derivative can be found from the NN inverse model controller derivation which has been done in the previous section of this chapter:

$$\frac{\partial v_a}{\partial \tau} = Jac_\tau = \begin{bmatrix} \frac{1}{m}t & 0 \\ 0 & \frac{1}{I_C + ma^2}t \end{bmatrix} \tag{4.191}$$

The fourth derivative can be calculated according to the following equation:

$$\tau = K_P e_c \tag{4.192}$$

The derivate part of the PD controller is neglected in the above equation because it is very small. This will simplify the analysis as follows:

$$\frac{\partial \tau}{\partial e_c} = K_P = \begin{bmatrix} K_{P_R} & 0 \\ 0 & K_{P_L} \end{bmatrix} \tag{4.193}$$

The fifth derivative is:

$$\frac{\partial e_c}{\partial v_c} = \frac{\partial (v_c - v_a)}{\partial v_c} = 1 \tag{4.194}$$

The required Jacobian matrix can be found by replacing equations (4.188) to (4.194) in (4.187):

$$Jac_v = \int Sdt \times Jac_\tau \times \begin{bmatrix} K_{P_R} & 0 \\ 0 & K_{P_L} \end{bmatrix} \tag{4.195}$$

As it can be seen in the above equation, calculation of the Jacobian using this method requires the exact knowledge of all system parameters. It is also assumed that all of the system parameters are certain and fixed which is not applicable to real life robot platforms which are subjected to uncertainty in parameters. Calculation of the Jacobian matrix using the neural network direct model approach solves this problem because the neural network will learn the real robot's model online and gives us a better

approximation of the system. Details about Jacobian calculation using the neural network direct model will be explained in the next section.

4.3.3.2 Jacobian calculation using the neural network direct model

The neural network which is used to approximate the robot platform or so called the direct model neural network is shown in Figure 4-81:

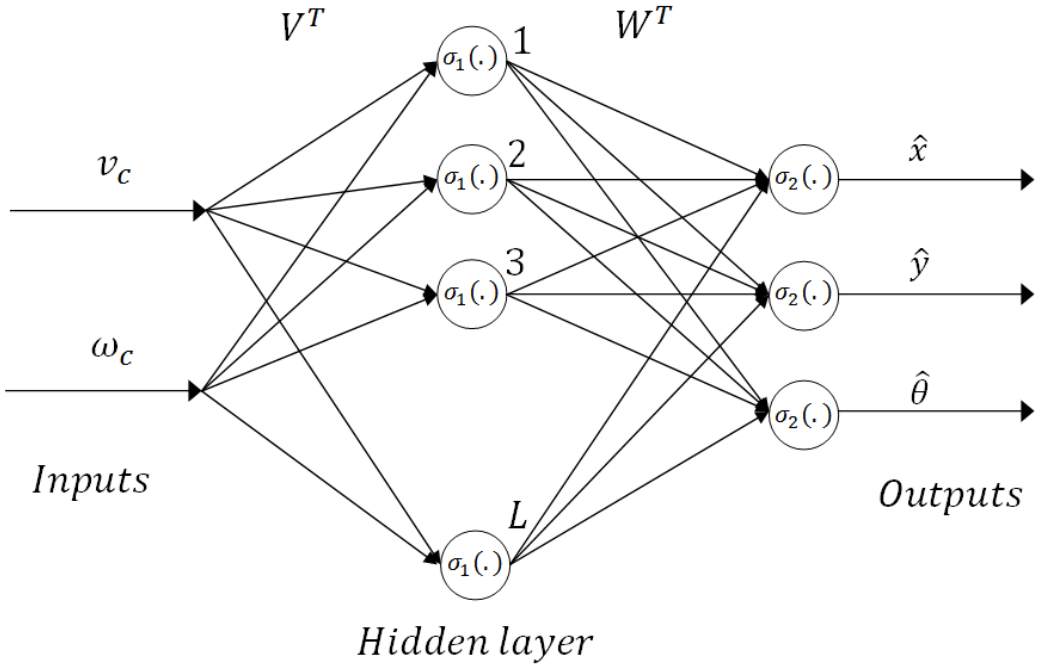

Figure 4-81: The direct model neural network

The Jacobian matrix that can be found from the above neural network is:

$$\frac{\partial \hat{q}}{\partial v_c} = Jac_v = \begin{bmatrix} \frac{\partial \hat{x}}{\partial v_c} & \frac{\partial \hat{x}}{\partial \omega_c} \\ \frac{\partial \hat{y}}{\partial v_c} & \frac{\partial \hat{y}}{\partial \omega_c} \\ \frac{\partial \hat{\theta}}{\partial v_c} & \frac{\partial \hat{\theta}}{\partial \omega_c} \end{bmatrix} = \begin{bmatrix} Jac_{v_{11}} & Jac_{v_{12}} \\ Jac_{v_{21}} & Jac_{v_{22}} \\ Jac_{v_{31}} & Jac_{v_{32}} \end{bmatrix} \tag{4.196}$$

As it is mentioned in the neural network introduction section, the equation relating the inputs and outputs of this network is:

$$y_i = \sigma\left(\sum_{l=1}^{L} W_{il}\sigma\left(\sum_{j=1}^{n} V_{lj}x_j + v_{l0}\right) + w_{i0}\right) \quad i = 1,2,\dots,m \tag{4.197}$$

The input to layer one is x_j. Define the input to layer two as:

$$z_l = \sigma\left(\sum_{j=1}^{n} V_{lj} x_j + v_{l0}\right) \quad l = 1,2,\dots,L \tag{4.198}$$

The thresholds can more easily be dealt with by defining $x_0 = 1$ and $z_0 = 1$. Then one can say:

$$y_i = \sigma\left(\sum_{l=1}^{L} W_{il}\, z_l\right) \tag{4.199}$$

$$z_l = \sigma\left(\sum_{j=1}^{n} V_{lj} x_j\right) \tag{4.200}$$

Using the chain rule, the required derivative for the Jacobian matrix is:

$$\frac{\partial y_i}{\partial x_j} = \frac{\partial y_i}{\partial z_l} \times \frac{\partial z_l}{\partial x_j} \tag{4.201}$$

The first derivative can be calculated according to equation (4.199):

$$\frac{\partial y_i}{\partial z_l} = W_{il} \times \acute{\sigma}\left(\sum_{l=1}^{L} W_{il}\, z_l\right) \tag{4.202}$$

The second derivative is calculated according to equation (4.200):

$$\frac{\partial z_l}{\partial x_j} = V_{lj} \times \acute{\sigma}\left(\sum_{j=1}^{n} V_{lj} x_j\right) \tag{4.203}$$

Therefore, the derivative of the equation (4.201) is:

$$\frac{\partial y_i}{\partial x_j} = W_{il} \times \acute{\sigma}\left(\sum_{l=1}^{L} W_{il}\, z_l\right) \times V_{lj} \times \acute{\sigma}\left(\sum_{j=1}^{n} V_{lj} x_j\right) \tag{4.204}$$

Equation (4.204) is the equation that can be used to compute the Jacobian matrix using the direct model neural network. The neural network to be used in this approach should

be well trained so it can perform a precise approximation of the system. The details of the neural network training and implementation will be explained in the simulation and results section. The simulation results of the above control algorithm are shown in detail in the next section.

4.3.3.3 Simulation results and analysis

The Simulink block diagram shown in Figure 4-82 is used to simulate the response of the controller using the neural network direct model approach:

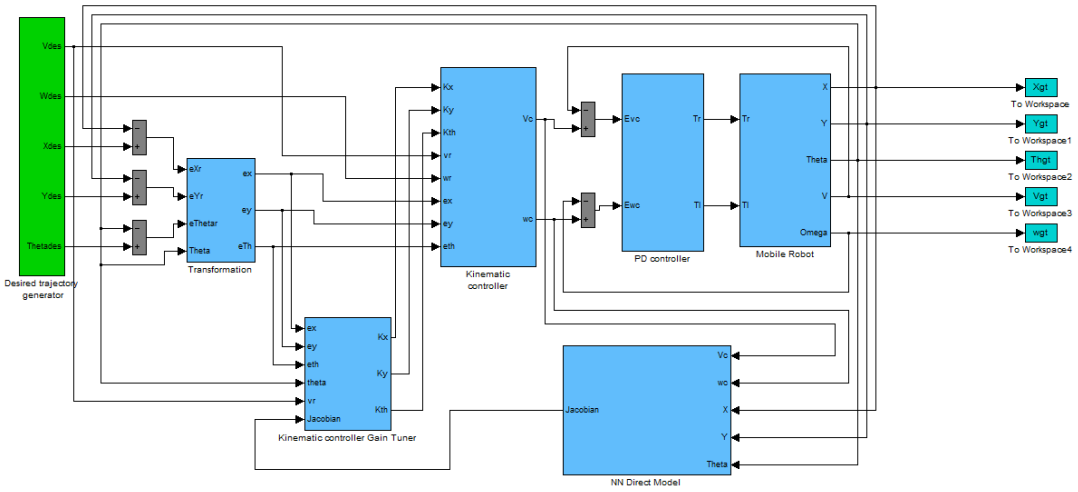

Figure 4-82: The Simulink block diagram used to simulate the response of the gain tuning controller

The NN direct model block in the above block diagram is responsible to approximate the mobile robot model and has the block diagram shown in Figure 4-83 inside:

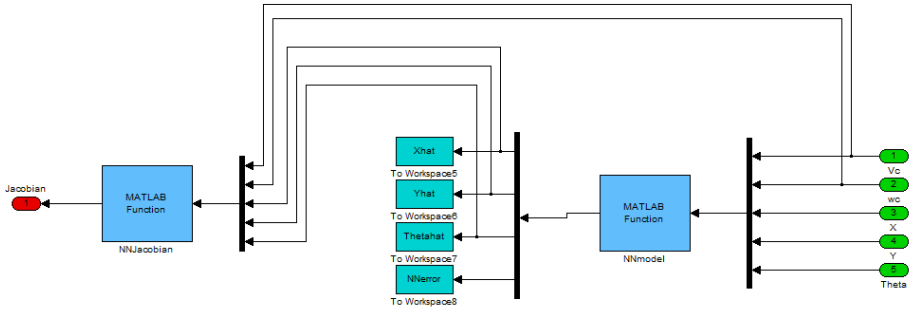

Figure 4-83: The NN direct model block

The NN model and NN Jacobian blocks in the above block diagram are two Matlab functions responsible to perform the neural network direct modeling and Jacobian calculation according to equation (4.204). Comprehensive neural network training should

be done to make the NN direct model error minimum and increase the precision of the calculated Jacobian. The different steps that have been done to train the neural network are explained thoroughly in next section.

4.3.3.3.1 Direct model neural network offline and online training

The following steps are taken to train the neural network and make it ready for online approximation of the system model:

- Offline training of the neural network with sinusoidal and chirp inputs with different frequencies to excite all the modal frequencies of the system.
- Testing the online learning performance of the neural network with the network weights obtained from the offline training and comparing it with the real robot model outputs.

The Simulink block diagram shown in Figure 4-84 is used to generate the required data to perform the offline training of the network with the sinusoidal input:

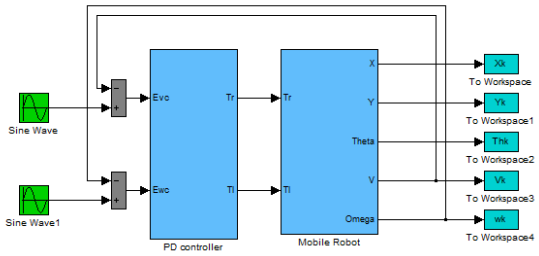

Figure 4-84: The Simulink model to generate the required data for neural network training with sinusoidal input

The PD controller and the mobile robot model are used to generate the required data so w can have the approximation of the exact model of the system. The input sinusoidal signals used to generate the data are shown in Figure 4-85:

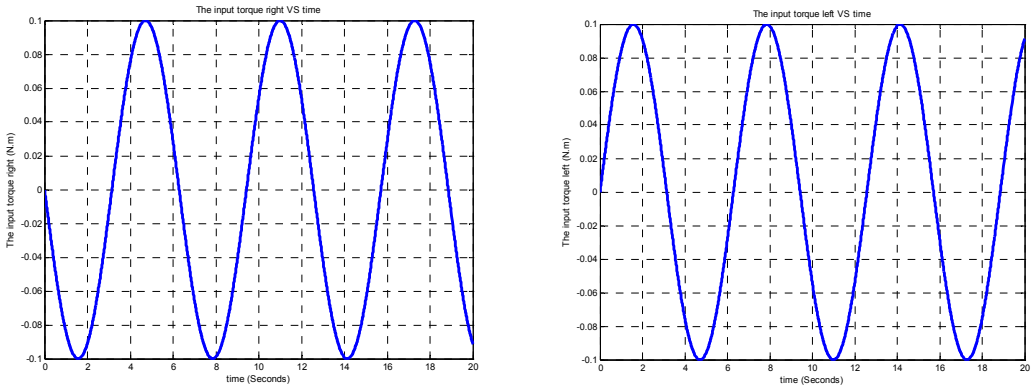

Figure 4-85: The sinusoidal input to the robot system

The robot model outputs with the above sinusoidal signal as the input are shown in Figure 4-86:

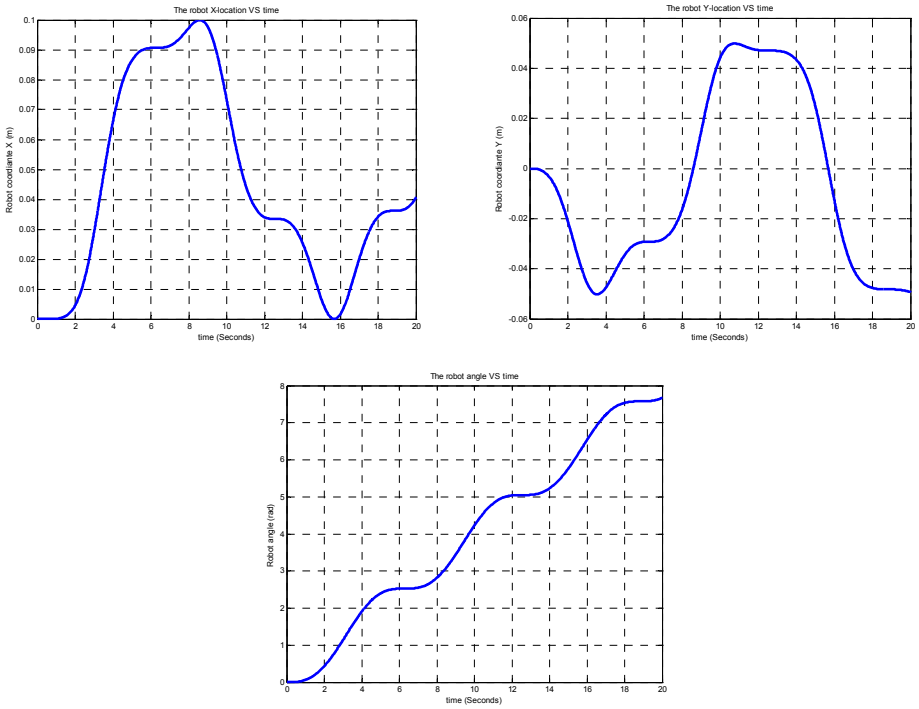

Figure 4-86: The robot output states X (t), Y (t) and Theta (t) with the sinusoidal input

The neural network which is used to approximate the robot model is shown in Figure 4-87:

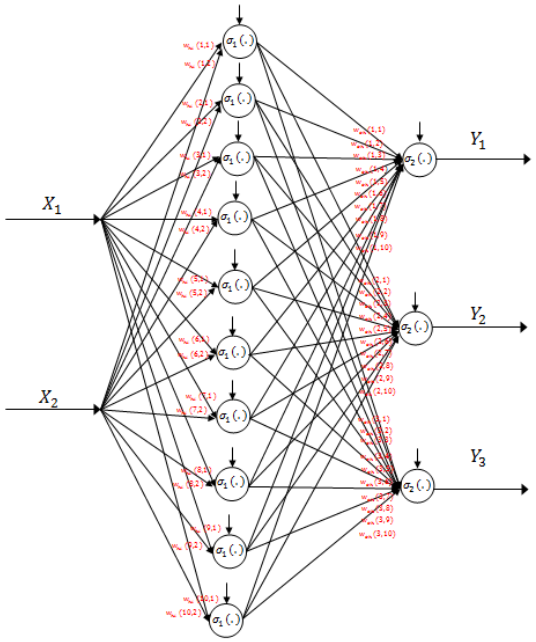

Figure 4-87: The neural network to approximate the robot direct model

The parameters used to implement the above neural network for offline training of the system are as follows:

$$N_h = 20 : Number\ of\ neurons\ in\ the\ hidden\ layer\ of\ the\ Neural\ Network$$

$$\eta = 0.3 \ : \ The\ learning\ rate\ for\ the\ back - propagation\ algorithm$$

$$\beta = 0.7 : \ The\ momentum\ rate\ for\ the\ back - propagation\ algorithm$$

$$N_c = 200 : \ The\ number\ of\ cycles\ the\ set\ of\ input\ data\ will\ repeat$$

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
$$: The\ hyper\ tangent\ activation\ function\ for\ the\ hidden\ layer$$

$$f_1(x) = x : The\ linear\ activation\ function\ for\ the\ output\ layer$$

The Matlab code used to perform the offline training with the above parameters is included in the appendix. The neural network mean square error after 200 cycles of training is shown in Figure 4-88:

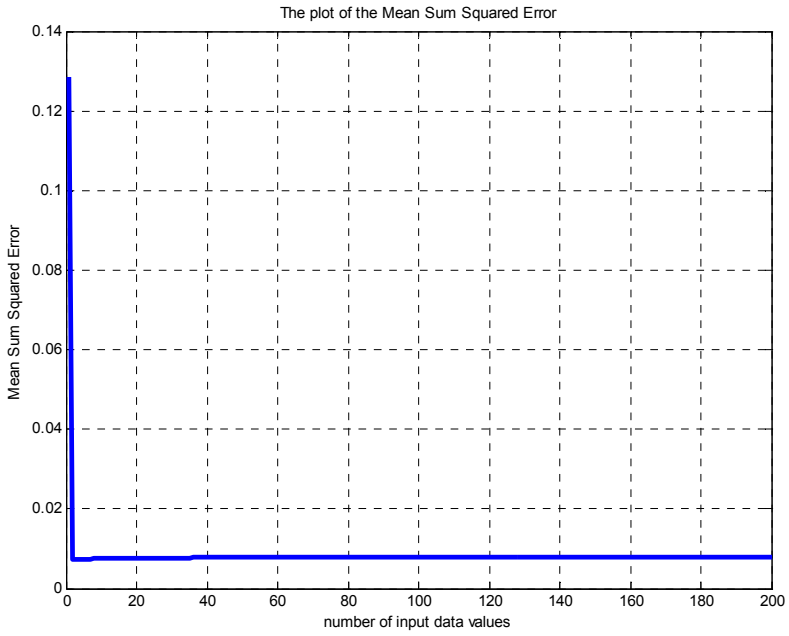

Figure 4-88: The mean squared error plot for 200 cycles (sinusoidal input)

The mean square error is at the minimum value after 200 cycles of training which shows that the number of cycles used to train the network is enough and the approximation performance should be satisfactory. The neural network outputs and the robot model's output are shown in Figures 4-89 to 4-91:

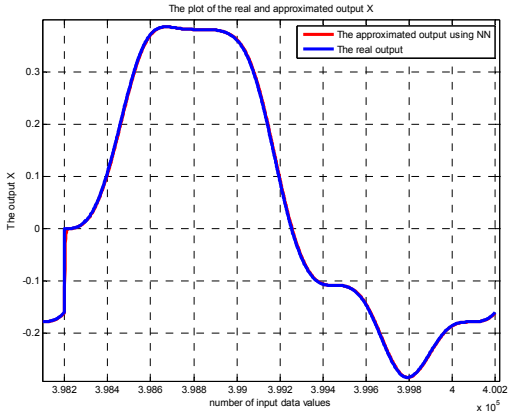

Figure 4-89: The outputs X of the model (BLUE) and the NN (RED) for the last cycle of learning (sinusoidal input)

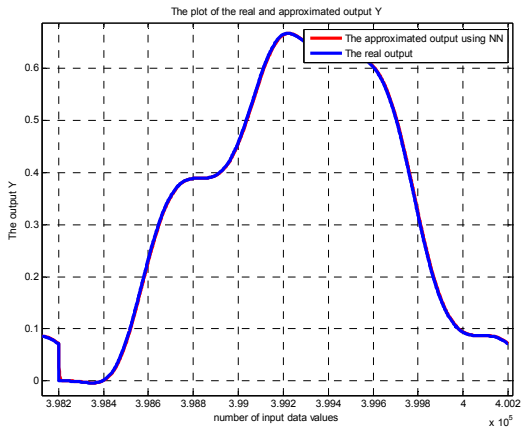

Figure 4-90: The output Y of the model (BLUE) and the NN (RED) for the last cycle of learning (sinusoidal input)

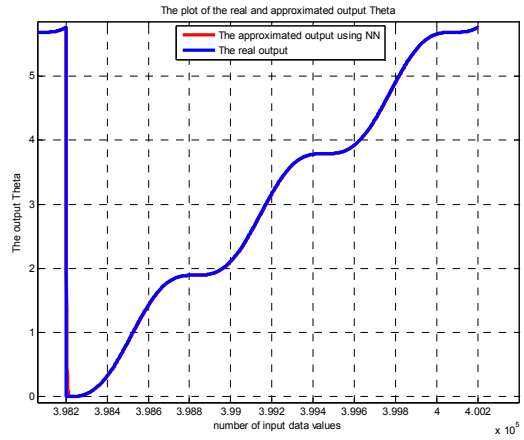

Figure 4-91: The output THETA of the model (BLUE) and the NN (RED) for the last cycle of learning (sinusoidal input)

Perfect estimation is achieved after this number of offline training as it can be seen from the above figures. The next step is to do the offline training using the chirp signal as the input to the robot system. The Simulink block diagram shown in Figure 4-92 is used to generate the required data to do the offline training using the chirp input:

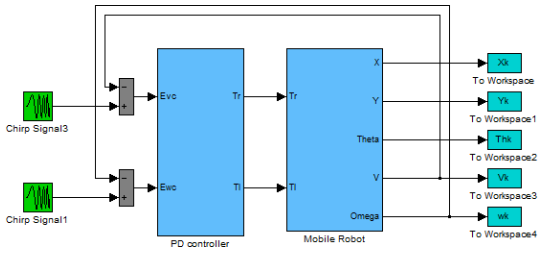

Figure 4-92: The Simulink block diagram used to generate the data for the offline training with the chirp input

The input chirp signals used to generate the data are shown in Figure 4-93:

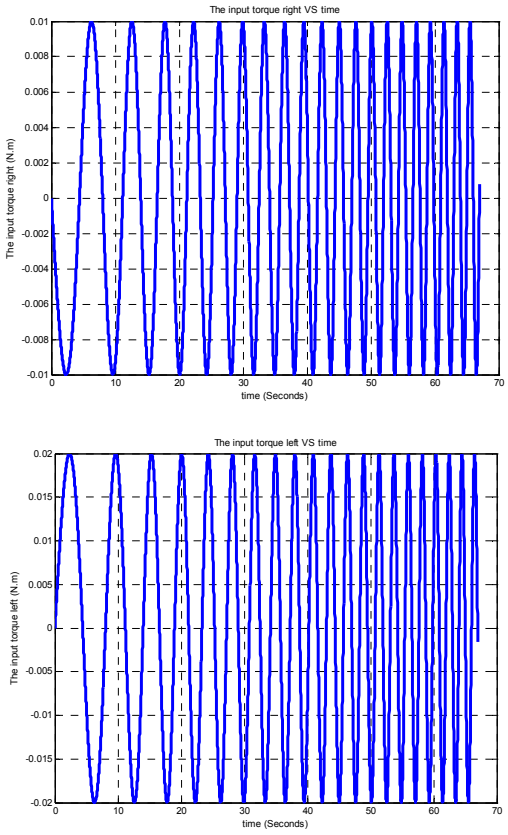

Figure 4-93: The input chirp signal to the robot model

The frequencies of the above chirp signal vary between 0.1 to 1 hertz. The robot model outputs with the above chirp signal as the input are shown in Figure 4-94:

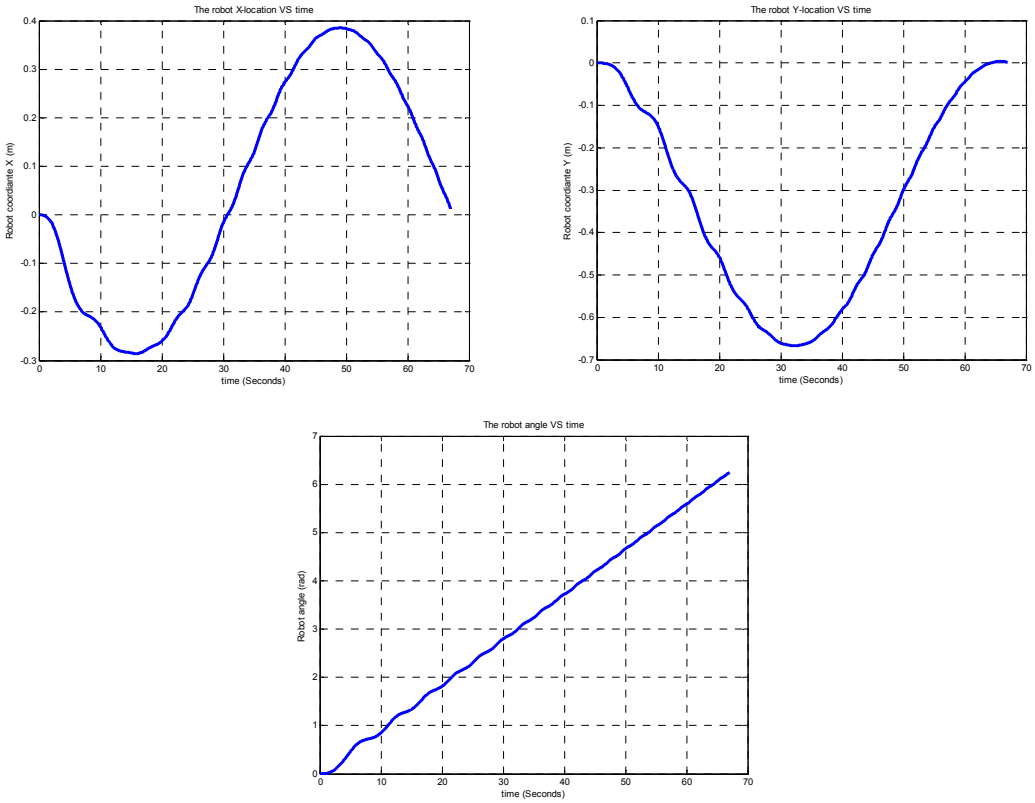

Figure 4-94: The robot output states X (t), Y (t) and Theta (t) with the chirp input

The same neural network structure and parameters are used to perform the offline training on the above data. The only difference is that the trained network with the sinusoidal inputs is used to do the training with the chirp input. The neural network mean square error after 200 cycles of training is shown in Figure 4-95:

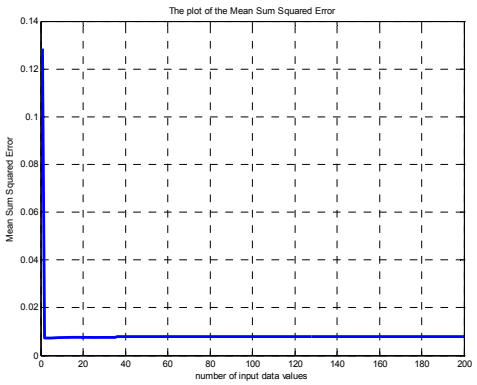

Figure 4-95: The mean squared error plot for 200 cycles (chirp input)

The mean square error is at the minimum value after 200 cycles of training which shows that the number of cycles used to train the network is enough and the approximation

performance should be satisfactory. The neural network outputs and the robot model's output are shown in Figures 4-96 to 4-98:

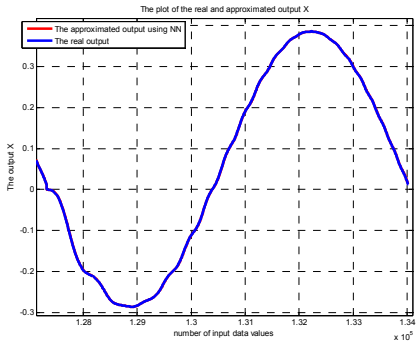

Figure 4-96: The output X of the model (BLUE) and the NN (RED) for the last cycle of learning (chirp input)

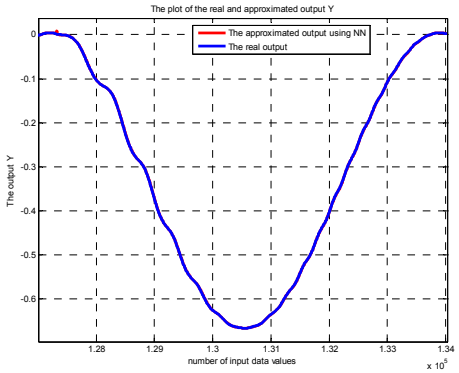

Figure 4-97: The output Y of the model (BLUE) and the NN (RED) for the last cycle of learning (chirp input)

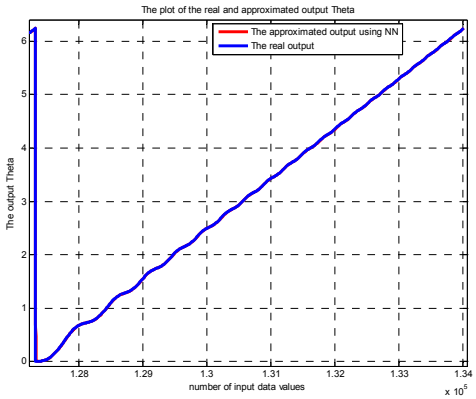

Figure 4-98: The output Theta of the model (BLUE) and the NN (RED) for the last cycle of learning (chirp input)

The next step of the neural network preparation for the direct modeling purpose is to test the offline trained network in online learning and check if it can result a precise approximation when it is learning the system online. The Simulink model shown in Figure 4-99 is used to check the online learning performance of the trained neural network and compare it with the actual outputs of the robot system:

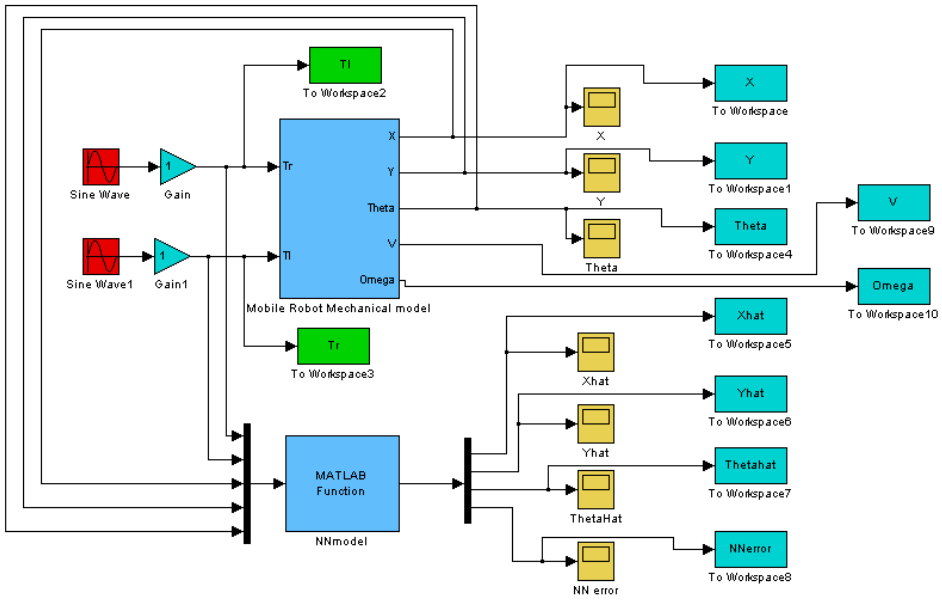

Figure 4-99: The Simulink block diagram used to check the online learning performance of the neural network

The above Simulink block diagram sends the same sinusoidal input signal to the robot model and neural network and compares their outputs. The same neural network parameters used for offline training are used for the online training. The only difference is that the weights used in the online neural network are the ones obtained from the comprehensive offline training explained above. The squared error time response of the online neural network is shown in Figure 4-100:

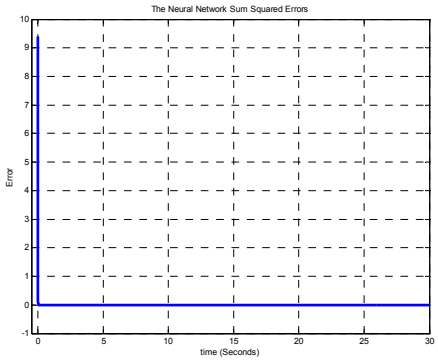

Figure 4-100: Sum squared error

As it can be seen from the above figure, the approximation error converges to the minimum value very fast which shows the perfect learning performance of the online neural network. The robot model outputs and the online neural network outputs are shown in Figures 4-101 to 4-103 to check the approximation performance:

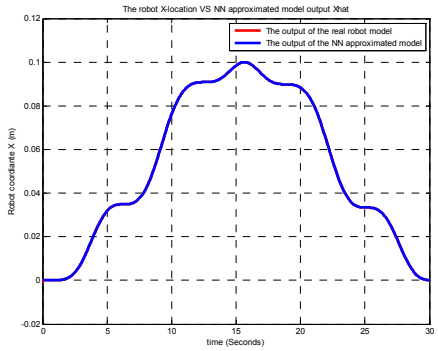

Figure 4-101: the robot X position VS NN model output X (online training)

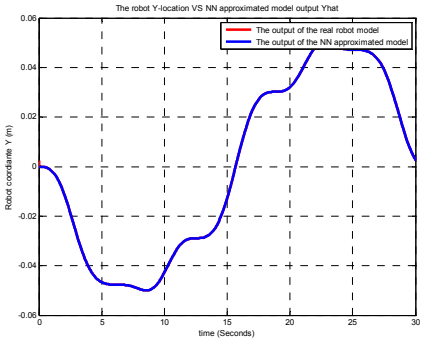

Figure 4-102: the robot Y position VS NN model output Y (online training)

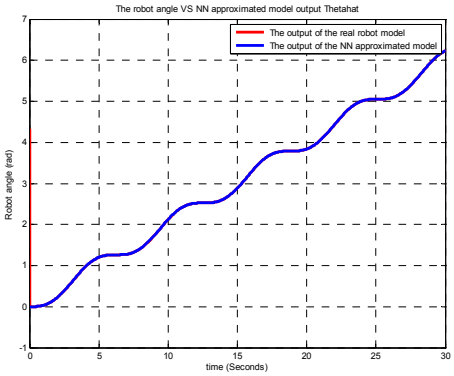

Figure 4-103: the robot Theta position VS NN model output Theta (online training)

As it can be seen from the above figures, the neural network is well trained and can perfectly approximate mobile robot model online. This trained neural network is the NN

model block in block diagram of figure (4-99) and will calculate the Jacobian matrix using equation (4.204).

4.3.3.4 Comparative analysis for NN-based adaptive backstepping controller and backstepping controller

After training the direct model neural network, we can simulate the performance of the adaptive gain tuning controller using the Simulink block diagram of figure (4-82). Comparison analysis between backstepping controller and the NN-based adaptive backstepping controller is done to show the advantages of the proposed controller. The following features of the controllers should be discussed to have a perfect comparison:

- The trajectory tracking error
- Smoothness of the robot trajectories
- Trajectory reach time
- Magnitude of the control action

Comparison between the performances of the controllers in the above categories results in a comprehensive analysis and shows the advantages and restrictions of each controller. The simulation results of the backstepping controller with fixed controller gains and adaptive gain tuning controller in response to different reference trajectories and robot initial positions will be shown in the remaining of this section and conclusion will be made about the performance of each controller.

Note that for the simulation and experimental implementation of this control algorithm, the sign of the Jacobian matrix is used.

Linear reference trajectory

Robot initial location: (0, 1, 0)

Backstepping controller gains: $K_x = 1, K_y = 55, K_{th} = 15$

Gains of the adaptive controller cost function:

$$J = \frac{1}{2}\sum \gamma_x {e_x}^2 + \gamma_y {e_y}^2 + \gamma_\theta {e_\theta}^2$$

$$\gamma_x = 1, \gamma_y = 50, \gamma_\theta = 1$$

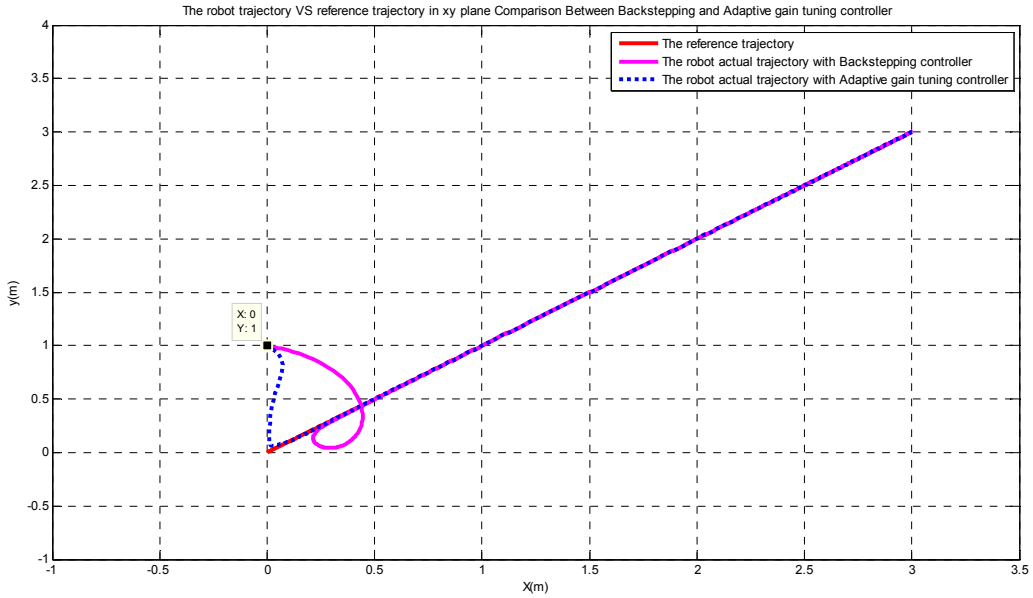

Figure 4-104: The robot trajectory in x-y plane, Comparison between the Backstepping controller and adaptive gain tuning controller (Linear reference trajectory)

According to the above figure, the trajectory tracking error and the trajectory reach time of the adaptive gain tuning controller is much smaller and the robot trajectory is much smoother in comparison with the backstepping controller. The time response of the robot output states $x(t), y(t)$ and $\theta(t)$, the trajectory errors $e_x(t), e_y(t)$ and $e_\theta(t)$, the robot velocities $v(t)$ and $\omega(t)$, the robot velocity errors $e_v(t)$ and $e_\omega(t)$, the controller output torques to the right and left robot wheels $T_r(t)$ and $T_l(t)$ and the adaptive gain tuning controller gains$K_x(t), K_y(t)$ and $K_\theta(t)$ are shown in the following figures.

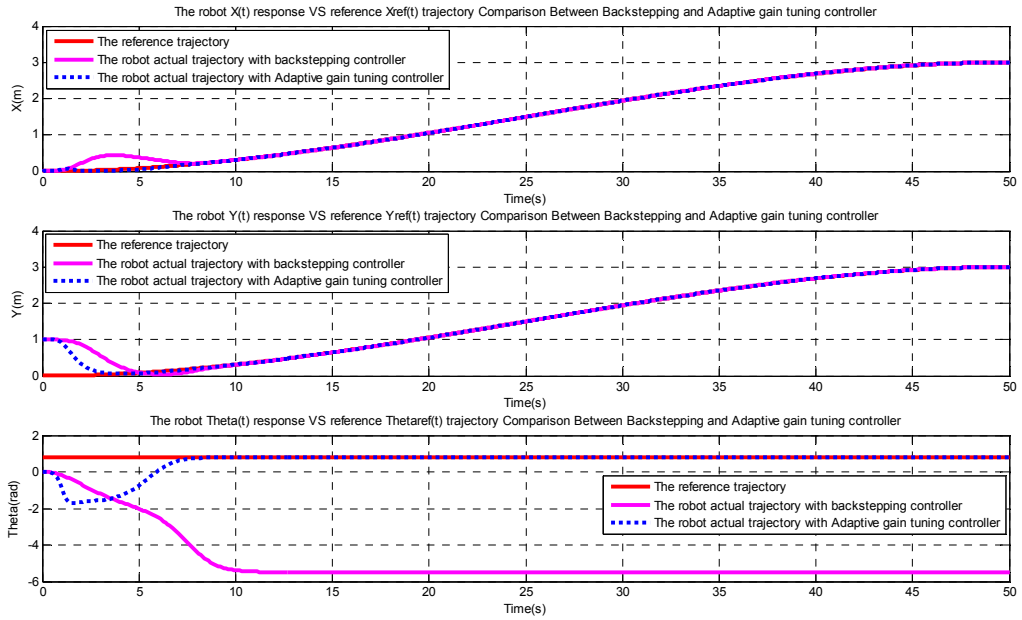

Figure 4-105: The robot output states time response vs. reference trajectories, Comparison between the Backstepping controller and adaptive gain tuning controller (Linear reference trajectory)

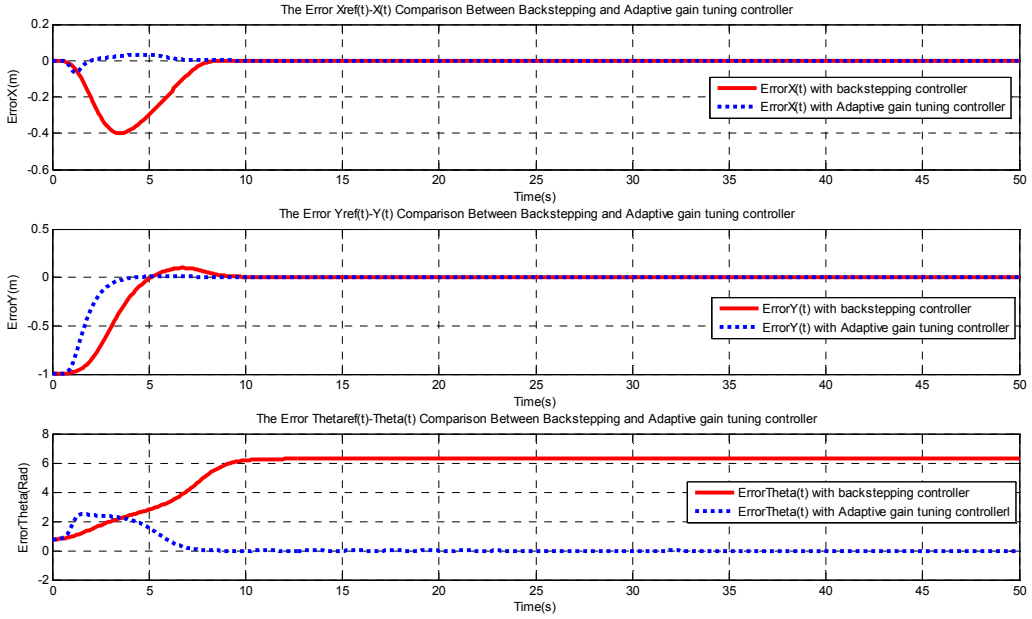

Figure 4-106: The robot output states errors Ex, Ey and Eth, Comparison between the Backstepping controller and adaptive gain tuning controller (Linear reference trajectory)

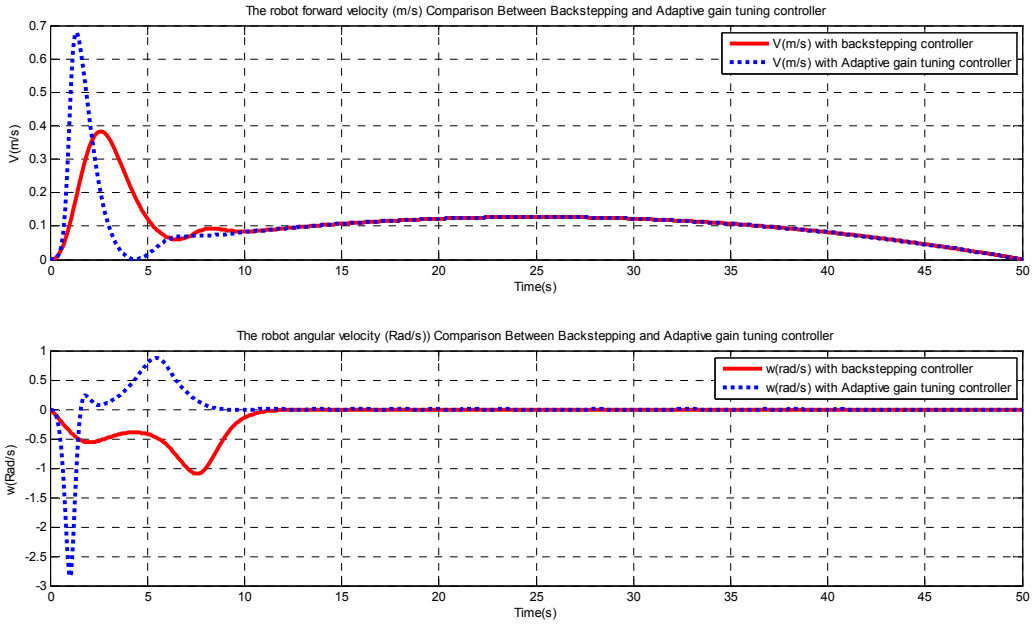

Figure 4-107: The robot linear and angular velocities, Comparison between the Backstepping controller and adaptive gain tuning controller (Linear reference trajectory)

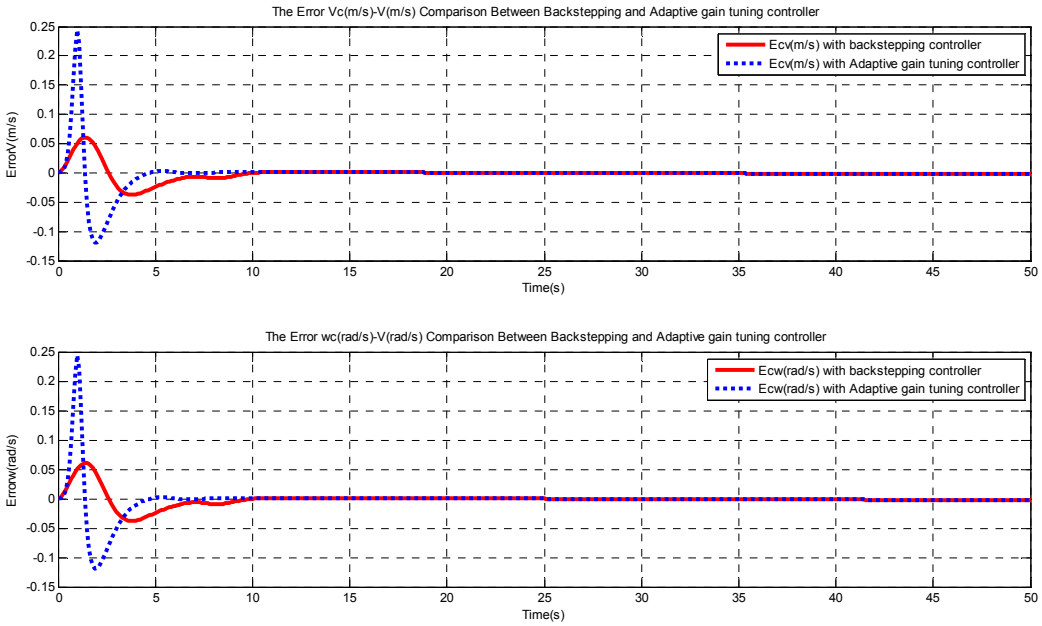

Figure 4-108: The robot velocity errors, Comparison between the Backstepping controller and adaptive gain tuning controller (Linear reference trajectory)

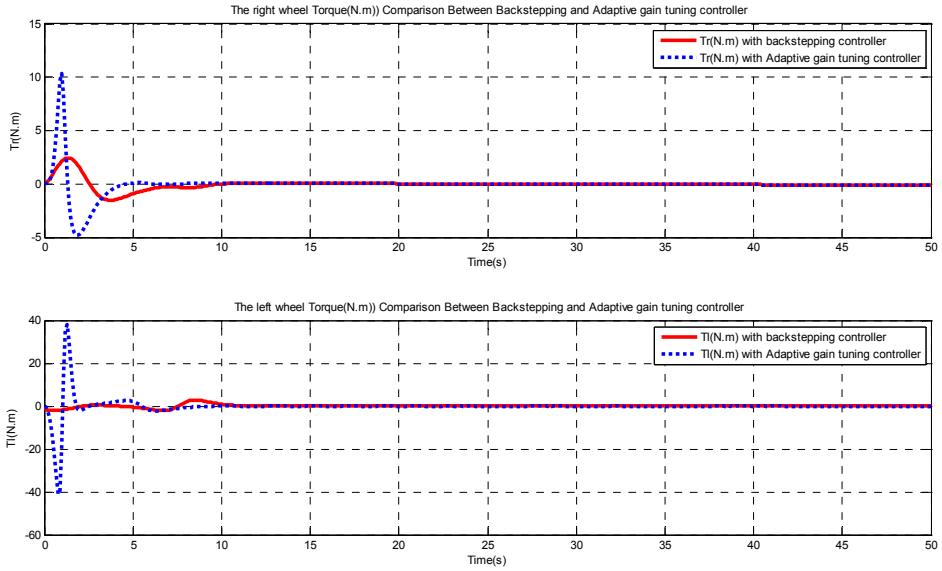

Figure 4-109: The rights and left wheel torques, Comparison between the Backstepping controller and adaptive gain tuning controller (Linear reference trajectory)

The controller output torques of the adaptive gain tuning controller is much bigger in the beginning of the trajectory because it wants to force the robot to reach the trajectory faster and smoother. This torques can be reduced by decreasing the parameter γ_y which results in a smaller trajectory reach time.

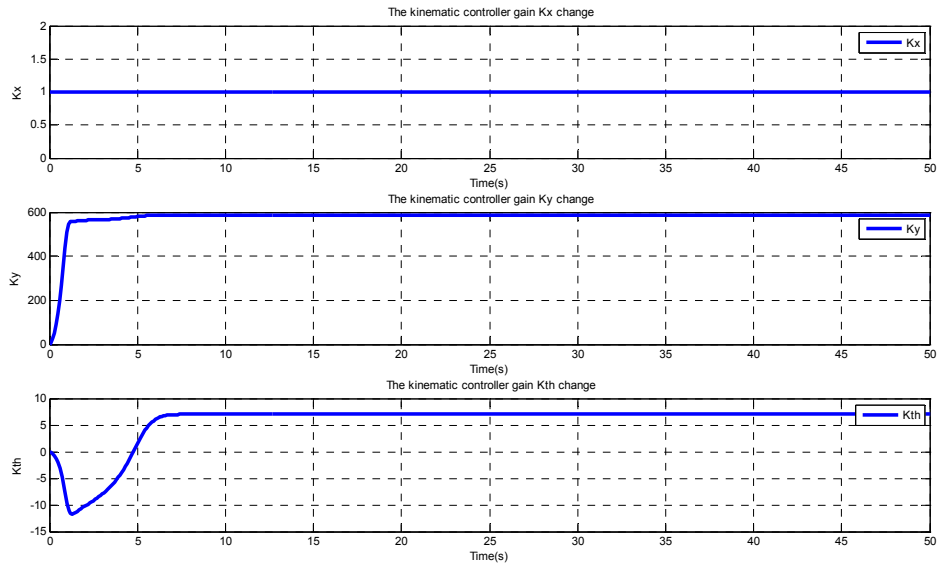

Figure 4-110: The adaptive gain tuning controller gains time response (Linear reference trajectory)

The controller gain K_x is made to be fixed because it will increase the controller output torque and make the robot unstable. The change in the other controller gains $K_y(t)$ and $K_\theta(t)$ will adapt the robot to each different reference trajectory and force the trajectory error to zero. As it can be seen from the above figure, the controller gains will become constant when the trajectory error becomes zero. This shows the convergence of the gradient decent learning algorithm which drives the cost function to zero.

Sinusoidal reference trajectory

Robot initial location: (0, 0, 0)

Backstepping controller gains: $K_x = 1, K_y = 55, K_{th} = 15$

Gains of the adaptive controller cost function:

$$J = \frac{1}{2}\sum \gamma_x {e_x}^2 + \gamma_y {e_y}^2 + \gamma_\theta {e_\theta}^2$$

$$\gamma_x = 1, \gamma_y = 50, \gamma_\theta = 1$$

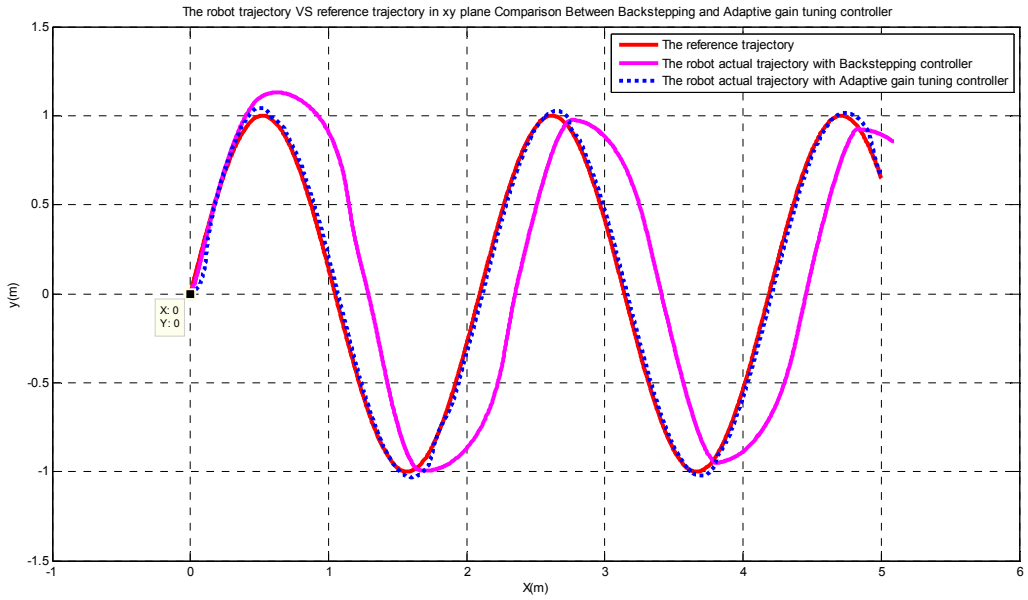

Figure 4-111: The robot trajectory in x-y plane, Comparison between the Backstepping controller and adaptive gain tuning controller (Sinusoidal reference trajectory)

The sinusoidal reference trajectory is considered a challenging reference trajectory for a nonholonomic mobile robot to track because of its peaks which involve a big change in

the heading angle. The perfect tracking performance of the adaptive gain tuning controller in comparison with the backstepping controller can be seen in the above figure.

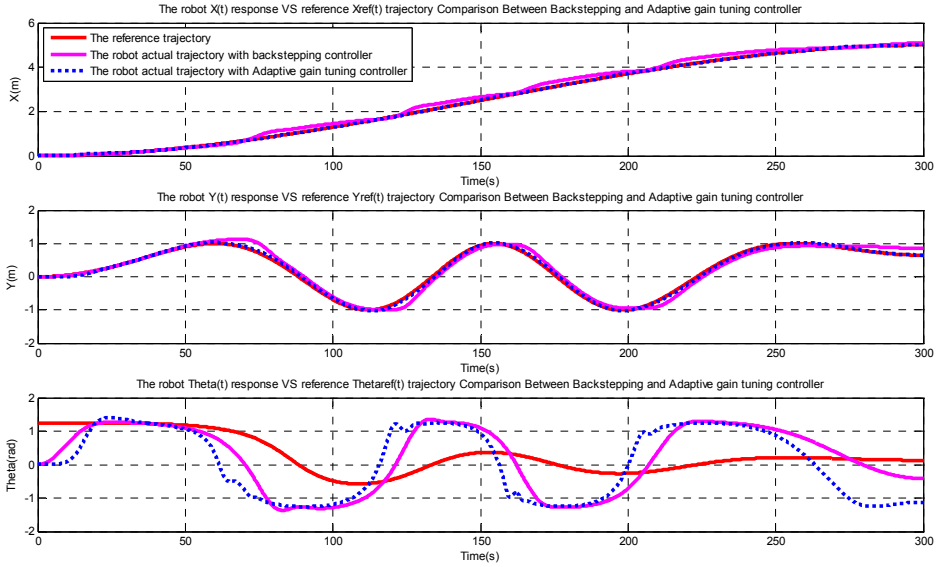

Figure 4-112: The robot output states time response vs. reference trajectories, Comparison between the Backstepping controller and adaptive gain tuning controller (Sinusoidal reference trajectory)

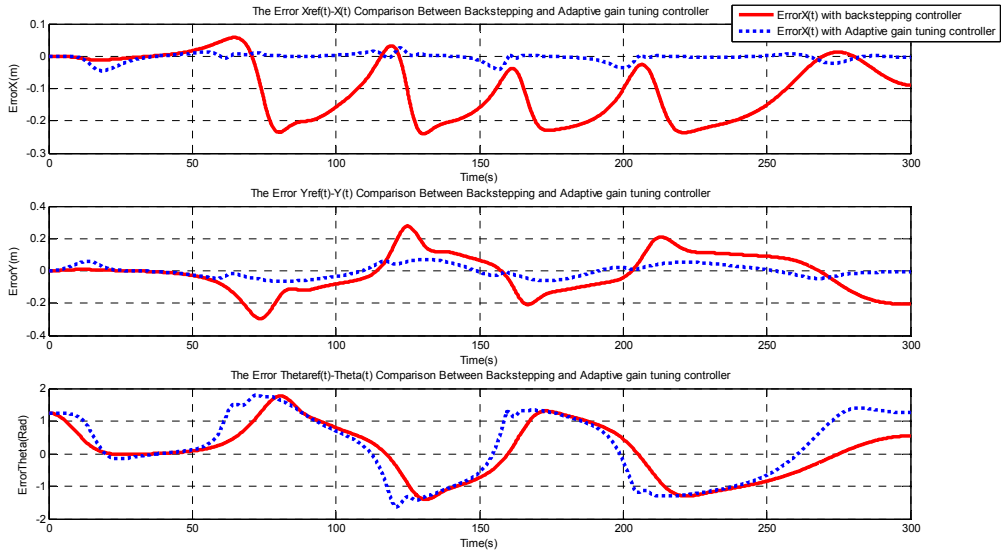

Figure 4-113: The robot output states errors Ex,Ey and Eth, Comparison between the Backstepping controller and adaptive gain tuning controller (Sinusoidal reference trajectory)

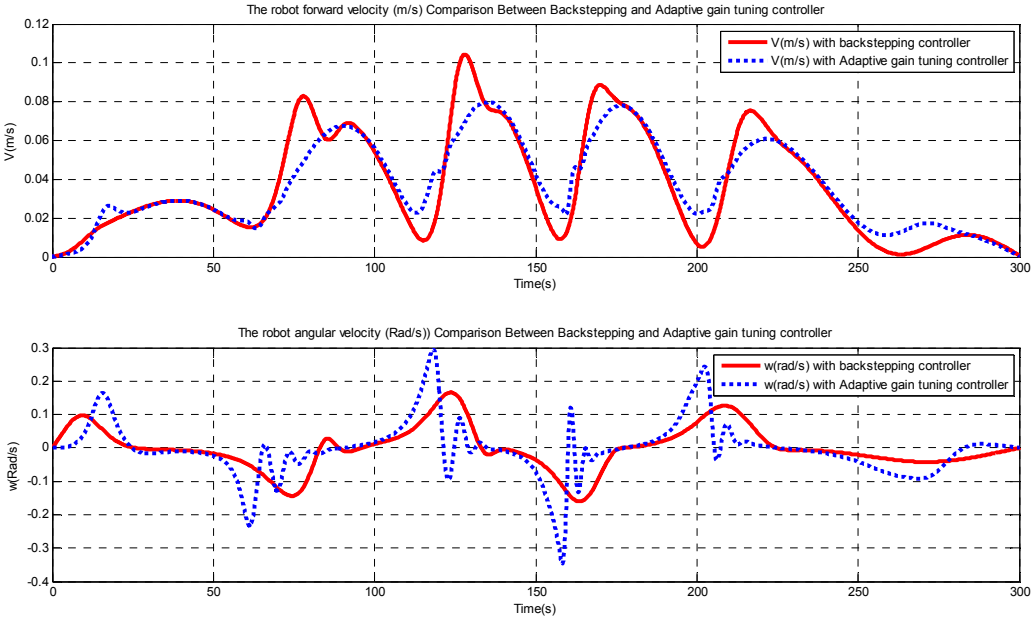

Figure 4-114: The robot linear and angular velocities, Comparison between the Backstepping controller and adaptive gain tuning controller (Sinusoidal reference trajectory)

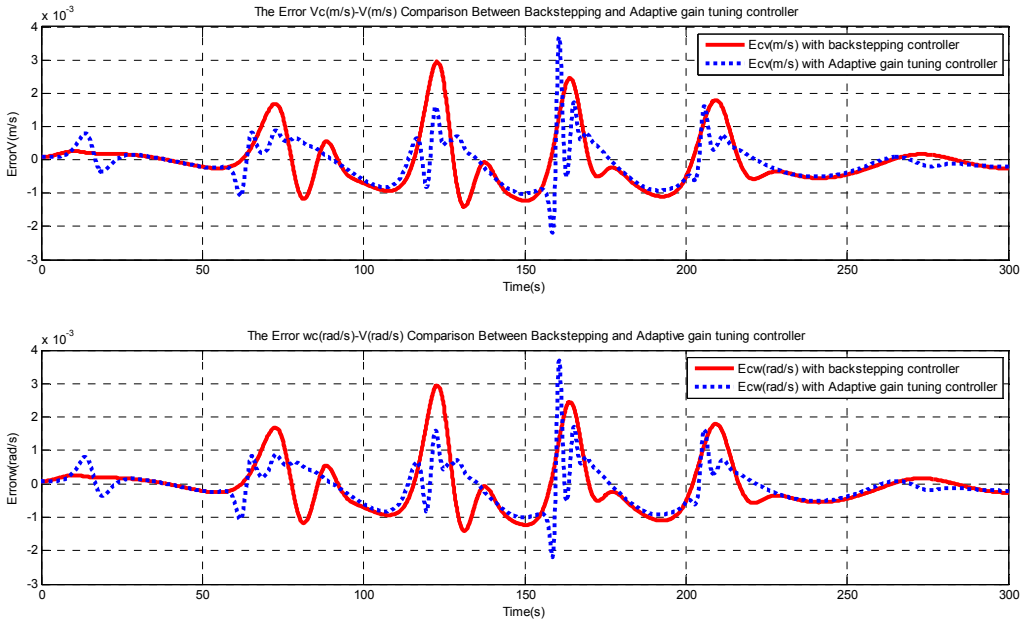

Figure 4-115: The robot velocity errors, Comparison between the Backstepping controller and adaptive gain tuning controller (Sinusoidal reference trajectory)

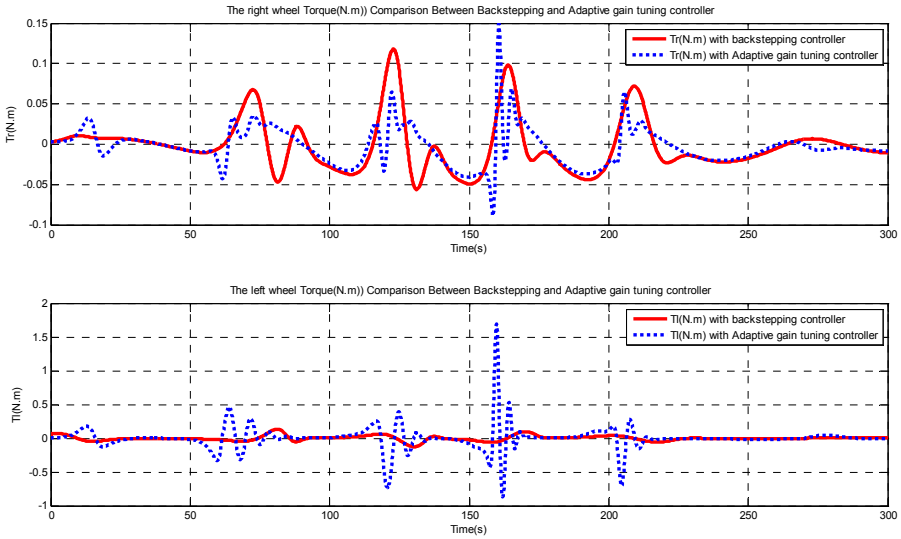

Figure 4-116: The rights and left wheel torques, Comparison between the Backstepping controller and adaptive gain tuning controller (Sinusoidal reference trajectory)

The controller output torques to the right and left robot wheels are within the actuator range and can be applies to the robot platform for the experimental verification of this control algorithm which will be explained in the experimental results section.

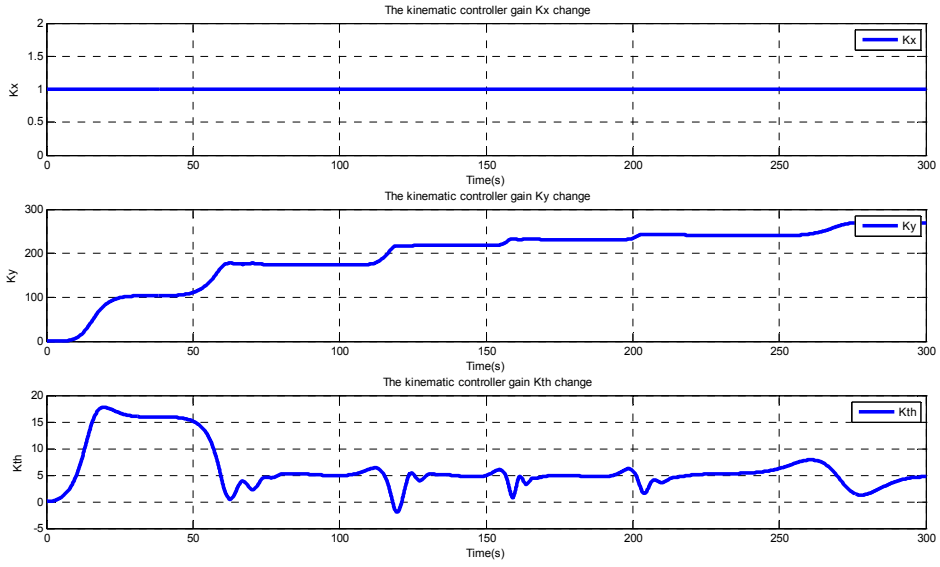

Figure 4-117: The adaptive gain tuning controller gains time response (Sinusoidal reference trajectory)

The controller gains are subject to change in the whole period of the trajectory because the reference trajectory has a lot of changes in the heading angle and it can be considered as a combination of different linear or parabolic trajectories.

Sinusoidal reference trajectory

Robot initial location: (-2, 1, 0)

Backstepping controller gains: $K_x = 1, K_y = 55, K_{th} = 15$

Gains of the adaptive controller cost function:

$$J = \frac{1}{2}\sum \gamma_x {e_x}^2 + \gamma_y {e_y}^2 + \gamma_\theta {e_\theta}^2$$

$$\gamma_x = 1, \gamma_y = 50, \gamma_\theta = 1$$

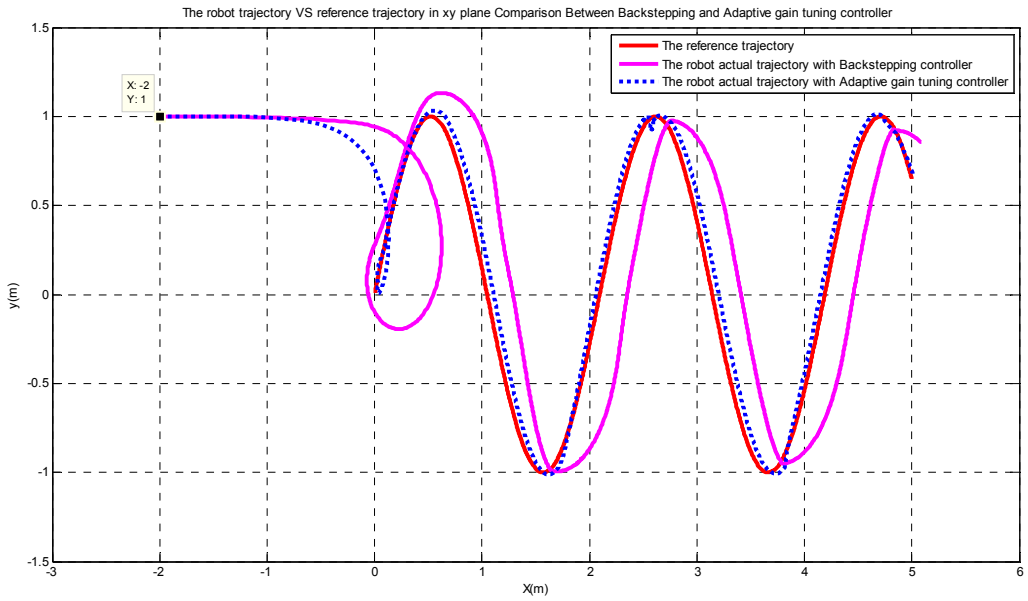

Figure 4-118: The robot trajectory in x-y plane, Comparison between the Backstepping controller and adaptive gain tuning controller (Sinusoidal reference trajectory)

The controllers tracking performance when the robot initial position is far from the reference trajectory is shown in the above figure. The adaptive performance of the proposed controller to each reference trajectory and each initial robot location can be seen and proved from the above figure.

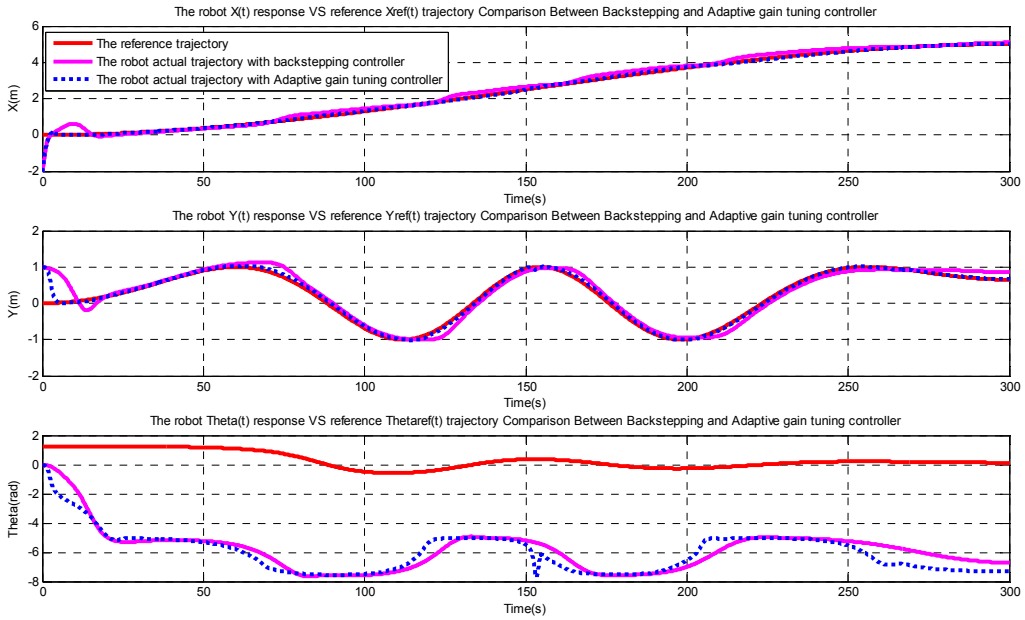

Figure 4-119: The robot output states time response vs. reference trajectories, Comparison between the Backstepping controller and adaptive gain tuning controller (Sinusoidal reference trajectory)

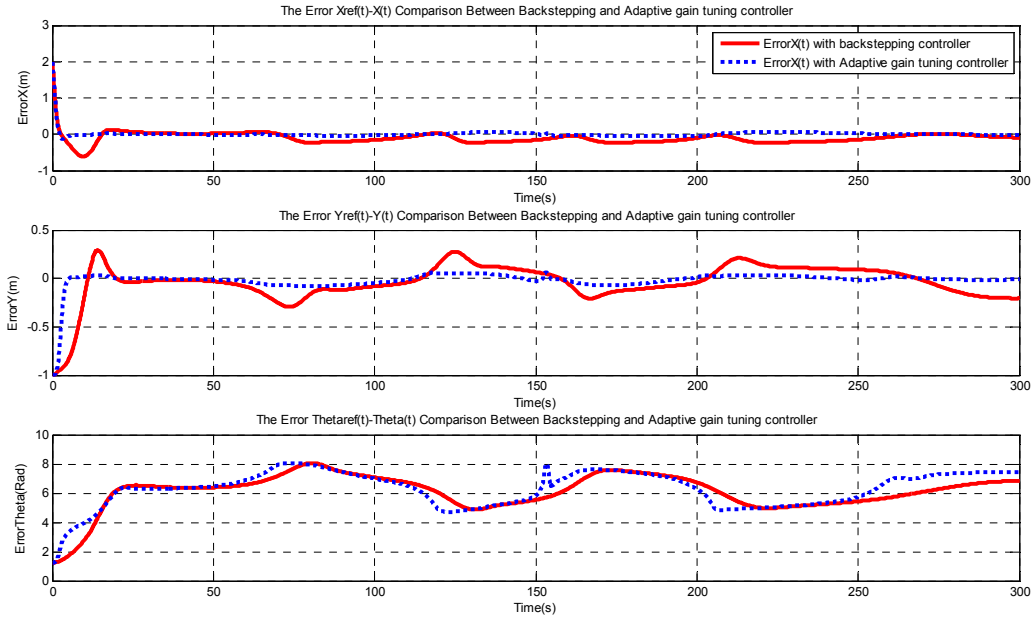

Figure 4-120: The robot output states errors Ex,Ey and Eth, Comparison between the Backstepping controller and adaptive gain tuning controller (Sinusoidal reference trajectory)

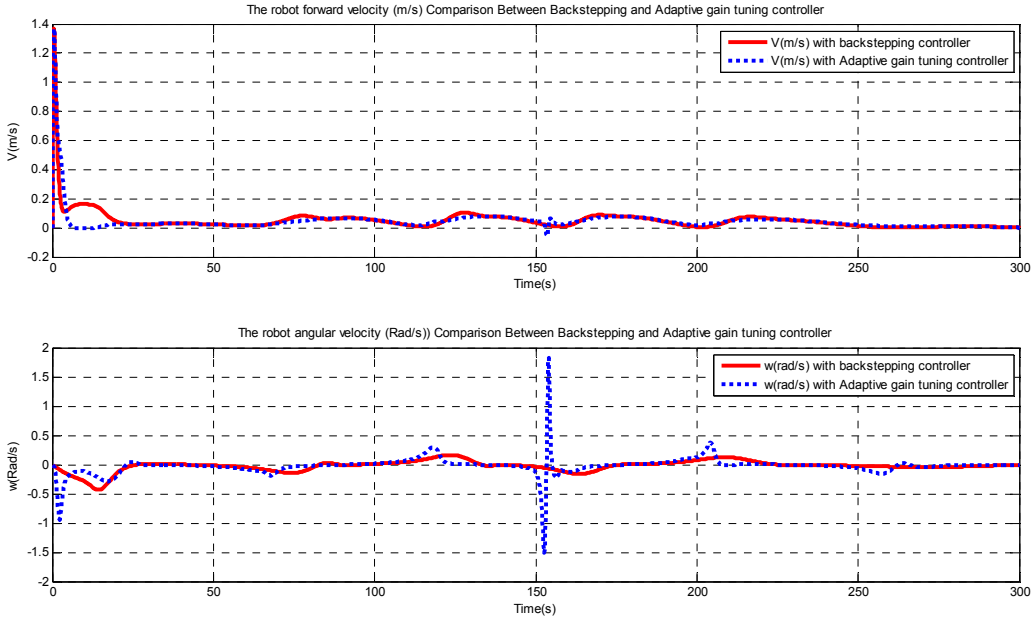

Figure 4-121: The robot linear and angular velocities, Comparison between the Backstepping controller and adaptive gain tuning controller (Sinusoidal reference trajectory)

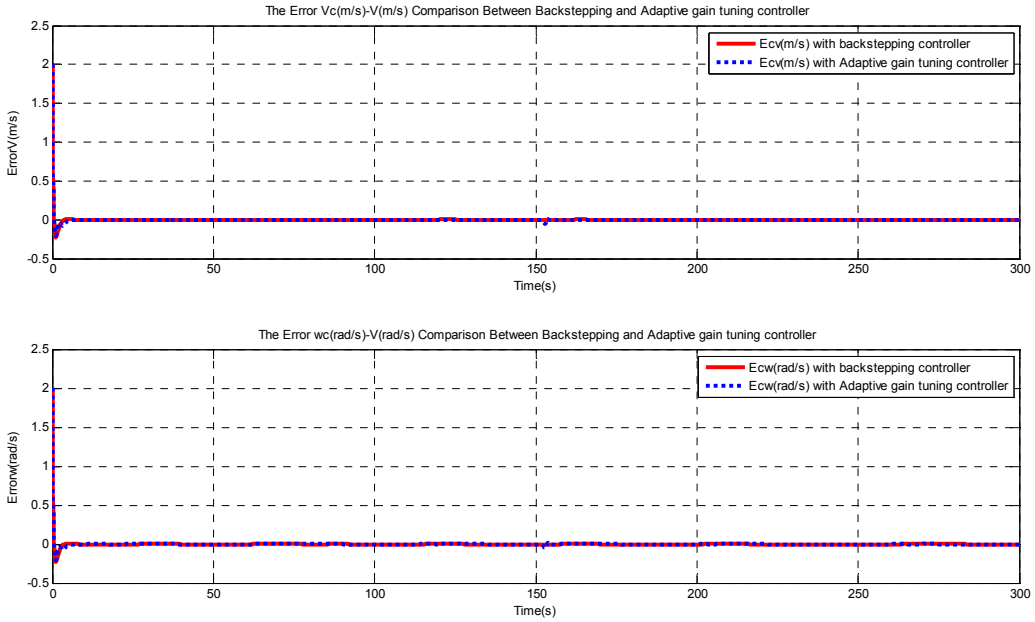

Figure 4-122: The robot velocity errors, Comparison between the Backstepping controller and adaptive gain tuning controller (Sinusoidal reference trajectory)

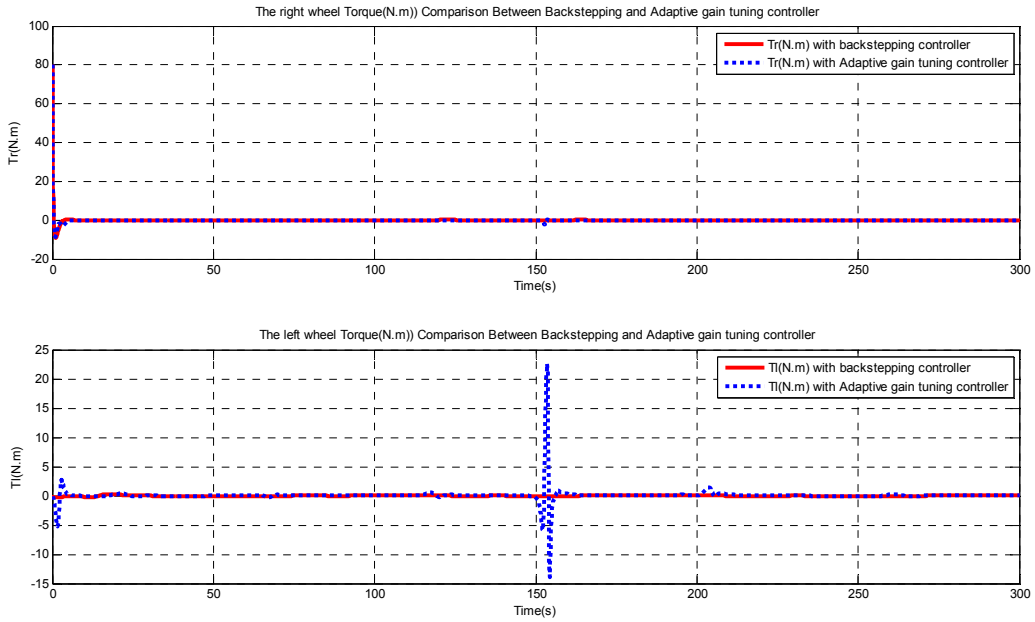

Figure 4-123: The rights and left wheel torques, Comparison between the Backstepping controller and adaptive gain tuning controller (Sinusoidal reference trajectory)

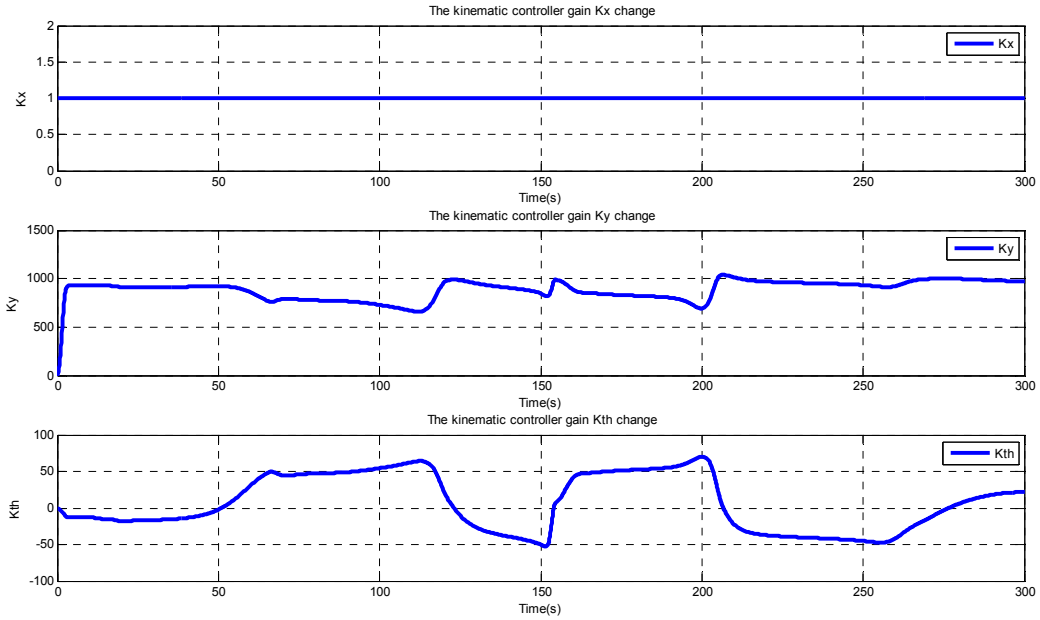

Figure 4-124: The adaptive gain tuning controller gains time response (Sinusoidal reference trajectory)

Rectangular reference trajectory

Robot initial location: (0, 1, 0)

Backstepping controller gains: $K_x = 1, K_y = 65, K_{th} = 15$

Gains of the adaptive controller cost function:

$$J = \frac{1}{2}\sum \gamma_x {e_x}^2 + \gamma_y {e_y}^2 + \gamma_\theta {e_\theta}^2$$

$$\gamma_x = 1, \gamma_y = 50, \gamma_\theta = 1$$

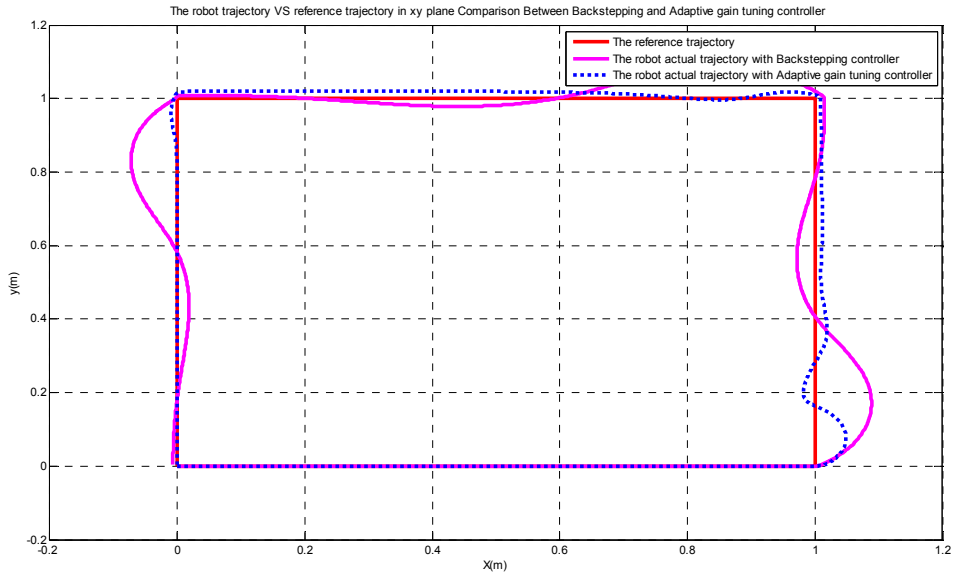

Figure 4-125: The robot trajectory in x-y plane, Comparison between the Backstepping controller and adaptive gain tuning controller (Rectangular reference trajectory)

The square reference trajectory is one of the famous trajectories which have been tested in literature for different trajectory tracking control algorithms. The perfect tracking performance of the adaptive gain tuning controller in comparison to the backstepping controller is shown in the above figure. Note that the backstepping controller with the fixed gains can be tuned for each of the reference trajectories but it a very time consuming process. Making the controller gains adaptive, solves this problem and makes the robot track any trajectory regardless of its initial position.

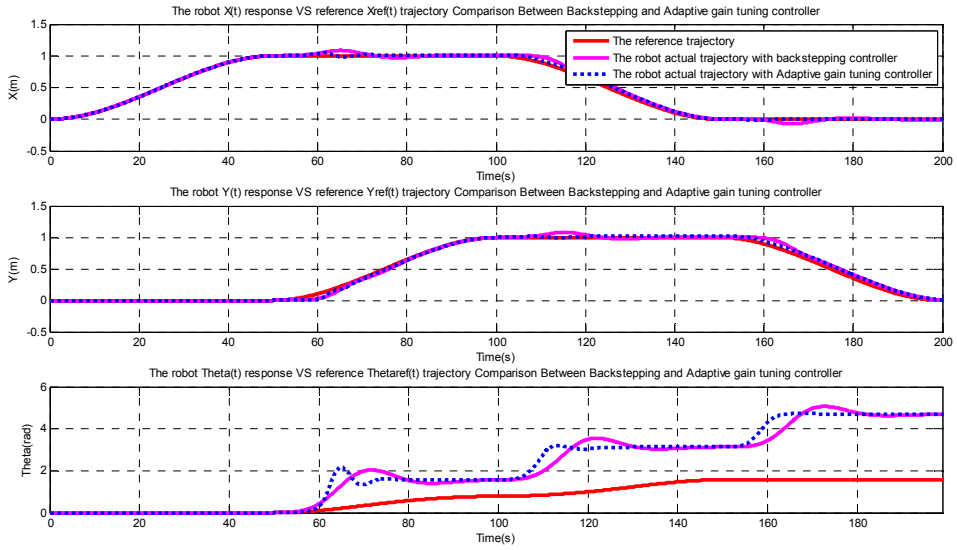

Figure 4-126: The robot output states time response vs. reference trajectories, Comparison between the Backstepping controller and adaptive gain tuning controller (Rectangular reference trajectory)

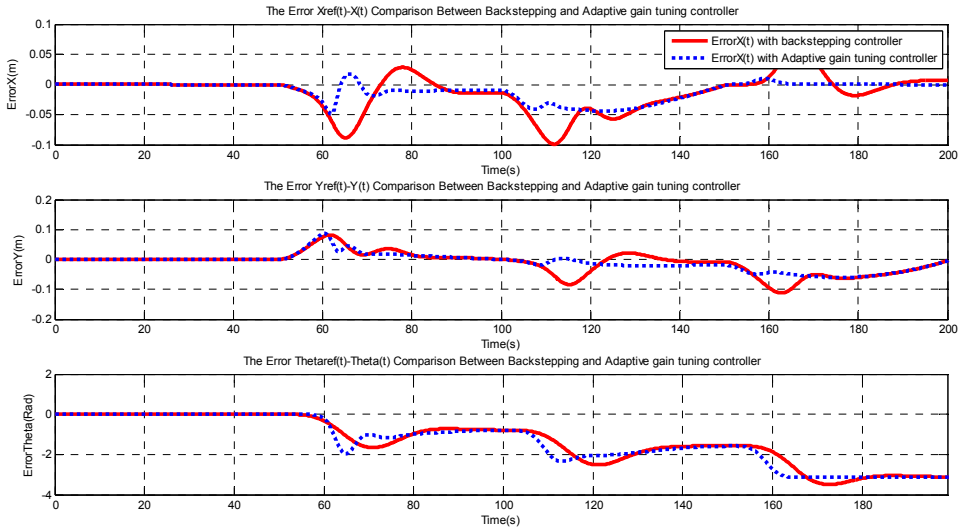

Figure 4-127: The robot output states errors Ex,Ey and Eth, Comparison between the Backstepping controller and adaptive gain tuning controller (Rectangular reference trajectory)

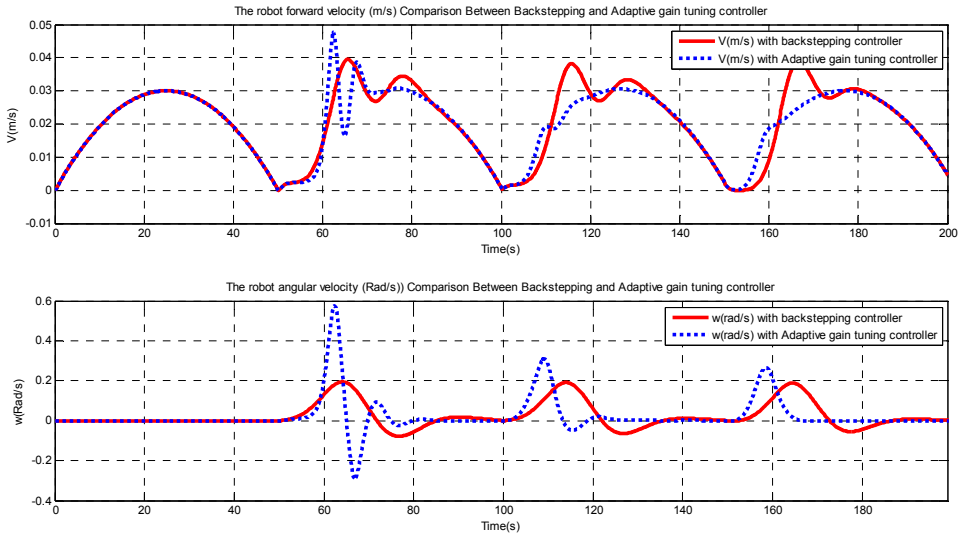

Figure 4-128: The robot linear and angular velocities, Comparison between the Backstepping controller and adaptive gain tuning controller (Rectangular reference trajectory)

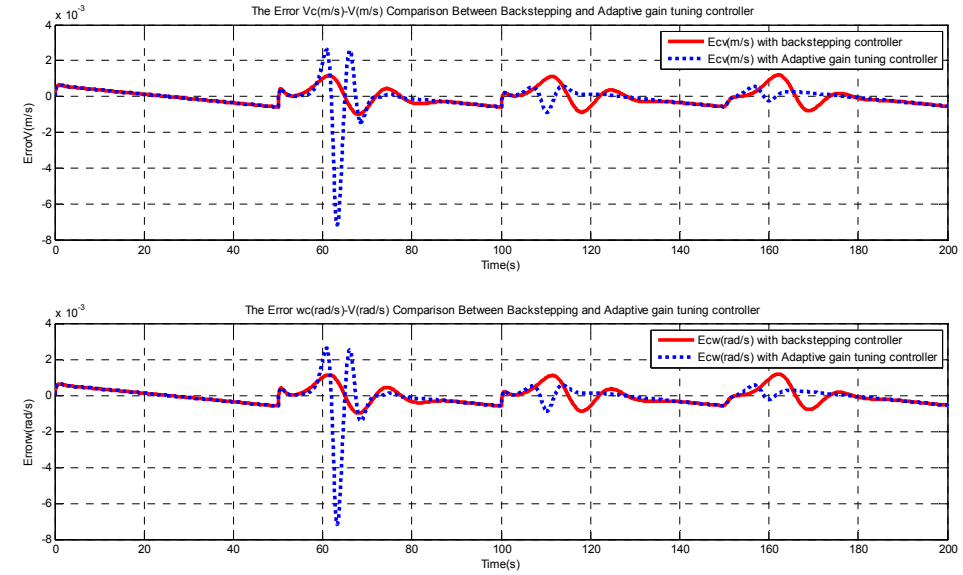

Figure 4-129: The robot velocity errors, Comparison between the Backstepping controller and adaptive gain tuning controller (Rectangular reference trajectory)

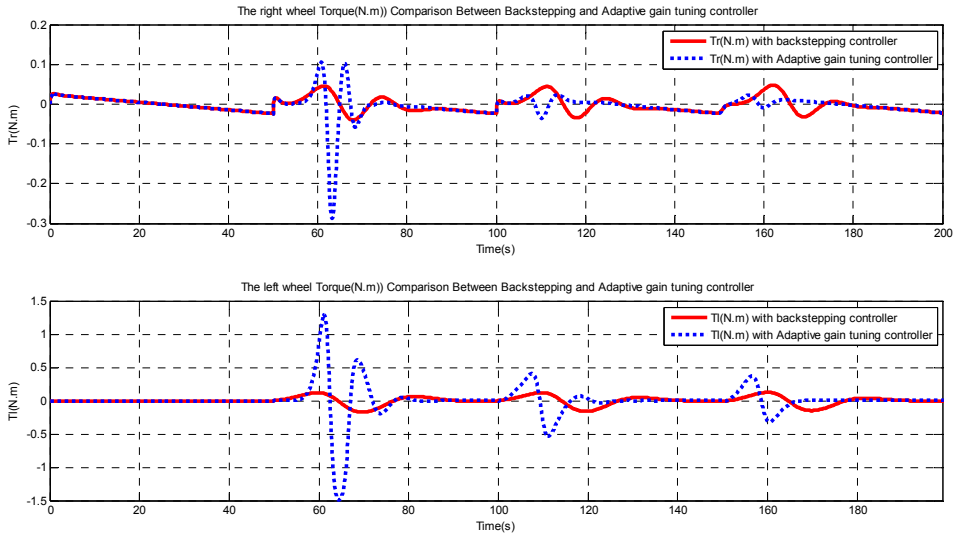

Figure 4-130: The rights and left wheel torques, Comparison between the Backstepping controller and adaptive gain tuning controller (Rectangular reference trajectory)

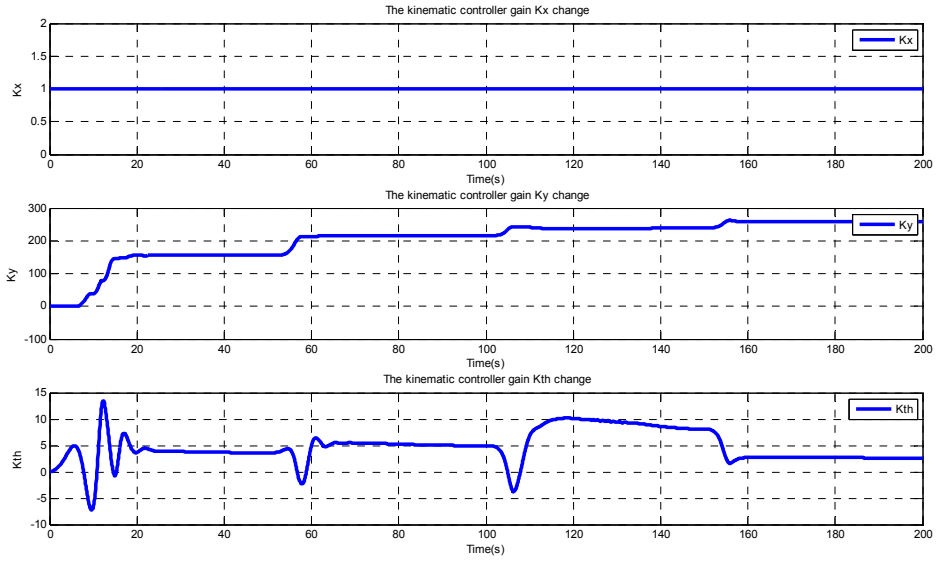

Figure 4-131: The adaptive gain tuning controller gains time response (Rectangular reference trajectory)

The edges of the each trajectory, which are the places that the robot heading angle has a big change are the most challenging parts of the trajectories. The change in the controller gains will happen mostly at these edge points as can be seen from the above figure.

Rectangular reference trajectory

Robot initial location: (-1, 0, 0)

Backstepping controller gains: $K_x = 1, K_y = 65, K_{th} = 5$

Gains of the adaptive controller cost function:

$$J = \frac{1}{2}\sum \gamma_x {e_x}^2 + \gamma_y {e_y}^2 + \gamma_\theta {e_\theta}^2$$

$$\gamma_x = 1, \gamma_y = 50, \gamma_\theta = 1$$

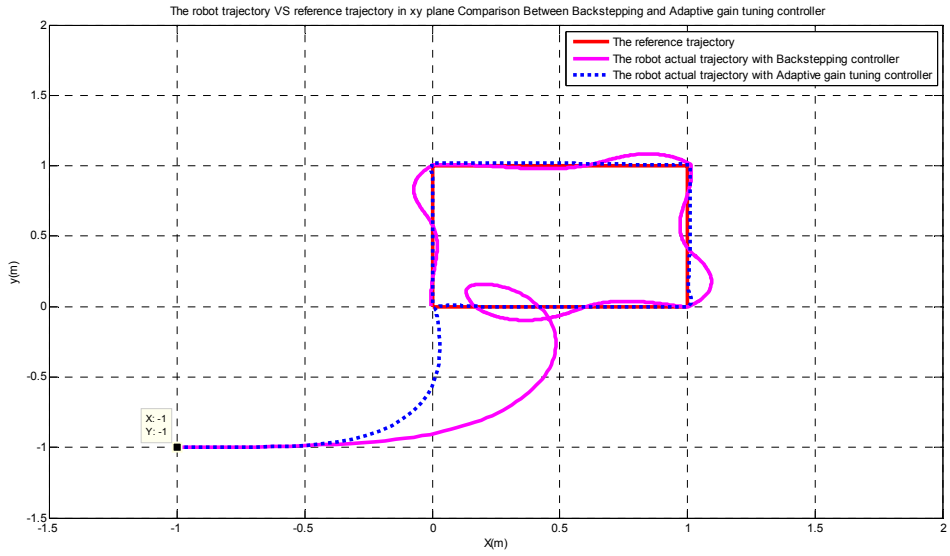

Figure 4-132: The robot trajectory in x-y plane, Comparison between the Backstepping controller and adaptive gain tuning controller (Rectangular reference trajectory)

The controller performance in existence of a big distance between the robot initial position and reference trajectory is shown in the above figure.

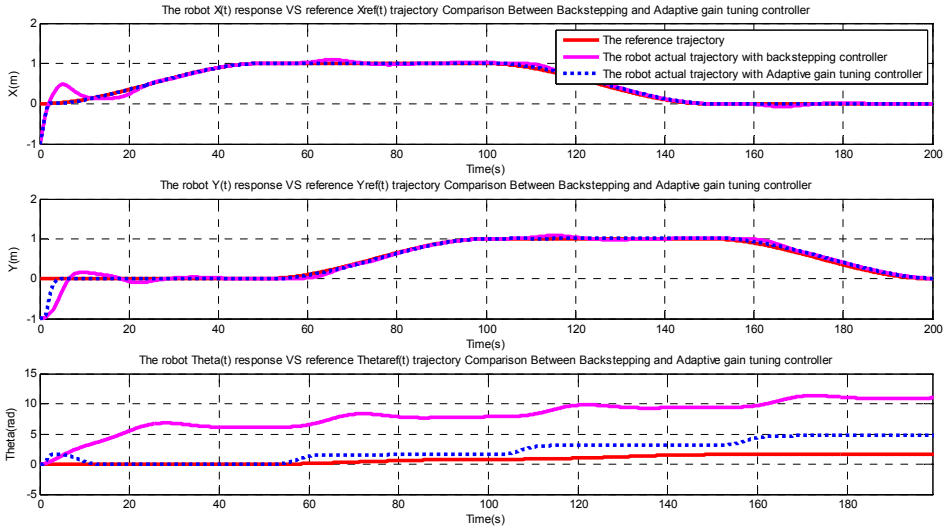

Figure 4-133: The robot output states time response vs. reference trajectories, Comparison between the Backstepping controller and adaptive gain tuning controller (Rectangular reference trajectory)

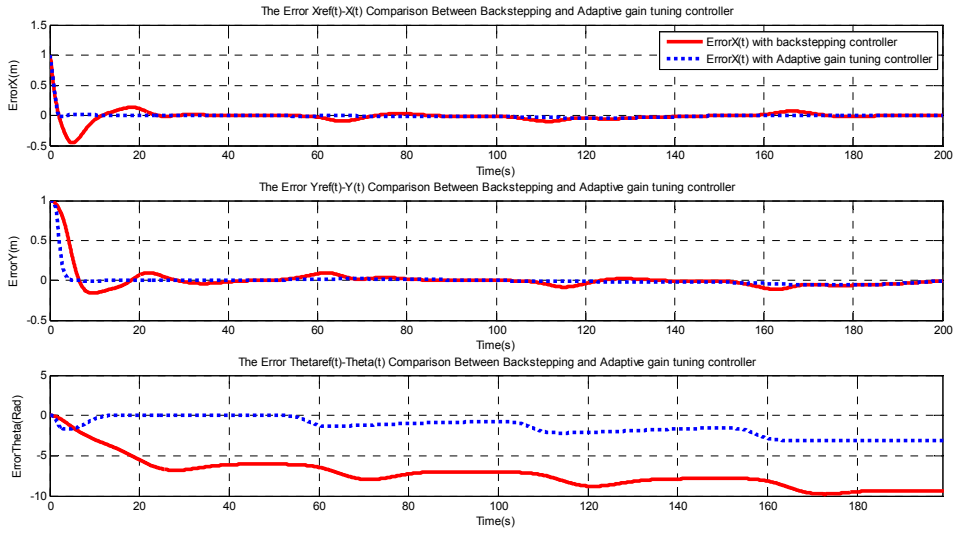

Figure 4-134: The robot output states errors Ex,Ey and Eth, Comparison between the Backstepping controller and adaptive gain tuning controller (Rectangular reference trajectory)

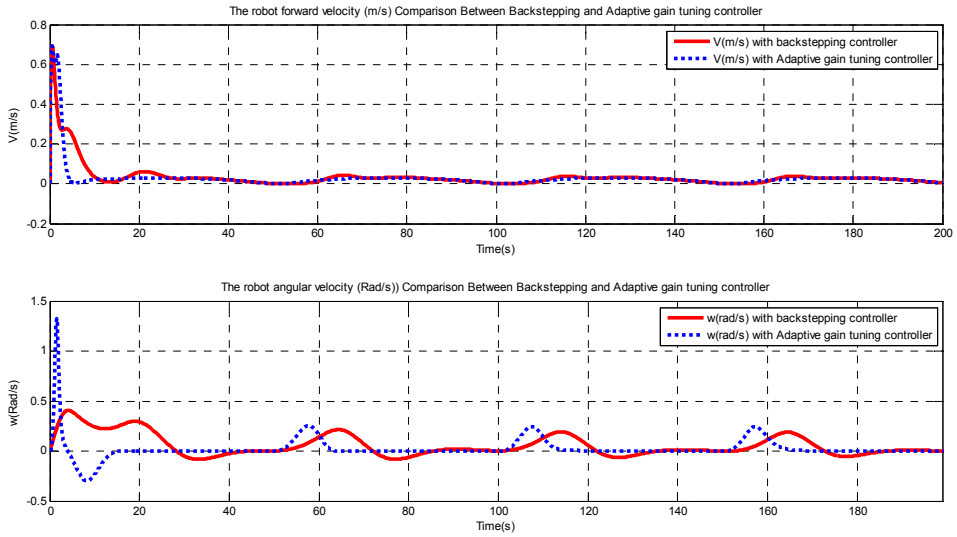

Figure 4-135: The robot linear and angular velocities, Comparison between the Backstepping controller and adaptive gain tuning controller (Rectangular reference trajectory)

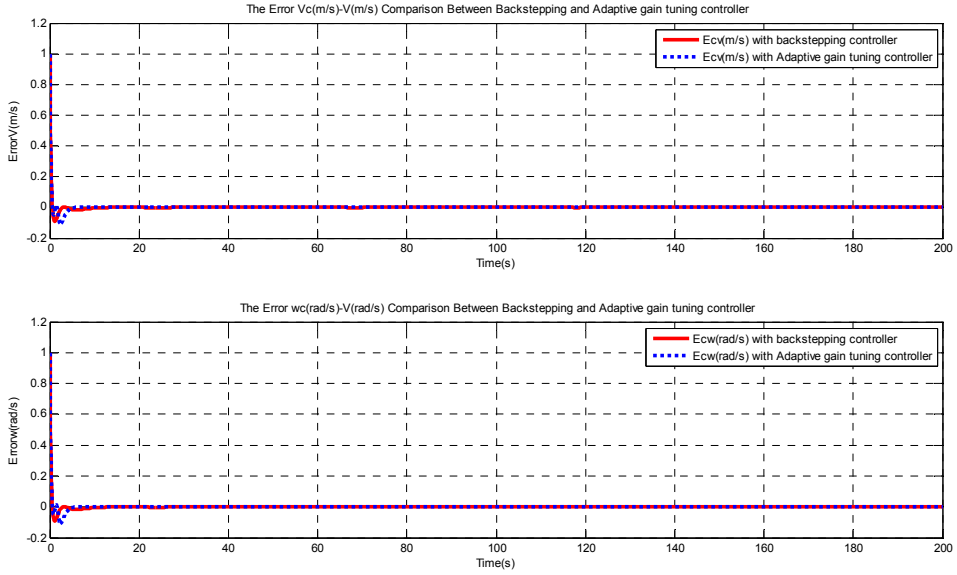

Figure 4-136: The robot velocity errors, Comparison between the Backstepping controller and adaptive gain tuning controller (Rectangular reference trajectory)

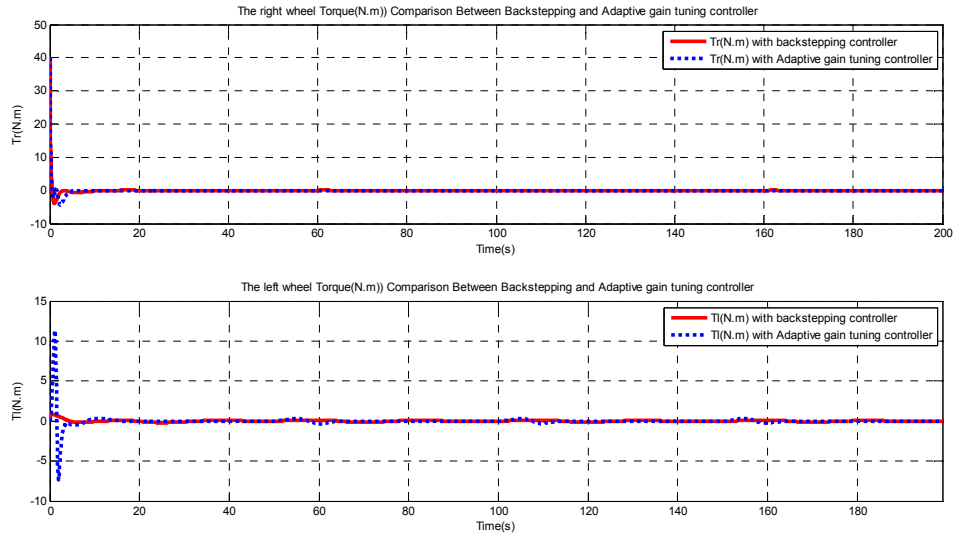

Figure 4-137: The rights and left wheel torques, Comparison between the Backstepping controller and adaptive gain tuning controller (Rectangular reference trajectory)

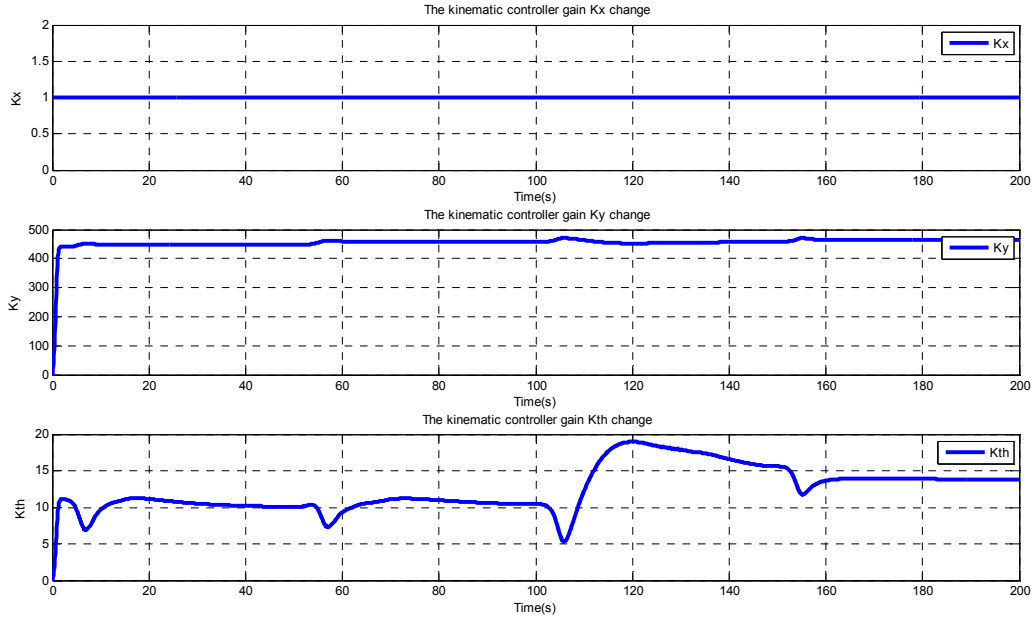

Figure 4-138: The adaptive gain tuning controller gains time response (Rectangular reference trajectory)

Circular reference trajectory

Robot initial location: (0, 0, 0)

Backstepping controller gains: $K_x = 1, K_y = 55, K_{th} = 15$

Gains of the adaptive controller cost function:

$$J = \frac{1}{2}\sum \gamma_x {e_x}^2 + \gamma_y {e_y}^2 + \gamma_\theta {e_\theta}^2$$

$$\gamma_x = 1, \gamma_y = 50, \gamma_\theta = 1$$

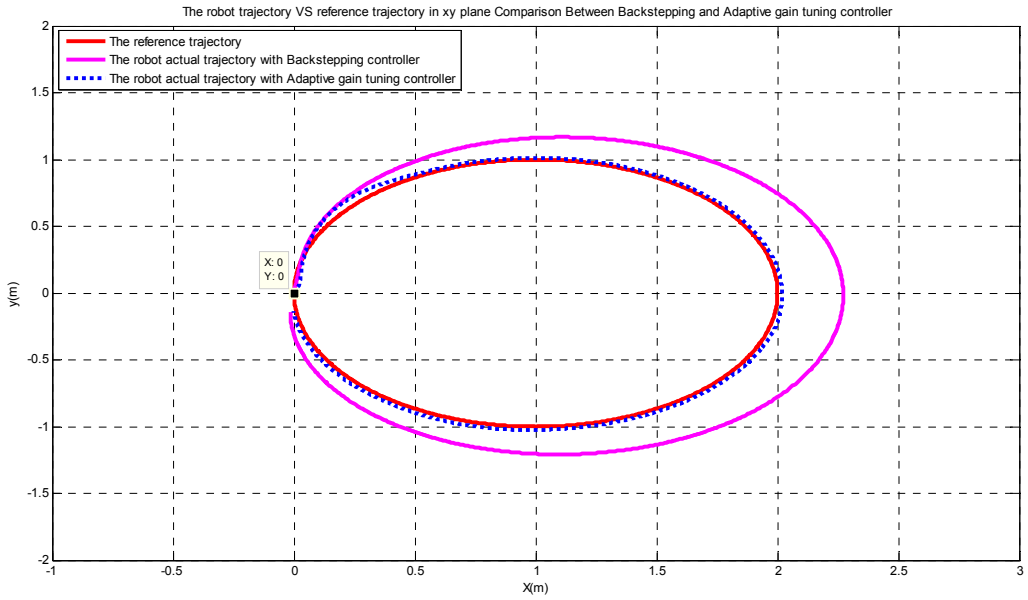

Figure 4-139: The robot trajectory in x-y plane, Comparison between the Backstepping controller and adaptive gain tuning controller (Circular reference trajectory)

The circular reference trajectory is another famous reference trajectories tested by different researched in this field. The great performance of the proposed adaptive gain tuning controller in comparison with the backstepping controller can be seen from the above figure.

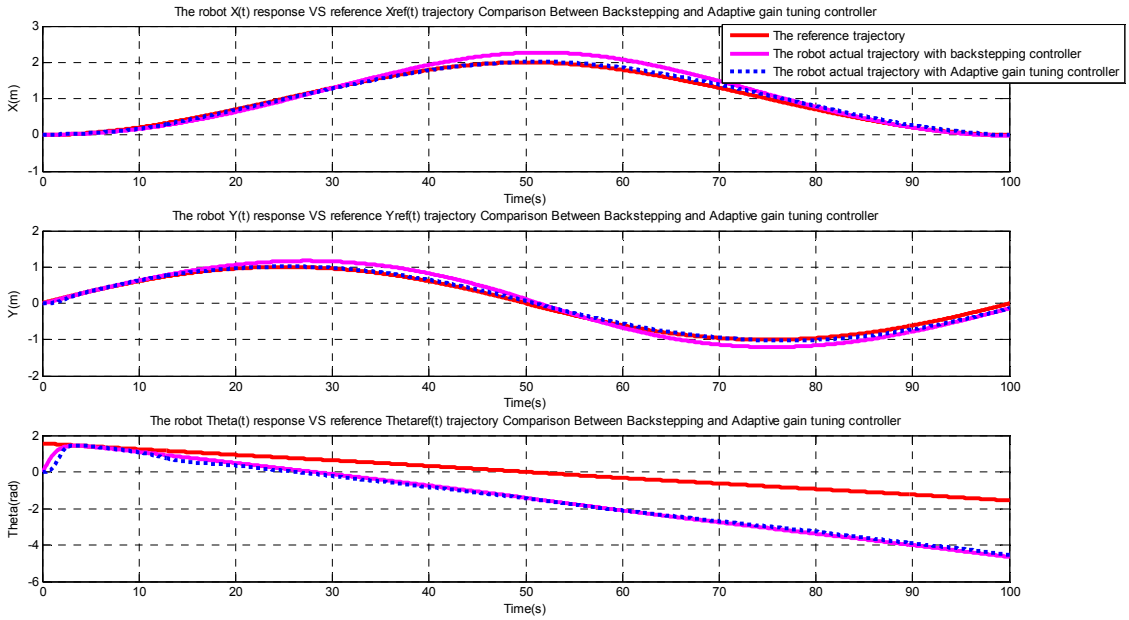

Figure 4-140: The robot output states time response vs. reference trajectories, Comparison between the Backstepping controller and adaptive gain tuning controller (Circular reference trajectory)

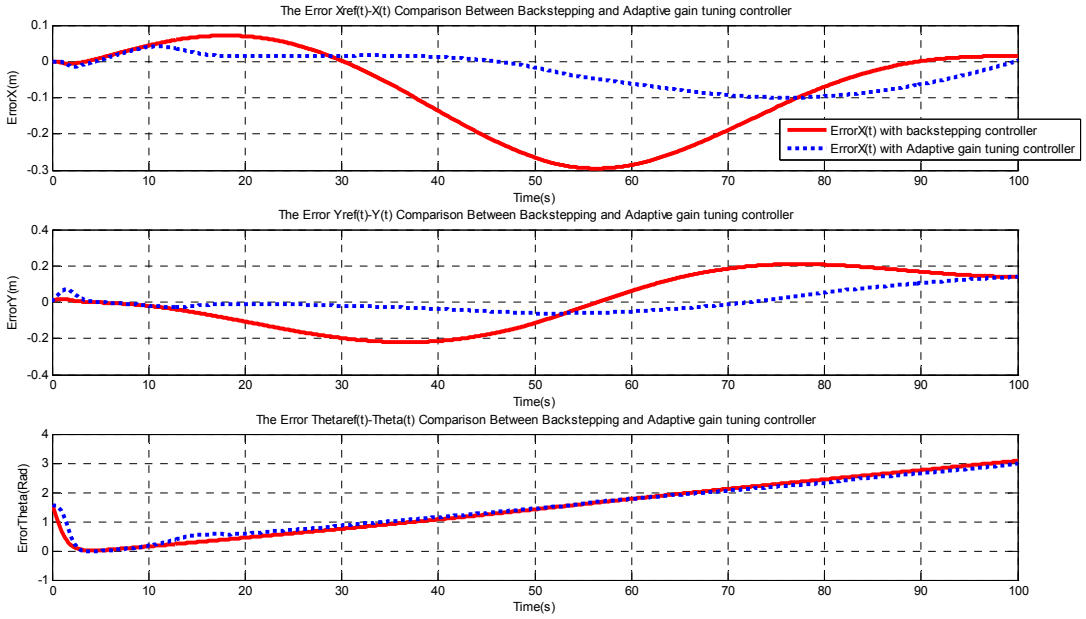

Figure 4-141: The robot output states errors Ex,Ey and Eth, Comparison between the Backstepping controller and adaptive gain tuning controller (Circular reference trajectory)

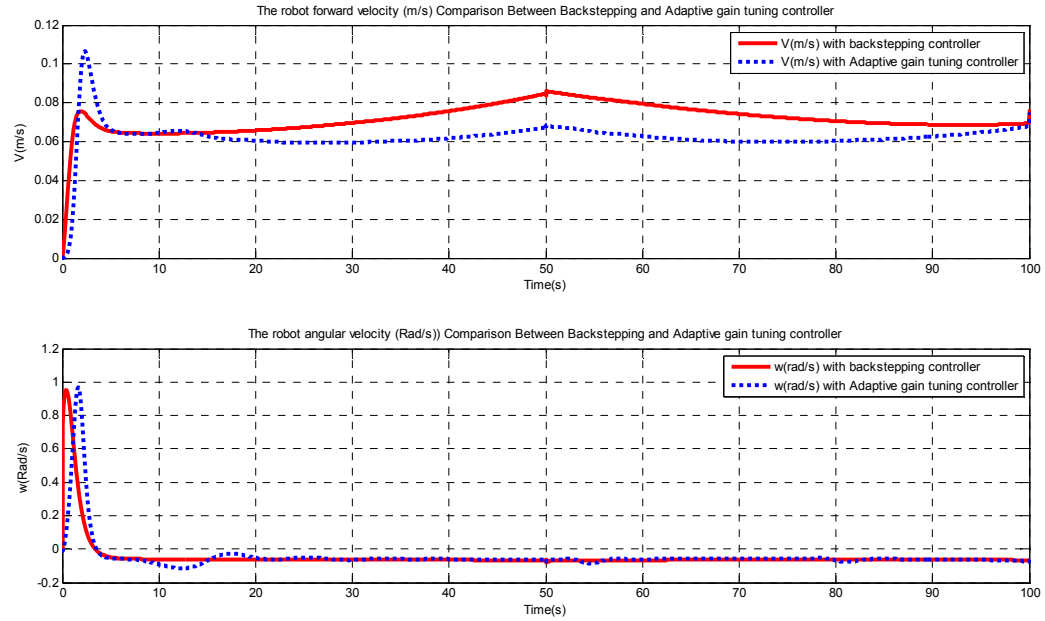

Figure 4-142: The robot linear and angular velocities, Comparison between the Backstepping controller and adaptive gain tuning controller (Circular reference trajectory)

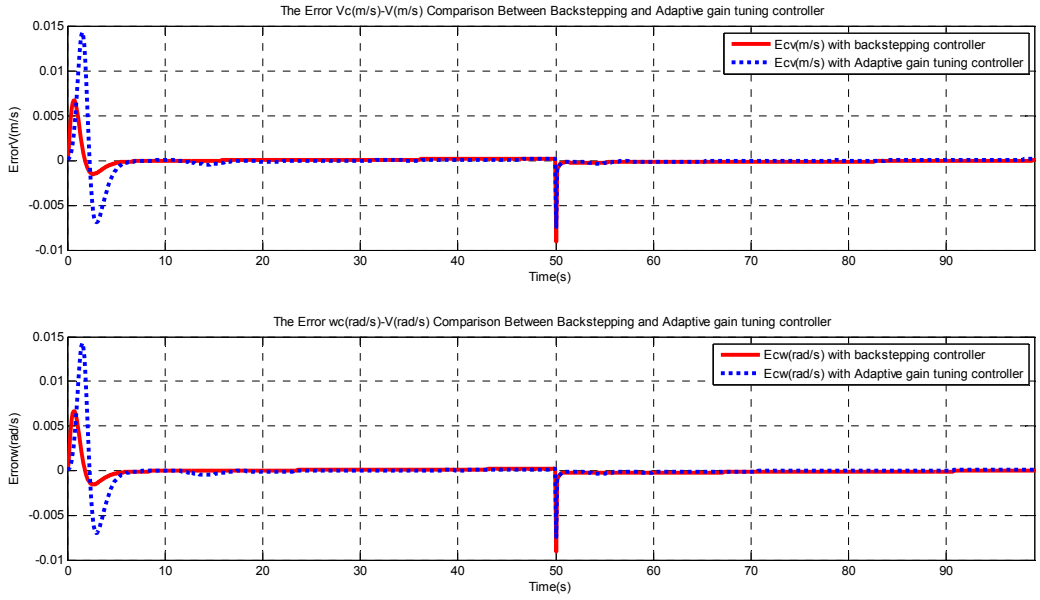

Figure 4-143: The robot velocity errors, Comparison between the Backstepping controller and adaptive gain tuning controller (Circular reference trajectory)

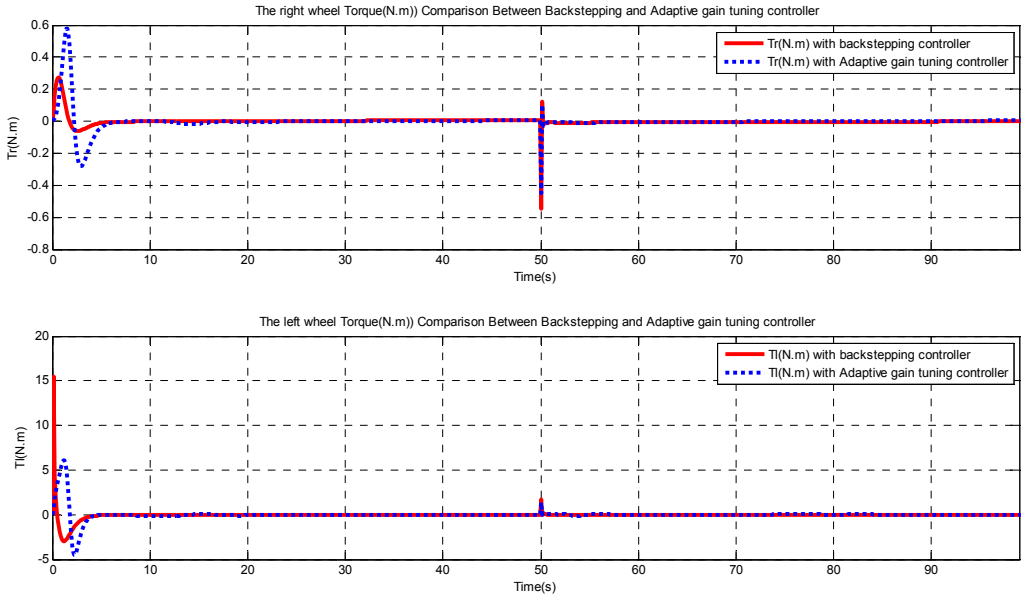

Figure 4-144: The rights and left wheel torques, Comparison between the Backstepping controller and adaptive gain tuning controller (Circular reference trajectory)

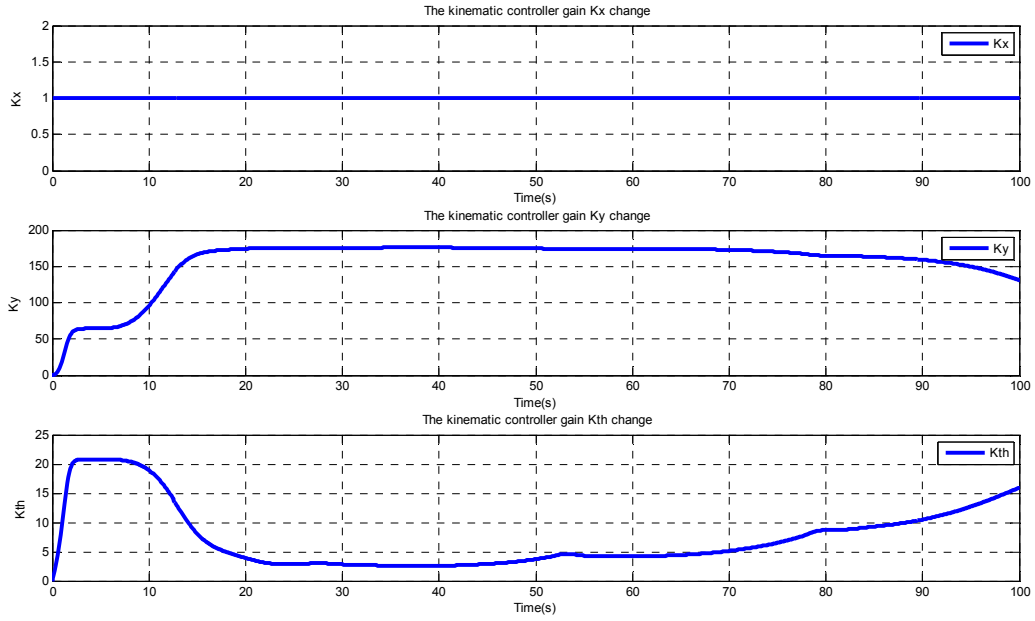

Figure 4-145: The adaptive gain tuning controller gains time response (Circular reference trajectory)

4.3.4 Conclusions

The above comprehensive comparison analysis between the NN-based adaptive backstepping controller and the backstepping controller shows that the NN-based adaptive backstepping controller is a perfect trajectory tracking controller according to the following advantages that it has over the backstepping controller:

- The zero tracking error will be achieved regardless of the shape of the reference trajectory and the robot initial position with respect to the reference trajectory.
- A respectively small trajectory reach time can be achieved when there is a big distance between the robot and the reference trajectory.
- A smooth robot trajectory will be produced using this controller which makes it easier to implement in real life applications.
- No knowledge of the system dynamics, robot initial position and the reference trajectory is needed before using this controller. The adaptive feature of this algorithm deals with the changes in the robot dynamics, reference trajectories and other uncertainties.
- The controller output torques are within the range of the robot actuators which makes this controller applicable to the experimental robot platform.

The experimental verification of the NN-based adaptive backstepping controller will be explained in detail in the next section.

CHAPTER 5

EXPERIMENTAL RESULTS AND VERIFICATIONS

The experimental mobile robot platform used to test the developed trajectory tracking controller will be introduced in this chapter in addition to the step by step procedure taken to run the robot and implement the control algorithm. The first section of this chapter talks about the general properties of the mobile robot and introduces its hardware and software layers and components. The section chapter is allocated to the details about the software structure used in the robot to implement any control algorithm and the third section will show the experimental trajectory tracking results of the robot with different reference trajectories and different control algorithms.

5.1 MOBILE ROBOT HARWARE STRUCTURE AND PROPERTIES

The mobile robot platform used in this research project is the Erratic (ERA) mobile robot [25] which is a full featured industrial mobile robot base and is shown in Figure 5-1:

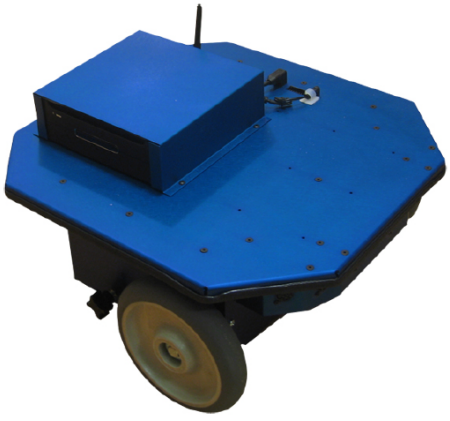

Figure 5-1: the ERA-MOBI platform

The general features of this mobile robot platform are as follows:

- Industrial-strength motors, aluminum alloy base, precision motion encoders
- Large, flat top plate for mounting sensors and peripherals
- Integrated controller and motor driver precise high-speed control of the drive system
- On-board low-power PC with integrated wireless networking
- High-level control software provided by open source Player/Stage system

The mobile robot base characteristics are shown in Figure 5-2:

Base platform size	40 cm (L) x 37 cm (W) x 18 cm (H)
Wheels	15 cm diameter (driven) 6.25 cm diameter (caster) Polymer core, soft non-marking rubber tread
Wheelbase	33 cm
Drive type	Differential, single rear caster
Maximum speed	2.0 m/sec, 720 deg/sec
Motors	DC reversible with gearhead 72 W continuous power
Encoder resolution	500 cycles per motor revolution
Controller	16 bit microcontroller Integrated controller / motor driver Analog, digital, and servo interfaces
Power	12V, 7AH lead-acid batteries (x3) 5A charger
Weight	4.5 Kg (base) 12 Kg (base + 3 batteries)
Payload	20 Kg

Figure 5-2: the ERA-MOBI base properties

The basic components of this mobile robot platform are shown Figure 5-3:

Figure 5-3: the basic components of the ERA-MOBI platform

The robot base, 12V batteries with attached plugs, the battery charger and the on-board PC are shown in the above figure. The computer is installed onto the robot using a 3M interlock tape system. The robot configuration with the embedded PC is shown in Figure 5-4:

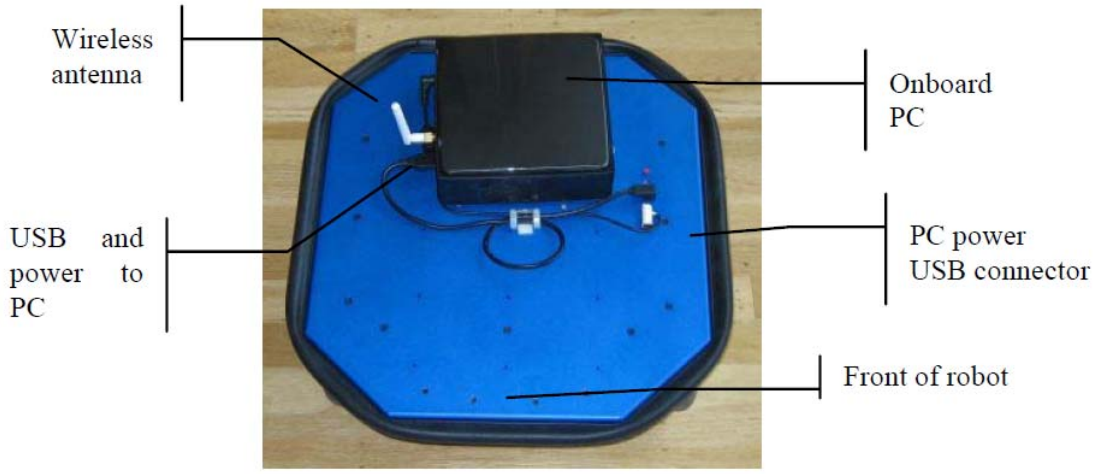

Figure 5-4: top of the robot, populated with a computer and cable connections. Front of the robot is at the bottom of the above image

There is one LED indicator on top of the robot which indicates the status of the robot. When the robot is powered, it is lit. The color indicated the status of the robot controller:

- Green when the controller is on, but there is no power in the robot wheels.
- Orange when the controller is on and there is power in the robot wheels.

- Flashing when the battery is low (<11V), otherwise, solid.

There is an emergency button beside the LED indicator which is responsible for stopping the motors when the robot is moving. The LED indicator and the emergency button are shown in Figure 5-5:

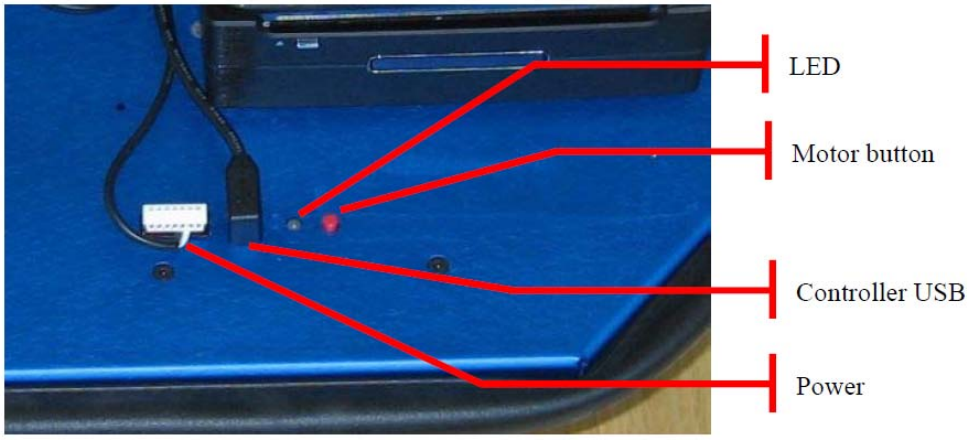

Figure 5-5: Ports, LED and power button on top of the robot

Accessing the on-board PC can be done by one of the following methods:

- Keyboard, mouse and monitor
- Wired Ethernet connection
- Wireless interface

The first method of accessing the robot PC is used in this project because of its simplicity. After connecting to the robot PC, one should reach the interfaces and find out how to use them. The robot controller is connected to several sensors and actuators on the robot. Through the USB connection, the controller accepts commands to control the actuators and send the information it collects from the sensors. The user programs on the robot PC can communicate directly with the controller using the predefined packet protocols which will be described in the software structure section of this chapter. The control parameters and velocity and acceleration limits will be described in the next section.

5.1.1 Mobile robot embedded controller parameters

Many of the functions of the controller are modified by a set of configuration parameters. These parameters are stored in the flash memory on the controller and most can be changes through controller commands. The Erratic controller drives the robot wheels using an integrated dual H-Bridge motor driver. The driver is able to supply up to 30A at 12V to the motors, although in practice the current is limited to less than this. The motors are rated at 75W continuous power and are able to drive the robot at speeds up to 2 m/sec

and 720 deg/sec. Each wheel is controlled independently, allowing the robot to turn as well as drive forwards and backwards (differential drive). The controller is also connected to the motor encoders, which counts 500 cycles per revolution of the motor axis. With this resolution the controller can implement a very stable feedback motion control , even at very low speeds. In addition, the controller integrates the motor encoder values into 2D pose estimates (X, Y and angle coordinates). This pose estimate is called the *odometry pose* or sometimes the *dead-reckoned pose*.

The controller implements a constant acceleration velocity profile when given a velocity set point. That is, when the controller is commanded to achieve a velocity, it uses a constant acceleration to change the velocity in a linear manner, until the set point is achieved.

The motors are driven by a pulse width modulation PWM signal, which efficiently throttles full battery current to produce slower speeds. A higher PWM frequency exhibits less noise, but too high can cause the overheating of the drivers. 16 KHZ is a good compromise – only the dogs and small children can hear this motor whine.

The PWM maximum duty cycle cuts the maximum current that can be delivered to the motors. For higher performance and faster speeds, set this number higher. The controller uses a 200 Hz control loop to achieve velocity set points, using feedback from the encoders. The PID parameters control the responsiveness and stability of this loop. The PWM and PID controller parameters are listed in Figure 5-6:

the robot controller parameter	parameter value	effect
Motor PWM frequency Hz	16 Hz	pulse frequency for motor current
Motor PWM max on fraction	0.7	the maximum duty cycle for PWM
PID translation P	40	Proportional value for the translation PID
PID translation D	80	Differential value for the translation PID
PID translation I	0	Integral value for the translation PID
PID rotation P	30	Proportional value for the translation PID
PID rotation D	60	Differential value for the translation PID
PID rotation I	0	Integral value for the translation PID

Figure 5-6: the PWM and PID controller parameters

The Erratic controller has a complement of analog and digital sensors. Some of these are attached to fixed functions, such as measuring the battery voltage or the motor current. The analog and digital information may be accessed via the low-level packet interface, or through the Player driver and associated proxies.

The battery voltage is measured by a fixed analog input on the controller. Its value is returned with every motor information packet, in tenths of a volt. The nominal voltage is 12V. When the voltage goes below 11V, the LED starts blinking, and the batteries should be recharged. Analog and digital inputs are available through the low-level packet interface. A controller command turns the sending of the packets on or off. The packet contains information about 4 analog channels and 3 digital channels. The analog channels return a count of [0, 1023], which translates to a voltage from [0V, 3.5V]. The digital channels return a value of [0, 1], which is mapped into a voltage of 0 or 1.

After describing all of the robot parameters, we can move to the software structure section and explain the method we used to implement the trajectory tracking controller on the robot platform.

5.2 MOBILE ROBOT SOFTWARE STRUCTURE AND PROPERTIES

The software we used to communicate with the robot hardware and low level controller is called Player software. Player is a hardware abstraction layer. That means that it talks to the bits of hardware on the robot (like a claw or a camera) and lets you control them with your code, meaning you don't need to worry about how the various parts of the robot work. A complete software structure then is composed of the following two parts:

- The written code by the robot user: this part talks to Player software.
- Player. This takes your code and sends instructions to a robot. From the robot it gets sensor data and sends it to your code.

In using the Player software, there are two important file types that one should be familiar with:

- A .cfg file
- A .c++ file

The .c++ file is the file in which the user writes the control algorithm and communicates with the player software. The .cfg file is what Player reads to get all the information about the robot that you are going to use. This file tells Player which drivers it needs to use in order to interact with the robot, if you're using a real robot these drivers are built in to player The .cfg file tells Player how to talk to the driver, and how to interpret any data

from the driver so that it can be presented to your code. The .cfg file does all this specification using interfaces and drivers, which will be discussed as follows:

- Drivers are pieces of code that talk directly to hardware. These are built in to Player so it is not important to know how to write these as you begin to learn Player/Stage. The drivers are specific to a piece of hardware so, say, a laser driver will be different to a camera driver, and also different to a driver for a different brand of laser. This is the same as the way that drivers for graphics cards differ for each make and model of card. Drivers produce and read information which conforms to an "interface".
- Interfaces are a set way for a driver to send and receive information from Player. Like drivers, interfaces are also built in to Player and there is a big list of them in the Player manual[2]. They specify the syntax and semantics of how drivers and Player interact.
- A device is a driver that is bound to an interface so that Player can talk to it directly. This means that if you are working on a real robot that you can interact with a real device (laser, gripper, camera etc) on the real robot, in a simulated robot you can interact with their simulations.

To learn how to write code for Player or Player/Stage it helps to understand the basic structure of how Player works. Player uses a Server/Client structure in order to pass data and instructions between your code and the robot's hardware. Player is a server, and a hardware device[22] on the robot is subscribed as a client to the server via a thing called a proxy. The .cfg _le associated with your robot (or your simulation) takes care of telling the Player server which devices are attached to it, so when we run the command player some_cfg.cfg this starts up the Player server and connects all the necessary hardware devices to the server. Figure 5-7 shows a basic block diagram of the structure of Player when implemented on a robot:

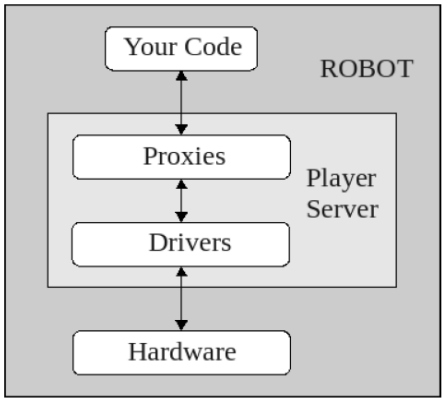

Figure 5-7: The server/client control structure of Player when used on a robot. There may be several proxies connected to the server at any time.

Player has functions and classes which will do all this for the user, but you need to actually call these functions with your code and know how to use them. This is the main responsibility of the c++ code written by the robot user.

The most important part of the code is interacting with the proxies. As you may expect, each proxy is specialised towards controlling the device it connects to. This means that each proxy will have different commands depending on what it controls. In Player version 2.1.0 there are 38 different proxies which you can choose to use, many of which are not applicable to Player/Stage. The most important and useful proxy used for the trajectory tracking purpose of the mobile robot is called the positon2d proxy. It controls the robot's motors and keeps track of the robot's odometry (where the robot thinks it is based on how far its wheels have moved). Some of the most important commands and function used in trajectory tracking codes which are related to this proxy are explained as follows:

Get/Set Speed: The Set Speed command is used to tell the robot's motors how fast to turn. The command used to do this task is as follows:

SetSpeed (double XSpeed, double YawSpeed)

The above command reads the wheels motors angular velocity and converts them to the linear and angular velocity of the robot. The value of the linear velocity is in meters/sec and the value of the angular velocity is in rad/sec.

GetPos: This function interacts with the robot's odometry. It allows you to monitor where the robot thinks it is. Coordinate values are given relative to its starting point, and yaws are relative to its starting yaw. The commands used to do the above task are as follows:

GetXPos (): gives current x coordinate relative to its x starting position.

GetYPos (): gives current y coordinate relative to its y starting position.

GetYaw (): gives current yaw relative to its starting yaw.

SetMotorEnable (): This function takes a Boolean input, telling Player whether to enable the motors or not. If the motors are disabled then the robot will not move no matter what commands are given to it, if the motors are enabled then the motors will always work, this is not so desirable if the robot is on a desk or something and is likely to get damaged. Hence the motors being enabled are optional. If you are using Player/Stage, then the motors will always be enabled and this command doesn't need to be run.

Other proxies and functions are used to perform the other tasks needed for the robot trajectory tracking such as reading the current of the motors, or setting and resetting the robot odometry.

The c++ codes and the generated library files and the .cfg files used to perform and implement the trajectory tracking controller are included in the appendix. The results of the robot trajectory tracking and verification of different control algorithms on the robot platform are included in the next section of this chapter.

5.3 EXPERIMENTAL RESULTS OF THE NN-BASED ADAPTIVE BACKSTEPPING CONTROLLER

The experimental verification of the proposed control algorithm is performed on the ERA-MOBI mobile robot. Two different methods can be used to calculate the Jacobian matrix in experiment. The first one is to read the robot output states $x(t), y(t)$ and $\theta(t)$ and use the following equation to calculate the Jacobian:

$$Jac_v = \begin{bmatrix} \frac{\partial x}{\partial v_c} & \frac{\partial x}{\partial \omega_c} \\ \frac{\partial y}{\partial v_c} & \frac{\partial y}{\partial \omega_c} \\ \frac{\partial \theta}{\partial v_c} & \frac{\partial \theta}{\partial \omega_c} \end{bmatrix} = \begin{bmatrix} \frac{x_t - x_{t-1}}{v_{c_t} - v_{c_{t-1}}} & \frac{x_t - x_{t-1}}{\omega_{c_t} - \omega_{c_{t-1}}} \\ \frac{y_t - y_{t-1}}{v_{c_t} - v_{c_{t-1}}} & \frac{y_t - y_{t-1}}{\omega_{c_t} - \omega_{c_{t-1}}} \\ \frac{\theta_t - \theta_{t-1}}{v_{c_t} - v_{c_{t-1}}} & \frac{\theta_t - \theta_{t-1}}{\omega_{c_t} - \omega_{c_{t-1}}} \end{bmatrix} \tag{5.1}$$

The problem with the above equation is that we need to have the Jacobian matrix to calculate the robot input velocities $v_c = \begin{bmatrix} v_c \\ \omega_c \end{bmatrix}$. Therefore, we should use the older data to calculate the current Jacobian as follows:

$$Jac_v = \begin{bmatrix} \frac{\partial x}{\partial v_c} & \frac{\partial x}{\partial \omega_c} \\ \frac{\partial y}{\partial v_c} & \frac{\partial y}{\partial \omega_c} \\ \frac{\partial \theta}{\partial v_c} & \frac{\partial \theta}{\partial \omega_c} \end{bmatrix} = \begin{bmatrix} \frac{x_{t-1} - x_{t-2}}{v_{c_{t-1}} - v_{c_{t-2}}} & \frac{x_{t-1} - x_{t-2}}{\omega_{c_{t-1}} - \omega_{c_{t-2}}} \\ \frac{y_{t-1} - y_{t-2}}{v_{c_{t-1}} - v_{c_{t-2}}} & \frac{y_{t-1} - y_{t-2}}{\omega_{c_{t-1}} - \omega_{c_{t-2}}} \\ \frac{\theta_{t-1} - \theta_{t-2}}{v_{c_{t-1}} - v_{c_{t-2}}} & \frac{\theta_{t-1} - \theta_{t-2}}{\omega_{c_{t-1}} - \omega_{c_{t-2}}} \end{bmatrix} \tag{5.2}$$

The other problem with the above method is that we should use the sign of the above Jacobian matrix because the practical implementation of the above equation causes the problem of singularity. Therefore the sign of the above Jacobian matrix is used to solve the singularity problem. The second method of calculating the Jacobian matrix is to use the neural network direct model as explained and simulated in the simulation results section. The only difference is that we should train the direct model neural network with the actual input and output data gathered from the robot platform sensors. Experimental verification of the proposed adaptive gain tuning controller using these two different methods of calculating the Jacobian will be explained in the following two sections.

5.3.1 Experimental trajectory tracking results using the direct Jacobian calculation method

Implementation of the adaptive backstepping controller is done using the C++ programming in Linux operating on the computer which is embedded to the ERA-MOBI mobile robot[25]. The C++ code used to formulate the controller algorithm is included in the appendix. The results of the robot trajectory tracking with the sinusoidal reference trajectory are shown in the remaining part of this section.

Note that the sign of the Jacobian matrix is used for the implementation purpose.

The controller parameters:

Gains of the adaptive controller cost function:

$$J = \frac{1}{2}\sum \gamma_x {e_x}^2 + \gamma_y {e_y}^2 + \gamma_\theta {e_\theta}^2$$

$$\gamma_x = 1, \gamma_y = 25, \gamma_\theta = 1$$

The learning rates:

$$\eta_x = 0.7, \eta_y = 0.7, \eta_\theta = 0.7$$
$$\beta_x = 0.3, \beta_y = 0.3, \beta_\theta = 0.3$$

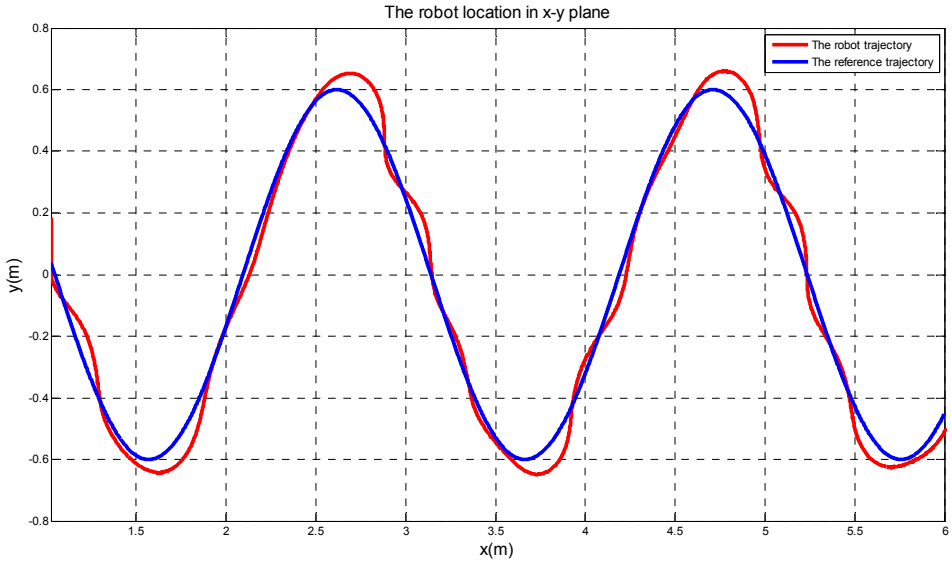

Figure 5-8: The trajectory of the robot in x-y plane VS the reference sinusoidal trajectory

The oscillation in the robot trajectory is because of the available noise in the robot sensors which are right and left wheel encoders. The above oscillation can be eliminated by filtering and signal conditioning of the sensor outputs. The robot output states errors are shown in Figure 5-9:

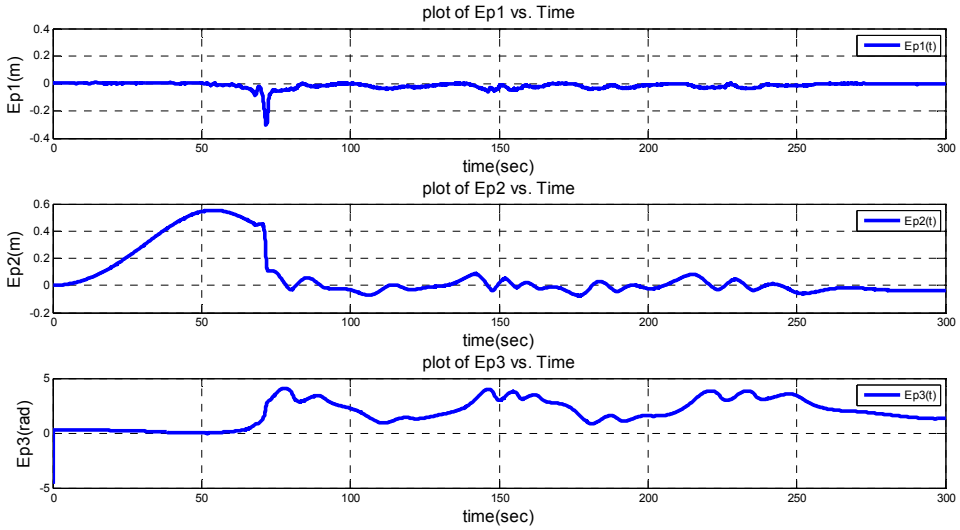

Figure 5-9: The robot output errors Ex, Ey and Eth

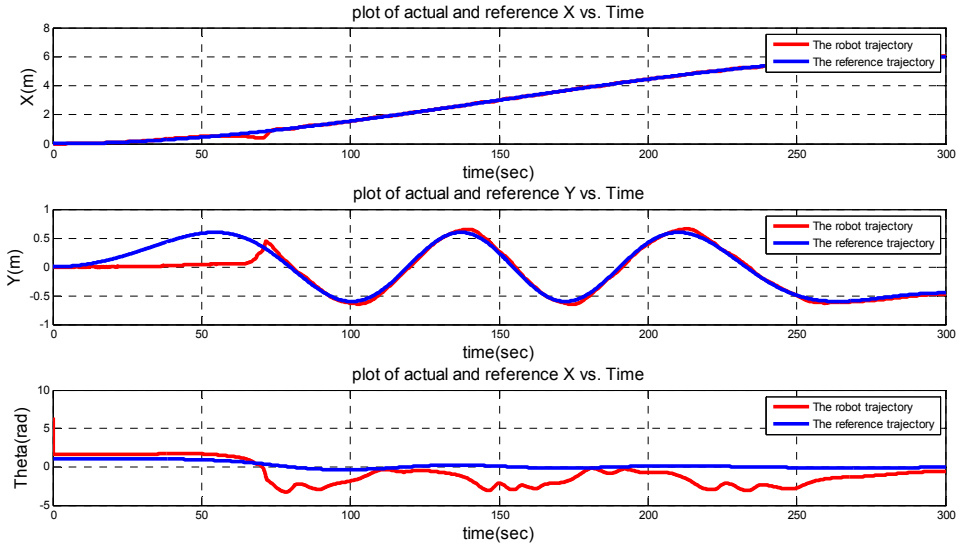

Figure 5-10: The robot output states X, Y and Theta Vs the reference trajectories

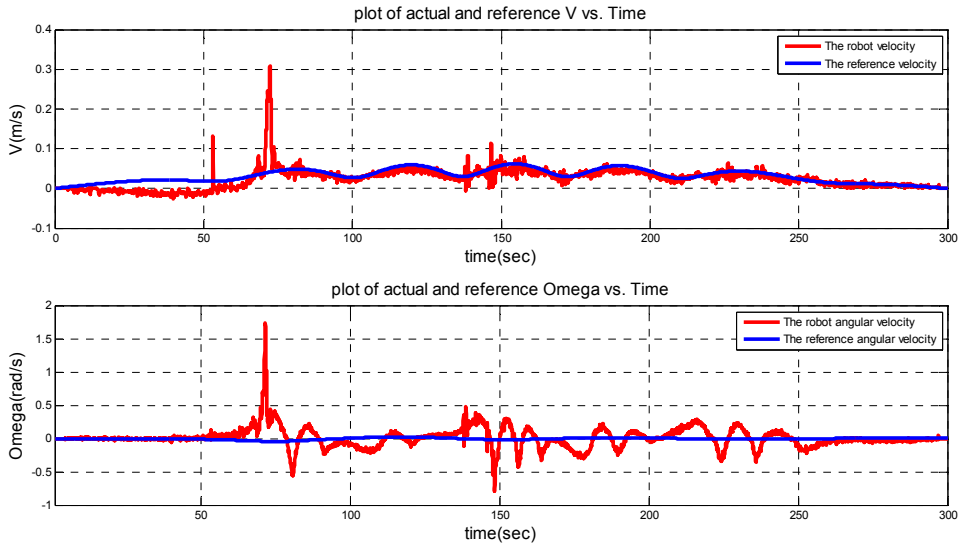

Figure 5-11: The robot velocities V and w Vs the reference trajectories

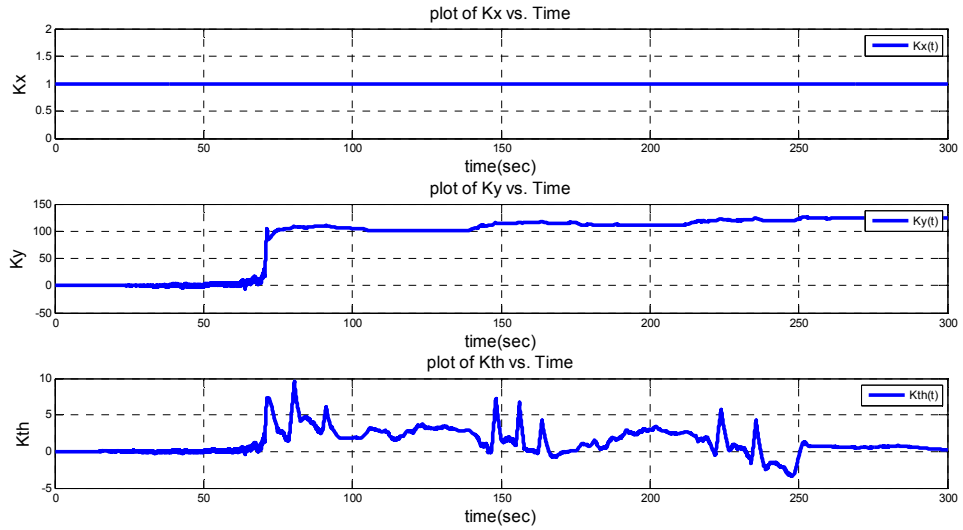

Figure 5-12: The controller gains time response

The controller gain K_x is kept constant because increasing this gain will increase the input torque to the robot wheel actuators and makes the robot unstable. The other two gains of the controller will adapt themselves to make the cost function of equation (1.5) zero. The gains become constant when the robot trajectory becomes zero and the derivative of the cost function with respect to the gains is zero. Note that the sign approximation of the Jacobian matrix is used for the above results. Using the neural network direct model to calculate the Jacobian does not have the problem of singularity

and will give us the exact Jacobian and better tracking results which will be explained in the next section.

5.3.2 Experimental trajectory tracking results using neural network direct model Jacobian calculation

Calculating the mobile robot Jacobian using the direct model neural network requires a precise and well trained neural network. In order to train the neural network and minimize its approximation error, comprehensive offline training has been done with different inputs to the system with different frequencies to excite all modal frequencies of the robot. The first stage of the offline training is to provide the robot with a chirp input and read the outputs using the robot sensors. The first chirp input given to the robot platform is shown in Figure 5-13:

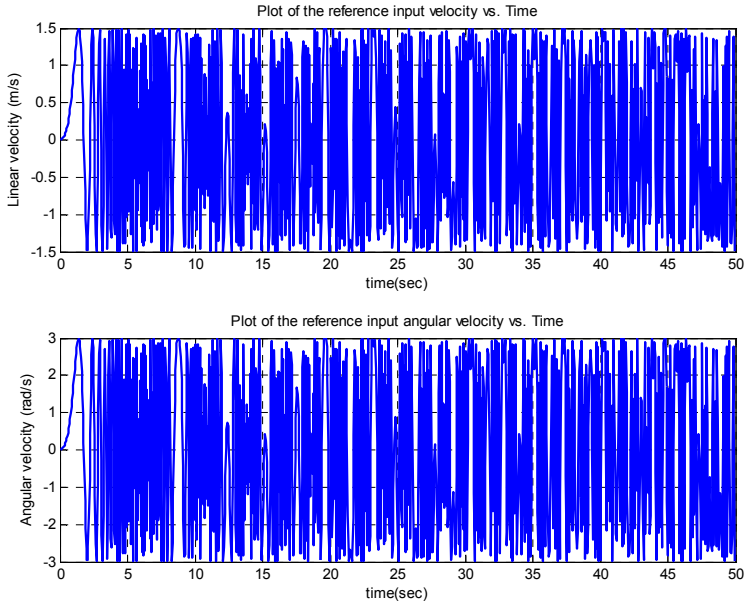

Figure 5-13: the chirp input with the frequency of 0.01 to 0.1 Hz

The frequency of the above chirp input changes between 0.01 and 0.1 Hz. Note that the system inputs are Linear and angular velocities because we are trying to construct the robot model with the robot velocities as inputs and (x, y, θ) as outputs. The outputs of the mobile robot collected from the sensors are shown in Figure 5-14:

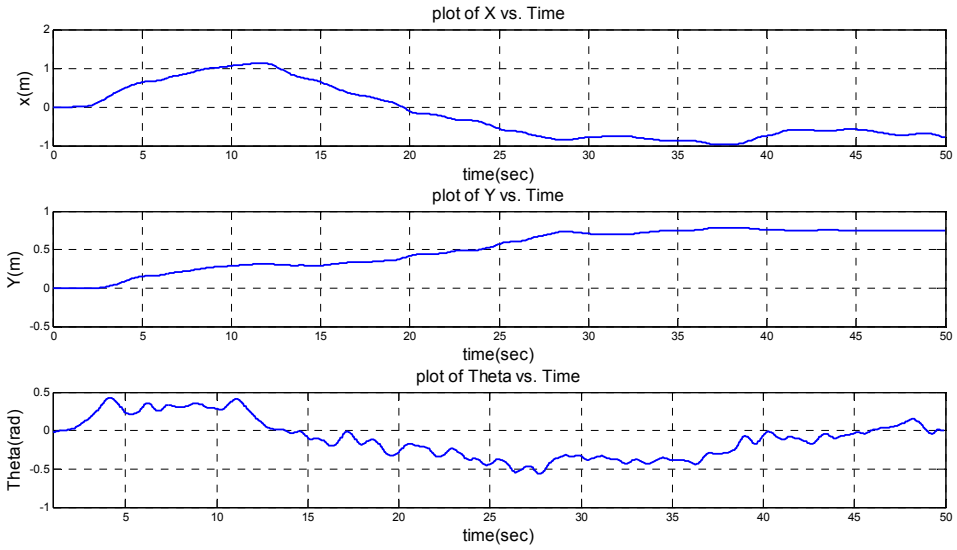

Figure 5-14: the robot output states (x, y, θ) with chirp input

The actual linear and angular velocities of the robot calculated using the collected sensors data are shown in Figure 5-15:

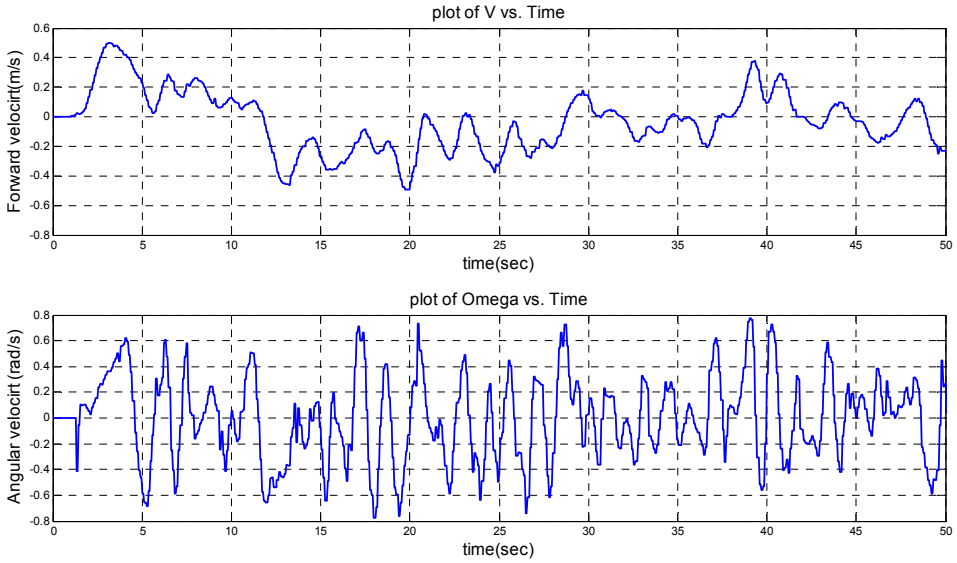

Figure 5-15: the robot actual linear and angular velocities with the chirp input

The robot trajectory in the x-y plane with the chirp input is shown in Figure 5-16:

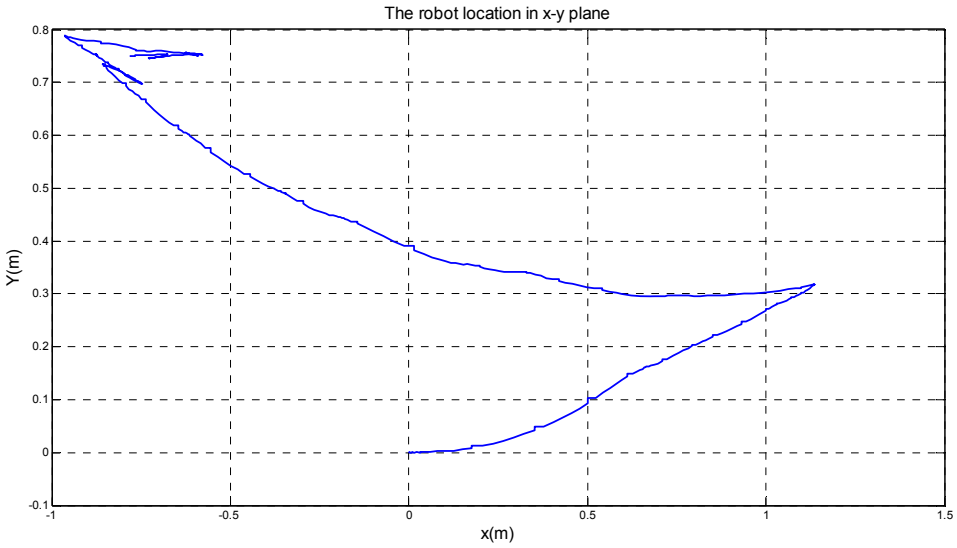

Figure 5-16: the robot trajectory in x-y plane with the chirp input

Collecting the required data for the neural network training, we can use the neural network offline training code included in the appendix to train the direct model neural network. The learning performance of the network after 500 cycles of training is shown in Figure 5-17 which shows the mean squared error:

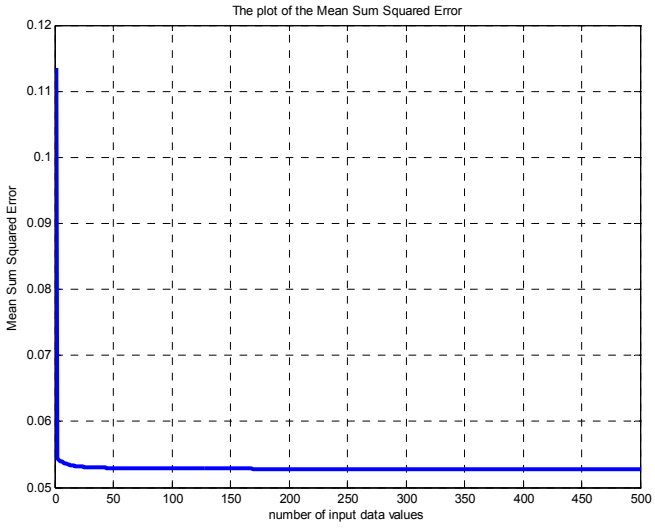

Figure 5-17: the mean squared error for 500 cycles of offline training (chirp input)

The robot outputs and neural network outputs in the last cycle of training are shown in the following figures to show the approximation performance of the neural network:

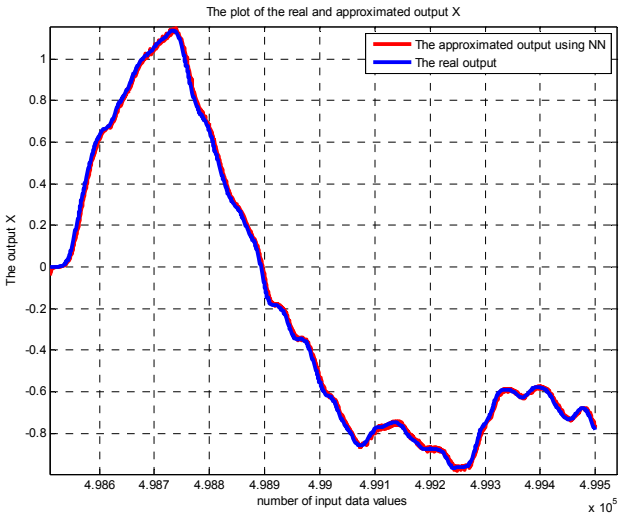

Figure 5-18: the neural network X output in the last cycle of training and the robot X output (chirp input)

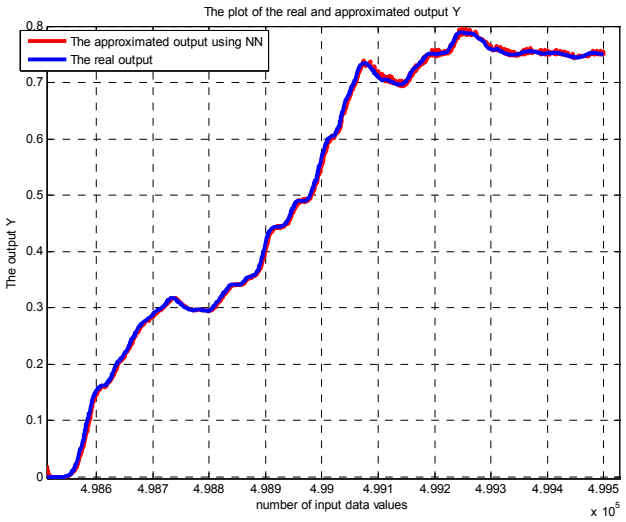

Figure 5-19: the neural network Y output in the last cycle of training and the robot Y output (chirp input)

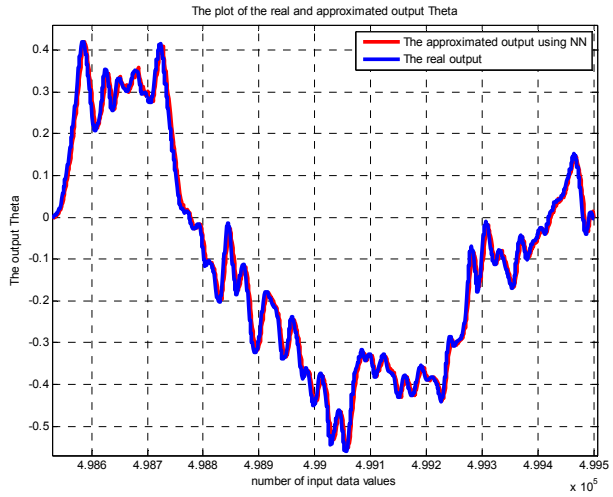

Figure 5-20: the neural network θ output in the last cycle of training and the robot θ output (chirp input)

Perfect approximation performance has been achieved after this stage of the offline training as it can be seen in the above figures. The direct model neural network has been trained with another chirp input with the frequency range of 0.1 to 1 Hz to make sure that we cover all the inner frequencies of the system. The next stage of the training process is to test the trained neural network with a sinusoidal input with a frequency within the range of the chirp input (0.05 Hz) to verify the performance of the neural network:

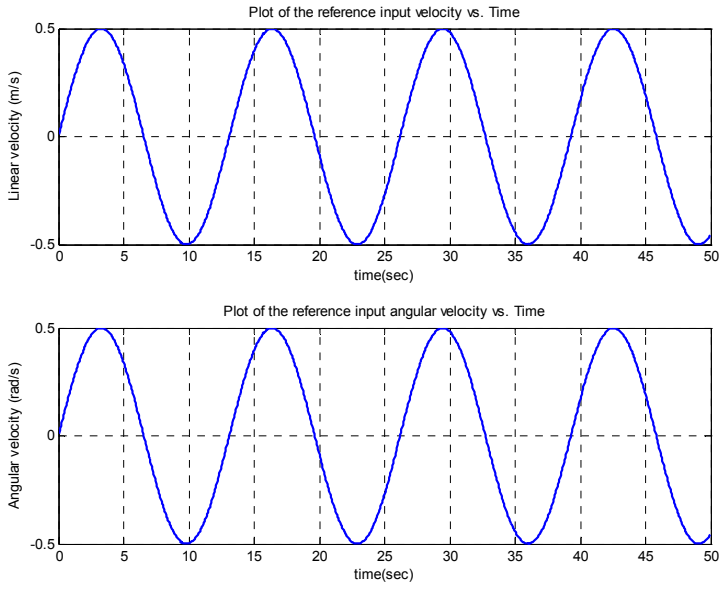

Figure 5-21: the sinusoidal input to the robot (frequency 0.05 Hz)

The outputs of the mobile robot collected from the sensors are shown in Figure 5-22:

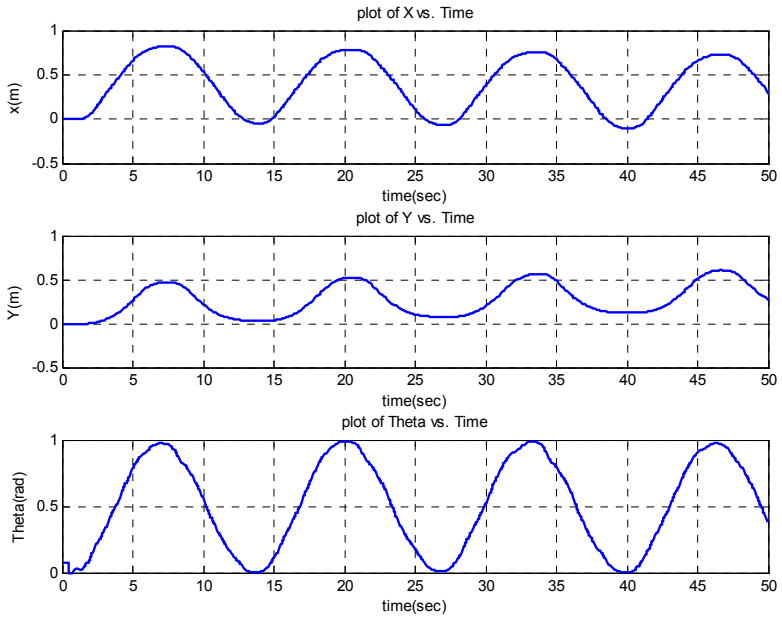

Figure 5-22: the robot output states (x, y, θ) with sinusoidal input

The actual linear and angular velocities of the robot calculated using the collected sensors data are shown in Figure 5-23:

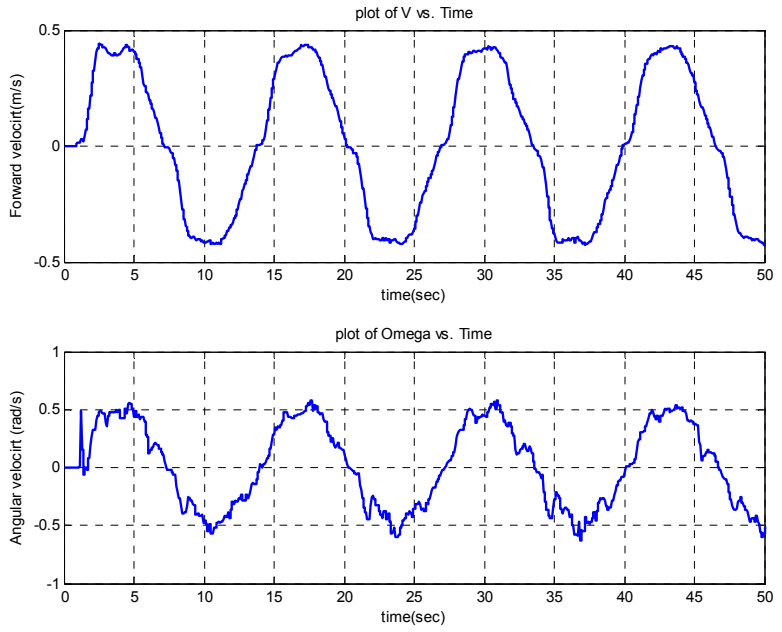

Figure 5-23: the robot actual linear and angular velocities with the sinusoidal input

The robot trajectory in the x-y plane with the sinusoidal input is shown in Figure 5-24:

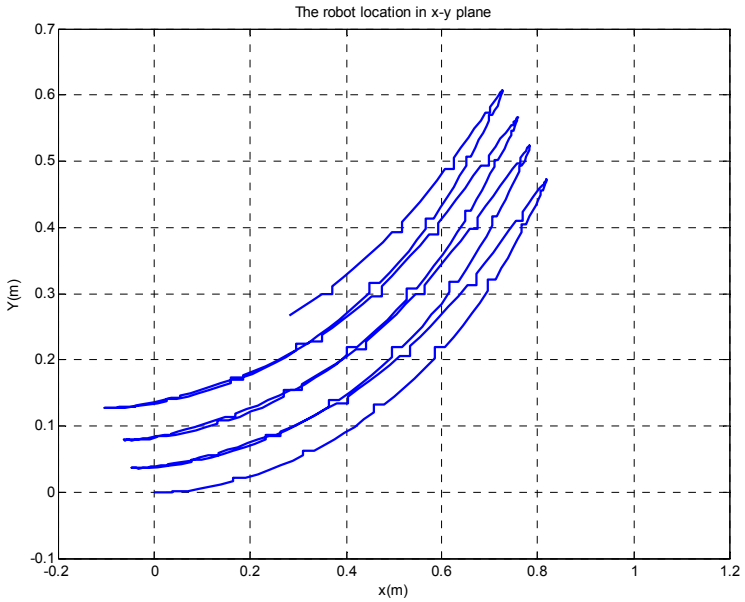

Figure 5-24: the robot trajectory in x-y plane with the Sinusoidal input

Collecting the required data for the neural network training, we can use the neural network offline training code included in the appendix to train the direct model neural network. The learning performance of the network after 500 cycles of training is shown in the following plot which shows the mean squared error:

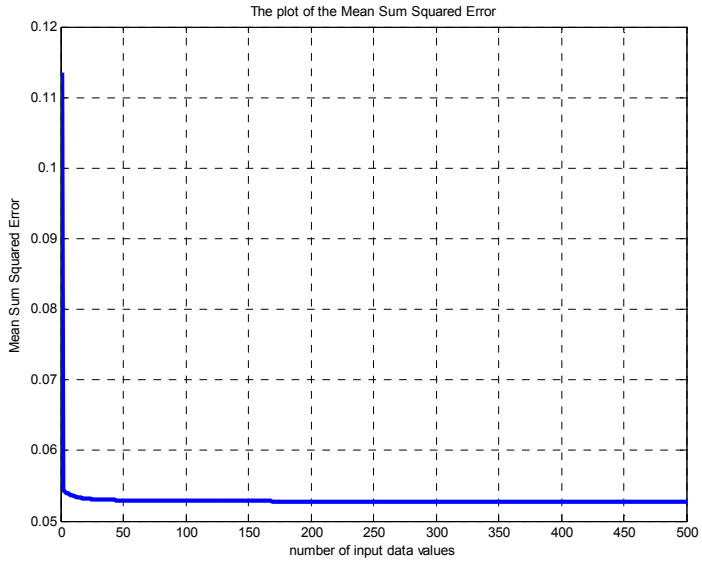

Figure 5-25: the mean squared error for 500 cycles of offline training (sinusoidal input)

The robot outputs and neural network outputs in the last cycle of training are shown in the following figures to show the approximation performance of the neural network:

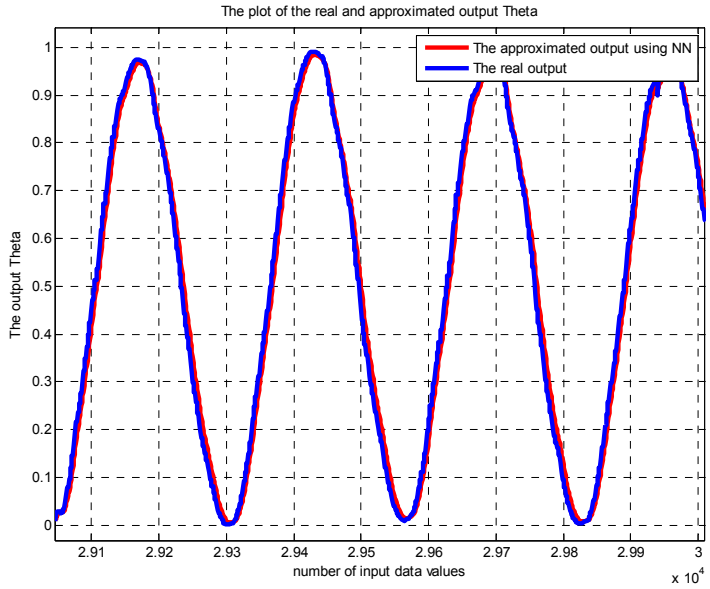

Figure 5-26: the neural network X output in the last cycle of training and the robot X output (sinusoidal input)

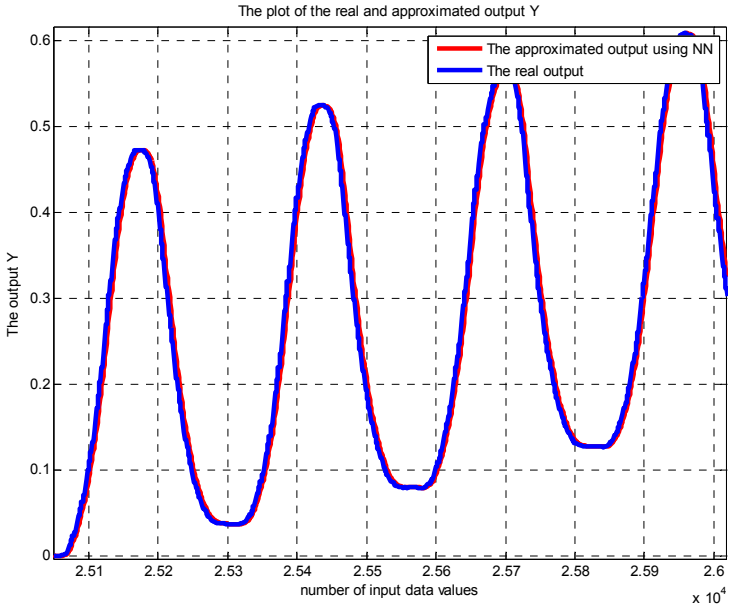

Figure 5-27: the neural network Y output in the last cycle of training and the robot Y output (sinusoidal input)

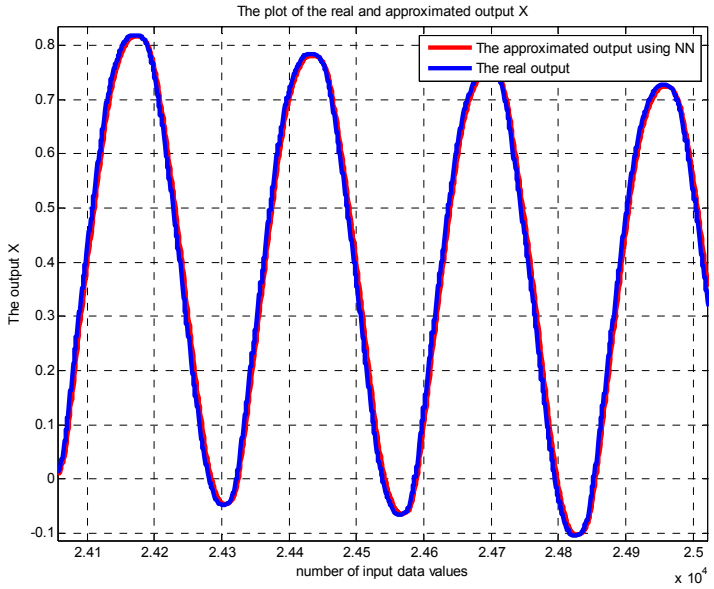

Figure 5-28: the neural network θ output in the last cycle of training and the robot θ output (sinusoidal input)

Looking at the above curves, one can find out that the direct model neural network is ready to be used for calculating the system Jacobian. The results of the mobile robot trajectory tracking with a sinusoidal reference trajectory are shown in the following figures:

Gains of the adaptive controller cost function:

$$J = \frac{1}{2}\sum \gamma_x {e_x}^2 + \gamma_y {e_y}^2 + \gamma_\theta {e_\theta}^2$$

$$\gamma_x = 1, \gamma_y = 50, \gamma_\theta = 1$$

The learning rates:

$$\eta_x = 0.9, \eta_y = 0.9, \eta_\theta = 0.9$$
$$\beta_x = 0.1, \beta_y = 0.1, \beta_\theta = 0.1$$

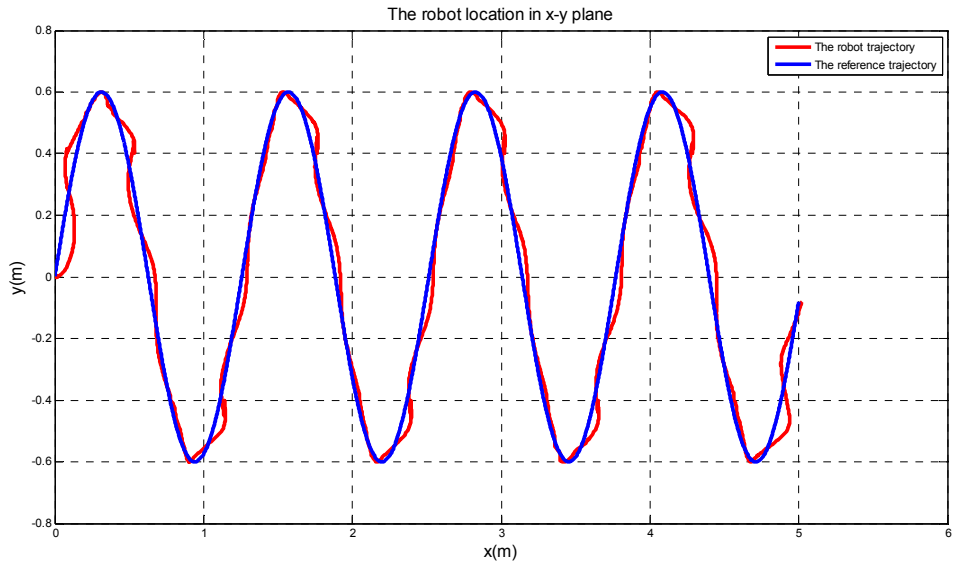

Figure 5-29: The trajectory of the robot in x-y plane VS the reference sinusoidal trajectory (using NN direct model)

A perfect and precise tracking performance has been achieved as it can be seen from the above figure. The small oscillations in the beginning of the trajectory are because of the neural network and the gradient descent algorithm learning. Once the neural network is well trained and the gradient descent algorithm which makes the errors is convergent, we achieve a good tracking as it can be seen from the above figure. The robot output states errors are shown Figure 5-30:

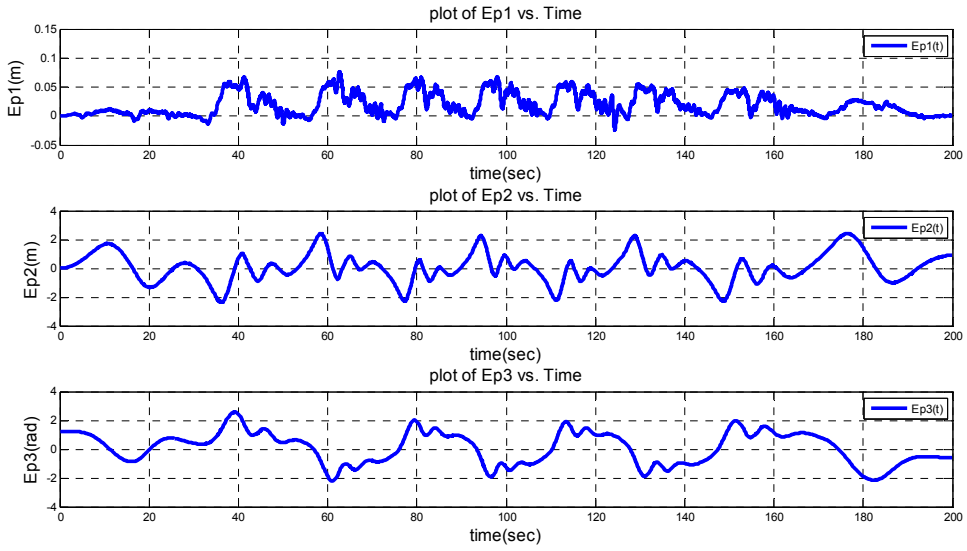

Figure 5-30: the robot output errors Ex, Ey and Eth (using NN direct model)

The time response of the robot output states and their reference trajectories are shown in Figure 5-31 to show the tracking performance of the robot states:

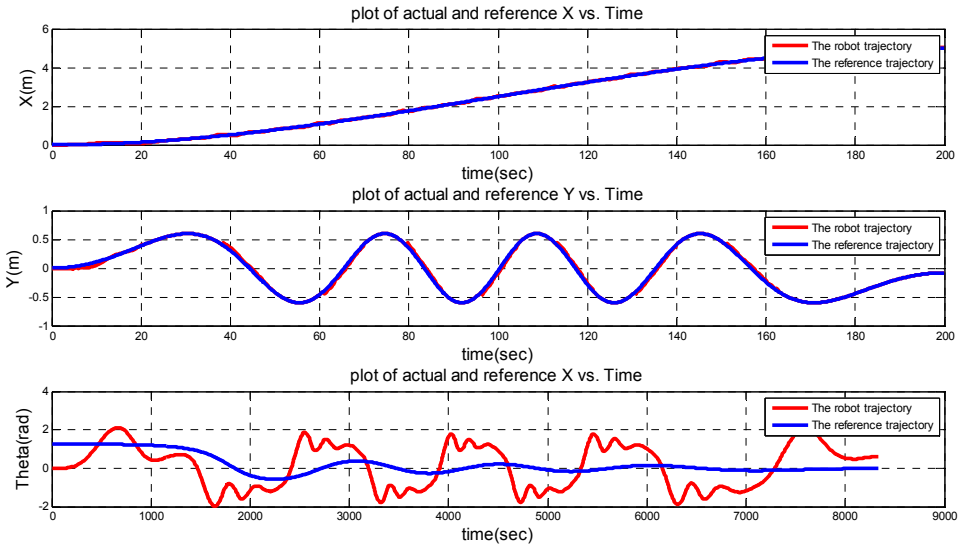

Figure 5-31: The robot output states X, Y and Theta Vs the reference trajectories (using NN direct model)

The robot linear and angular velocities and their reference trajectories are shown in Figure 5-32:

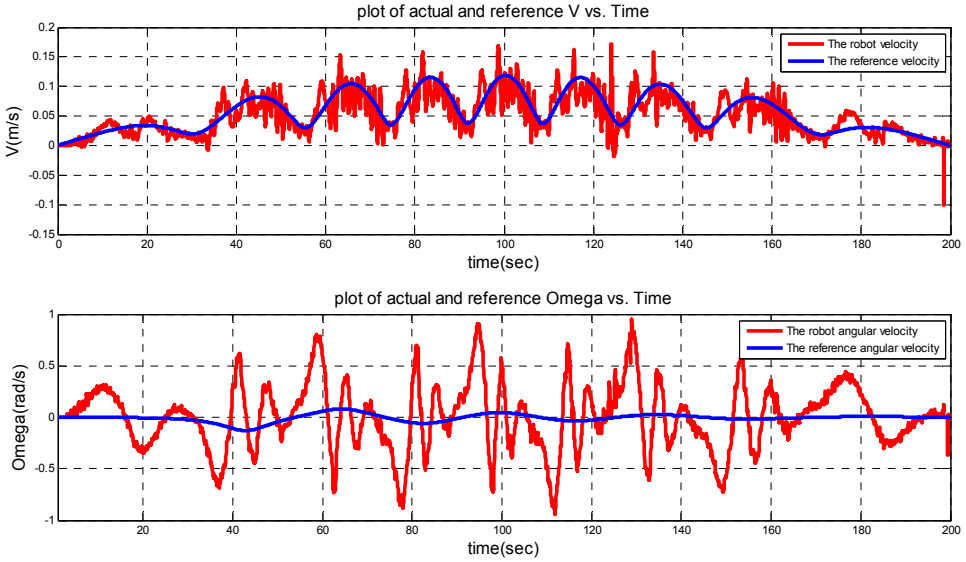

Figure 5-32: The robot velocities V and w Vs the reference trajectories (using NN direct model)

The adaptive change of the controller gains during the trajectory period is shown in Figure 5-33:

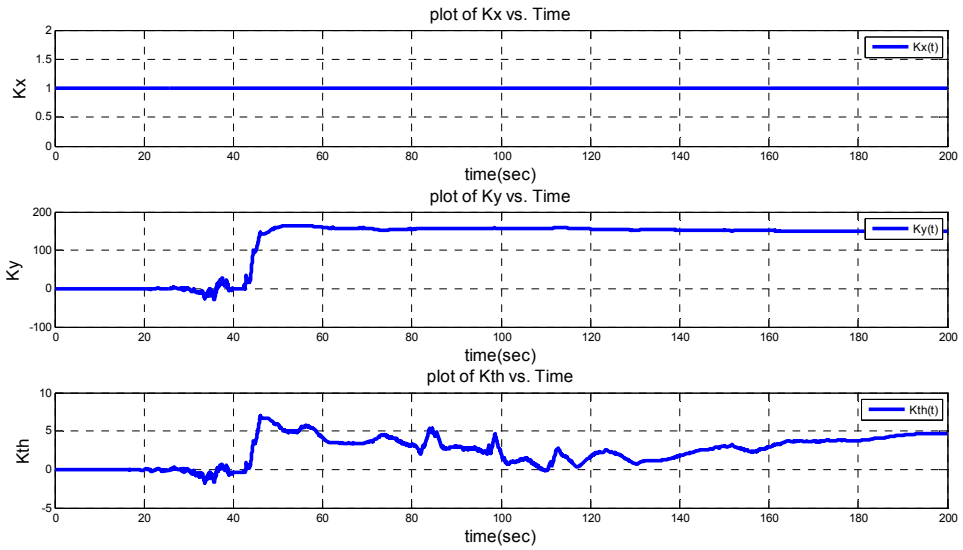

Figure 5-33: The controller gains time response (using NN direct model)

The gains of the adaptive gain tuning controller will converge to a constant value after the controller pushes the trajectory error to zero. The effect of having an adaptive gain tuning controller instead of a backstepping controller with constant gains will be clearer if we show the experimental trajectory tracking performance of the robot with the backstepping controller which will be done in the comparative analysis section. Comparing tracking results in figures (5-8) and (5-29) one can find out that the controller performs a better tracking performance when neural network direct model is used to calculate the Jacobian matrix.

The next reference trajectory to be tested by the NN-based adaptive backstepping controller is a linear trajectory. A linear trajectory is a very important trajectory because it is the basic part of each complicated trajectory and all complex trajectories can be divided to linear trajectories to be followed. The tracking results of the mobile robot with the adaptive gain tuning controller are shown in the following figures:

The controller parameters:

Gains of the adaptive controller cost function:

$$J = \frac{1}{2}\sum \gamma_x {e_x}^2 + \gamma_y {e_y}^2 + \gamma_\theta {e_\theta}^2$$

$$\gamma_x = 1, \gamma_y = 50, \gamma_\theta = 1$$

The learning rates:

$$\eta_x = 0.9, \eta_y = 0.9, \eta_\theta = 0.9$$
$$\beta_x = 0.1, \beta_y = 0.1, \beta_\theta = 0.1$$

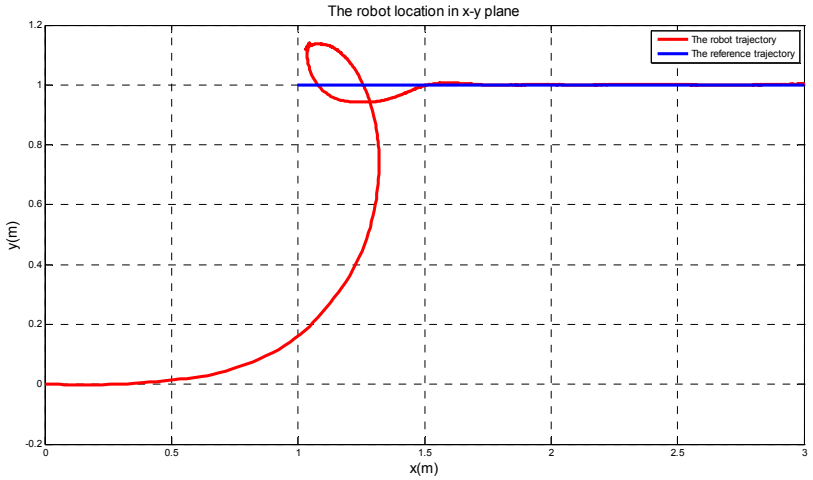

Figure 5-35: The trajectory of the robot in x-y plane VS the reference sinusoidal trajectory (using NN direct model)

The initial distance between the robot and the trajectory is about 1.4 meters which is a relatively large number. The controller is forcing the robot to reach the trajectory very fast and track the maximum possible portion of the track. This situation is useful when the robot is responsible to do a task on the track and it needs to cover the maximum part of the trajectory. The time response of the robot states are shown in Figure 5-34:

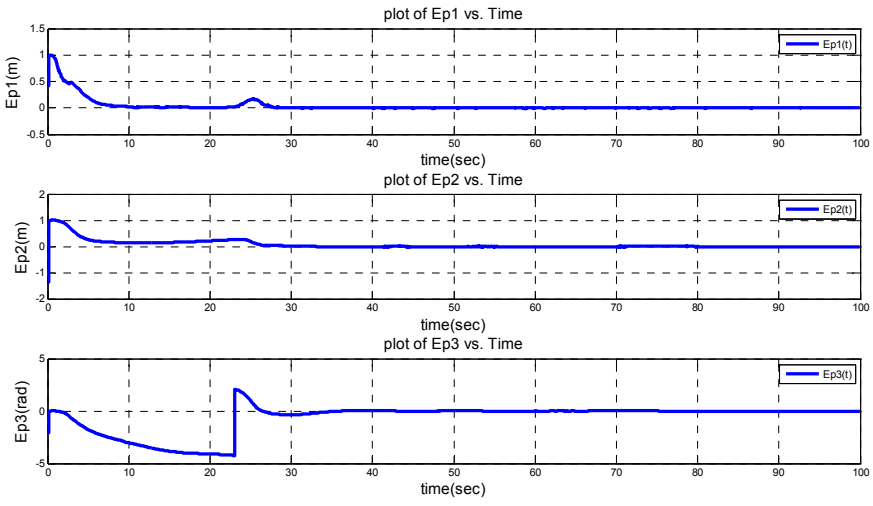

Figure 5-34: the robot output errors Ex, Ey and Eth (using NN direct model)

The error time response of the system shows that the controller will make a big effort in the beginning of the trajectory to decrease the initial large error as fast as possible. The fast response of the tracking controller can be observed from the above figures. The tracking performance of the robot states are shown in Figure 5-35:

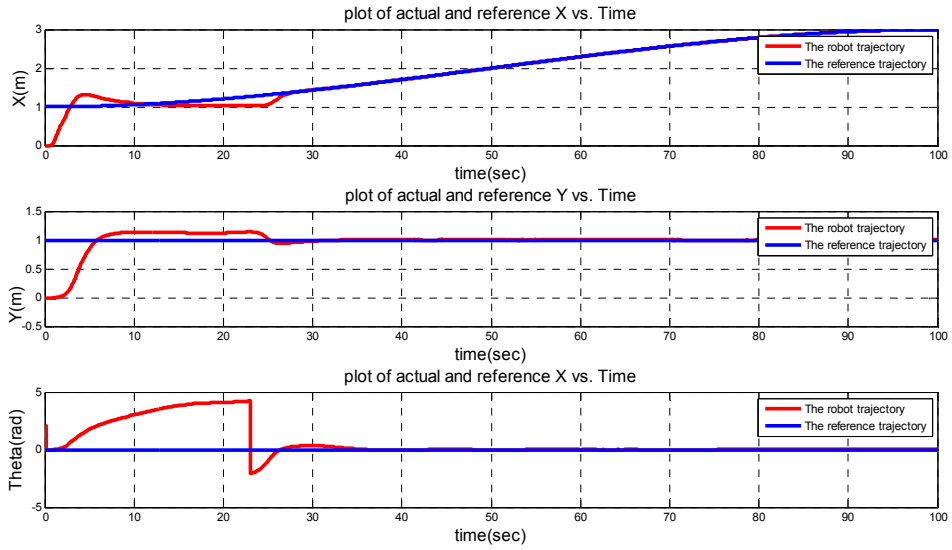

Figure 5-35: The robot output states X, Y and Theta Vs the reference trajectories (using NN direct model)

The robot linear and angular velocity performances are shown in Figure 5-36:

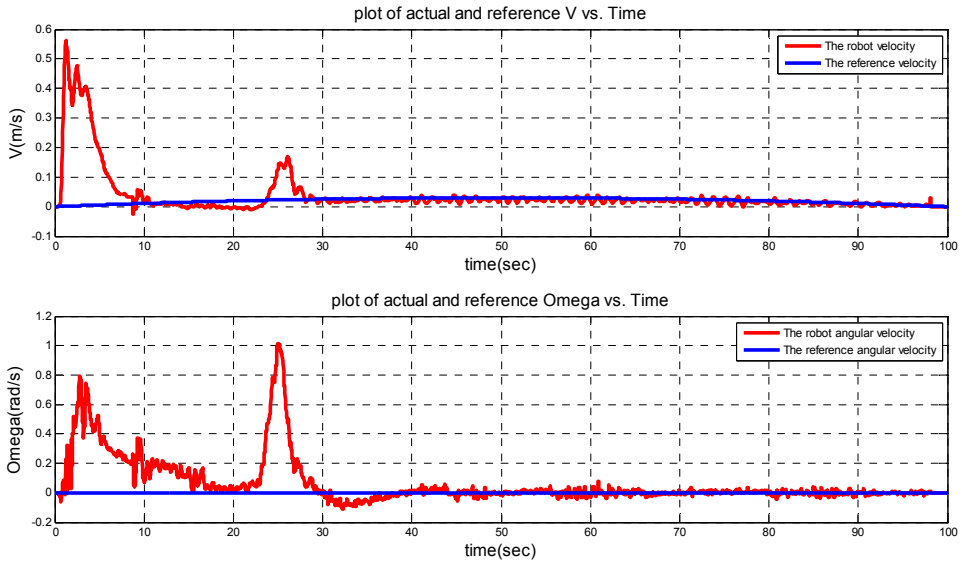

Figure 5-36: The robot velocities V and w Vs the reference trajectories (using NN direct model)

The effect of the adaptive gain tuning controller will be clearer by looking at the time response of the controller gains which is shown in Figure 5-37:

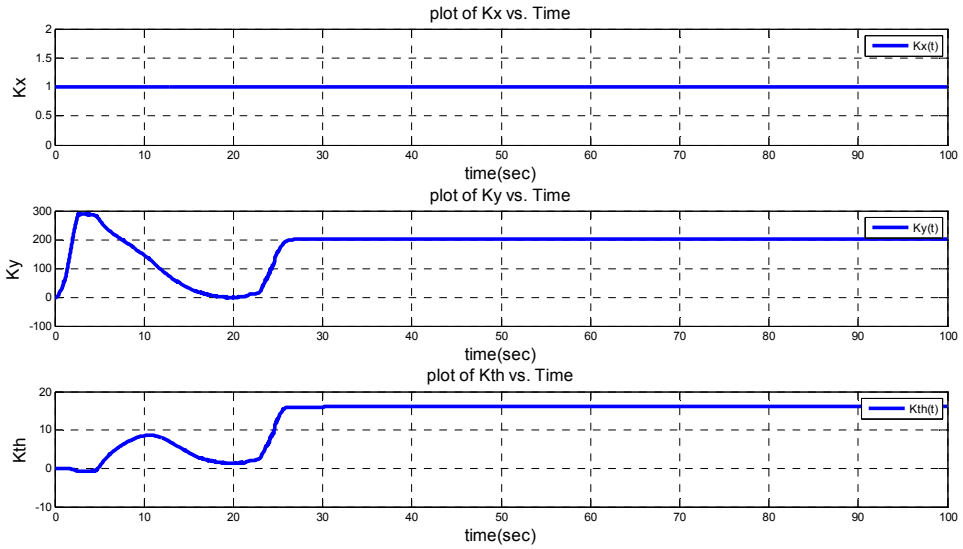

Figure 5-37: The controller gains time response (using NN direct model)

As it can be seen from the above figure, the adaptive controller gains will converge to a constant value after the controller made the tracking error zero.

5.3.3 Comparative analysis

To assess the performance of the proposed adaptive backstepping control scheme compared with the conventional backstepping controller, they have been tested with the same reference trajectory and robot initial posture and different performance indices are compared. As it is mentioned before, the sinusoidal trajectory is one of the most challenging trajectories that a robot can follow. The backstepping controller gains chosen for this comparison are the ones tuned for the sinusoidal reference trajectory. The tracking performance of the robot with NN-based adaptive backstepping and backstepping controller are shown in Figure 5-38:

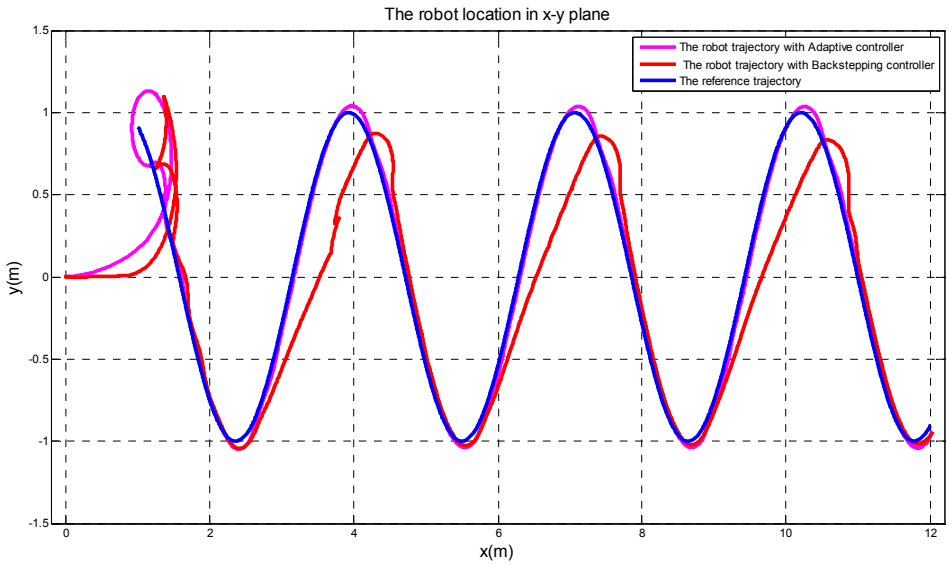

Figure 5-38: The comparison between NN-adaptive backstepping controller and backstepping controller- Robot trajectory in x-y plane

The above figure shows the tracking performance of the robot with NN-based adaptive backstepping controller and backstepping controller with the following gains:

Gains of the adaptive controller cost function:

$$J = \frac{1}{2}\sum \gamma_x {e_x}^2 + \gamma_y {e_y}^2 + \gamma_\theta {e_\theta}^2$$

$$\gamma_x = 1, \gamma_y = 20, \gamma_\theta = 1$$

The learning rates:

$$\eta_x = 0.9, \eta_y = 0.9, \eta_\theta = 0.9$$
$$\beta_x = 0.1, \beta_y = 0.1, \beta_\theta = 0.1$$

Gains of the backstepping controller:

$$K_x = 1, K_y = 120, K_\theta = 3$$

Note that the gains of the backstepping controller are carefully tuned for the sinusoidal reference trajectory on the experimental robot platform and the above tracking performance is relatively the best response that can be achieved using the backstepping controller. The time responses of the robot states with the above two controllers are shown in Figure 5-39:

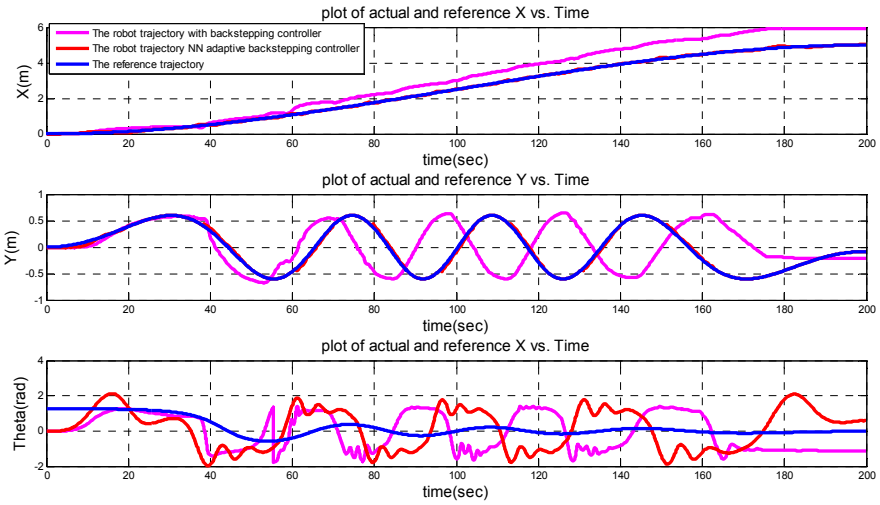

Figure 5-39: The comparison between NN-adaptive backstepping controller and backstepping controller-Robot states time response

The tracking performance improvement using the NN-adaptive backstepping controller in comparison with the conventional backstepping controller can be seen from the robot states X and Y in the above figure. Note that the oscillations in the robot heading angle is because of not using a heading sensor to read the robot heading angle. The robot angle is calculated based on $tan^{-1}(\frac{y}{x})$ equation which can cause oscillations and errors because of the noisy encoder readings. The robot states errors are shown in Figure 5-40 to compare the performance of these controllers:

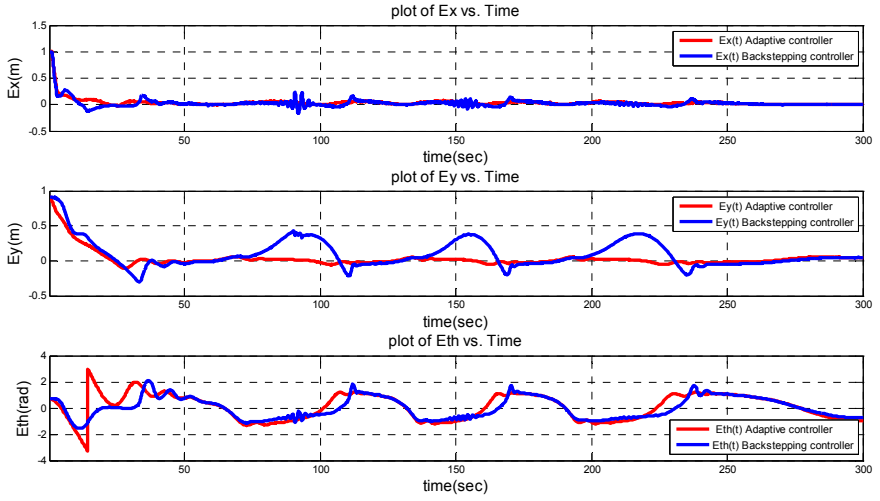

Figure 5-40: The comparison between NN-adaptive backstepping controller and backstepping controller-Robot states errors

A big decrease in the robot states errors e_x and e_y corresponding to the NN-based adaptive backstepping controller can be seen in the above figure. The robot linear and angular velocities are shown in Figure 5-41:

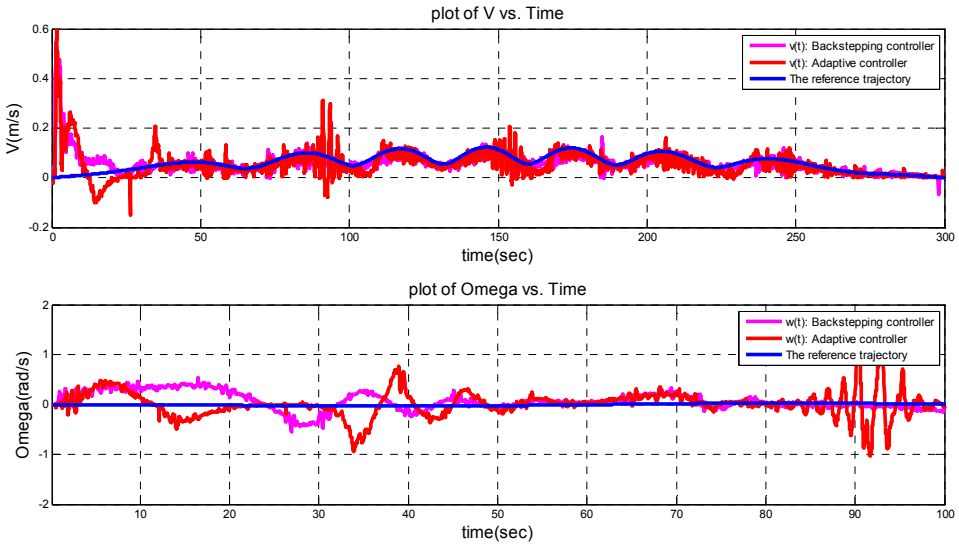

Figure 5-41: The comparison between NN-adaptive backstepping controller and backstepping controller-Robot velocities

The adaptive controller gains and the backstepping controller gains are shown in Figure 5-42:

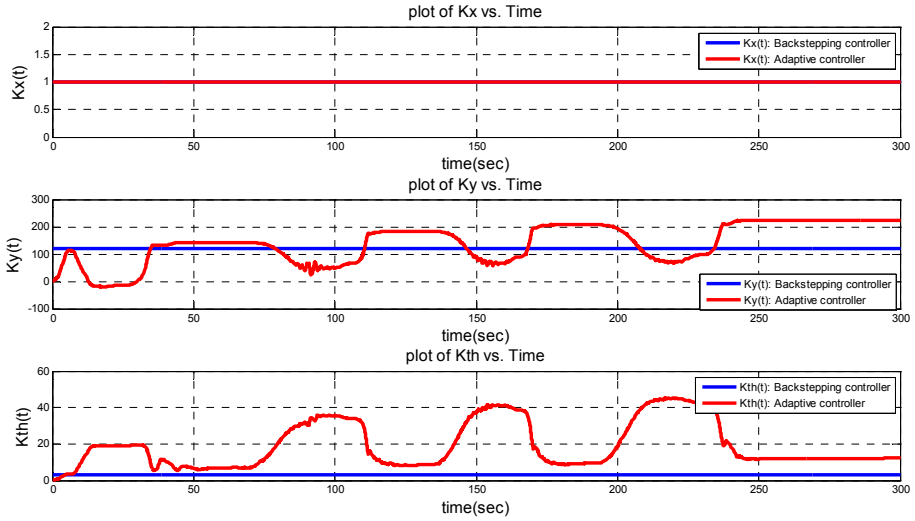

Figure 5-42: The comparison between NN-adaptive backstepping controller and backstepping controller-controller gains

Better comparison between the tracking performances of the controllers can be achieved by calculating the trajectory error based on the following equation:

$$e_{tr} = \sqrt{{e_x}^2 + {e_y}^2} = \sqrt{(x - x_r)^2 + (y - y_r)^2} \tag{5.3}$$

The trajectory errors corresponding to the above two controller is shown Figure 5-43:

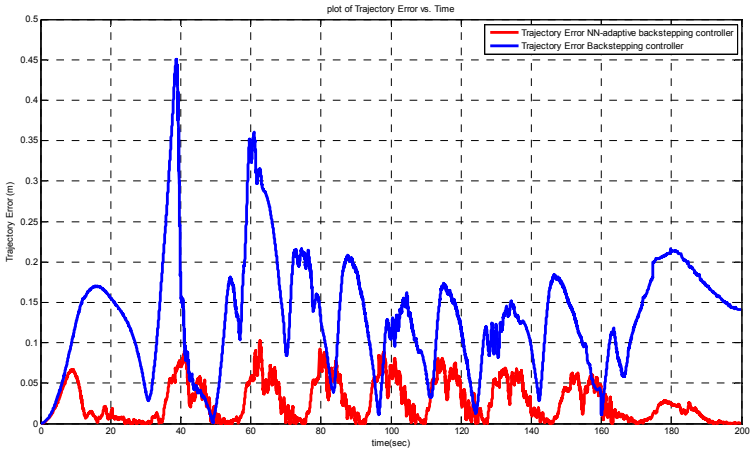

Figure 5-43: The comparison between NN-adaptive backstepping controller and backstepping controller-Trajectory errors

As it can be seen from the above figure, the trajectory error of the NN-based adaptive backstepping controller is highly reduces in comparison with the backstepping controller with fixed gains. To assess the performance of the NN-based adaptive backstepping controller compared with the conventional backstepping controller the following three performance indices are used:

1. The mean of squares of trajectory errors:

$$MSTE = \frac{1}{N_s} \sum_{k=1}^{N_s} {E_{traj}}^2 \tag{5.4}$$

2. The power spectral density of the trajectory error signal (PSD)
3. The sum of the modulus of the control signal (SMC) which in this case are the robot velocity inputs v and ω according to the following equation:

$$SMC_v = \sum_{k=1}^{N_s} |v(k)| \tag{5.5}$$

$$SMC_{\omega} = \sum_{k=1}^{N_s} |\omega(k)| \tag{5.6}$$

The first performance index which is mean squared trajectory error is computed for the trajectory errors shown in Figure (5-43) and the results are as follows:

$$Backstepping\ controller\ MSTE = 0.1335$$

$$NN - based\ adaptive\ Backstepping\ controller\ MSTE = 0.0545$$

The above values for the mean squared trajectory errors corresponding to these two controllers show that the tracking performance of the robot is highly improved using the proposed NN-based adaptive backstepping controller. The tracking improvement percentage can be calculated according to the following equation:

$$Tracking\ improvement\ percentage = \frac{MTE_{backstepping} - MTE_{adaptive}}{MTE_{backstepping}} \times 100 \tag{5.7}$$

The tracking performance of the NN-based adaptive controller in comparison with the backstepping controller is improved by 59.18% which means that the trajectory error is decreased by this percentage.

The next performance index is the power spectral density (PSD) of the trajectory error signal which describes how the power (variance) of the error signal is distributed with frequency. The PSD plot of the trajectory errors shown in Figure (5-43) is shown in Figure 5-44:

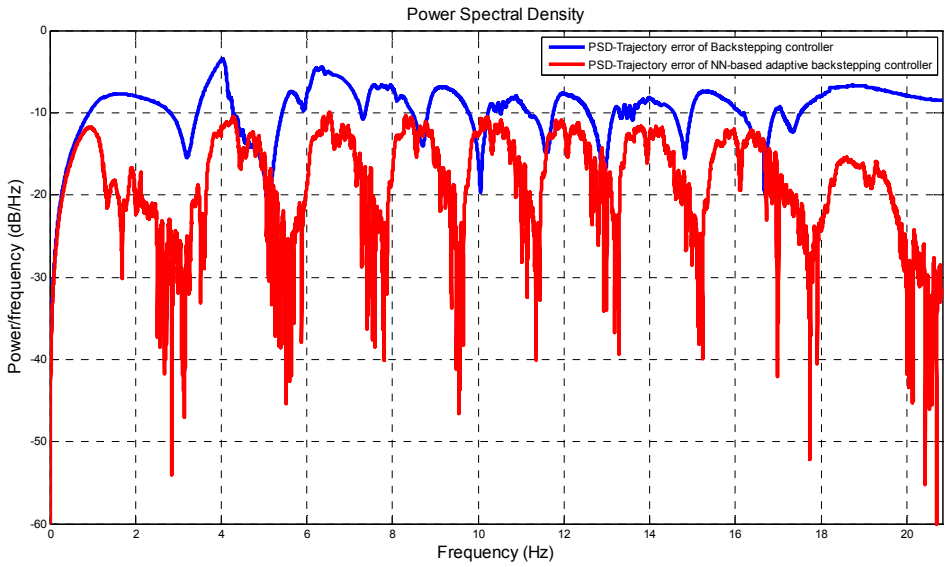

Figure 5-44: The comparison between NN-adaptive backstepping controller and backstepping controller-PSD of the trajectory errors

The above power spectral density plot confirms the tracking improvement of the robot using the NN-based adaptive backstepping controller. The next performance index is the sum of modulus of the control signal which indicates the value and power of the control signal needed for each of the above tracking performances. For the case of the above control algorithms, the control inputs are the linear and angular velocities of the robot which are calculated by the controllers. The SMC corresponding to each control signal and each controller are shown in Figure 6-45:

SMC\controller	NN-based adaptive backstepping controller	Backstepping controller
SMCv (m/s)	424.847	465.034
SMCw(rad/s)	1990.1	1872.5

Figure 5-45: The comparison between NN-adaptive backstepping controller and backstepping controller-SMC for the linear and angular velocity inputs

According to the above table, the sum of the modulus of the control signal corresponding to the linear velocity is lower for the NN-based adaptive backstepping controller by 8.64%. The sum of the modulus of the control signal corresponding to the angular velocity is higher for the NN-based adaptive controller by 5.91%. These values show that using the NN-based adaptive backstepping controller will increase the angular velocity of the robot and may cause small oscillations but decreases the linear velocity which produces smoother and more stable tracking performance.

CHAPTER 6

CONCLUDING REMARKS

A novel trajectory tracking controller for differential drive mobile robot has been designed and implemented on ERA-MOBI robot platform in AUS Mechatronics center.

6.1 SUMMARY

The followings are the summary of the works that have been done in this thesis:

- A Complete literature review on different trajectory tracking controllers for nonholonomic mobile robots has been done to place this work in the literature.
- Comprehensive and detailed modeling of a differential drive mobile robot including dynamics modeling, kinematic modeling and actuator modeling. The derived model has been simulated using Matlab Simulink and different simulation results are presented to verify the model.
- Design, derivation and simulation of the kinematic based backstepping controller for trajectory tracking of mobile robots have been done. The stability of the controller has been proved using the Lyapunov stability approach and all of the advantages and disadvantages has been discussed and analyzed.
- Design, derivation and simulation of the backstepping controller with nonlinear feedback for trajectory tracking of mobile robots has been done and presented. The stability of the controller has been proved using Lyapunov stability approach and different performance indices of the controller have been analyzed and discussed in detail.
- Development and implementation of a neural network for approximation of a general multi input multi output function. The choice of training algorithm and neural network parameters have been discussed and analyzed.
- Design, derivation and simulation of a neural network computed torque controller to improve the tracking performance of the mobile robot in presence of bounded

disturbances and uncertainties in the robot model. A control structure that integrates the backstepping controller and a neural network computed torque controller for nonholonomic mobile robots is proposed. A combined kinematic/torque control law is developed using backstepping and the stability is proved using Lyapunov approach.

- Design, derivation, simulation and implementation of a NN-based adaptive backstepping controller which highly improves the tracking performance of the backstepping controller for any kind of reference trajectory. It is an adaptive controller which tunes the gains of the backstepping controller online according to the robot reference trajectory and its initial posture. In this method, a neural network is needed to learn the characteristics of the plant dynamics and make use of it to determine the future inputs that will minimize error performance index so as to compensate the backstepping controller gains. The advantages and disadvantages of the two proposed control algorithms have been discussed and analyzed.
- Establishment of the ERA-MOBI robot system in both hardware and software aspects in AUS Mechatronics center.

6.2 CONCLUSIONS

Two different neural network based trajectory tracking controllers are proposed in this work. The comparison analysis between the neural network inverse model controller and the backstepping controller shows that this proposed controller has the following advantages over the backstepping controller:

- This controller can deal with unmodeled bounded disturbances and unstructured unmodeled dynamics in the vehicle which are a part of any environment that robot wants to perform the trajectory tracking.
- No knowledge of the system dynamics and parameters is needed in order to compensate for the nonlinear terms in the system. Especially when a readymade robot platform with unknown parameters and inner layers is used for the tracking task.

The comparative analysis between the NN-based adaptive backstepping controller and the backstepping controller shows that the proposed adaptive controller highly improves the tracking performance of the mobile robot and can adapt the tracking performance according to any kind of reference trajectory and robot initial posture. This controller is implemented successfully on the AUS mobile robot platform and the tracking performance of the robot is tested with different reference trajectories.

6.3 LIMITATIONS AND DIRECTIONS FOR FUTURE WORK

The physical mobile robot platform in AUS Mechatronics center can be improved by addition of one heading sensor for precise measurement of the heading angle of the robot. This feature will eliminate the drifting error of the robot due to encoder readings and highly improves the tracking performance.

The future research on this mobile robot platform can be in the field of path planning. Addition of the implemented trajectory tracking controllers to a path planning algorithm will complete the robot motion planning task with a very precise performance.

BIBLIOGRAPHY

[1] E. Wijn, "Unstable behavior of a unicycle mobile robot with tracking control*,"* Report No. DCT 2004.63, Eindhoven, Eindhoven University of technology,2004.

[2] R. Barzamini and A. Afshar, "Dynamic adaptive tracking control for wheeled mobile robots*,"* AmirKabir University of Technology, Tehran,2006.

[3] R.W. Brockett, R. H, "Asymptotic stability and feedback stabilization ," Boston,MA 1983.

[4] J. M. Yang and J. H. Kim, "Sliding mode control for trajectory tracking of nonholonomic wheeled mobile robots ," *IEEE Transactions on Robotics and Automation* , 578-587,1999

[5] G. Campion, G. Bastin and B. D'Andrea-Novel," Structural properties and classification of kinematic and dynamic models of wheeled mobile robots," *IEEE Transactions on Robotics and Control* , 47-62,1996.

[6] G. Campion and G. Bastin, "On adaptive linearizing control of omni-directional mobile robots,", *MTNS*, (pp. 531-538), Amsterdam,1989

[7] G. Klancar and I. Skrjanc, "Tracking-error model-based predictive control for mobile robots in real time," *Robotics and autonomous systems* , 460-469,2007.

[8] J. Ye, "Tracking control for nonholonomic mobile robots: Integrating the analog neural network into backstepping technique,". *Neurocomputing* , 3373-3378. (2007).

[9] J. Velagic, N. Osmic and B. Lacevic, " Neural Network controller for mobile robot motion control," *International journal of intelligent systems and technologies,2008* .

[10] Y. kanayama and Y. Matsuo, " A Stable tracking control method for an autonomous mobile robot*," IEEE Transactions on Robotics and Control* , arizona,1990

[11] R. Fierro and F. L. Lewis, " Control of a nonholonomic mobile robot usin neural networks," *IEEE transactions of neural networks* , 589-600,1998.

[12] R. Fierro and F. L. Lewis, “Control of a nonholonomic mobile robot: Backstepping kinematics into dynamics,” Texas: Automation and Robotics institute, The University of Texas at Arlington,1996.

[13] A. Bloch and S. Drakunov, “ Stabilization of a nonholonomic system via sliding modes,” *IEEE international conference on Decision Control*, pp. 2961-2963,1994.

[14] K. N. Faress, M. T. El hargy and A. A. El kosy, “ Trajectory tracking control for a wheeled mobile robot using fuzzy logic controller,” *Proceedings of the 9th WSEAS International Conference on Systems*, Athens, Greece,2005.

[15] K. Sangwon and P. Chongkug, “ Optimal tracking controller for an autonomous wheeled mobile robot using fuzzy-genetic algorithm,” *SPIE, the International Society for Optical Engineering,2006.*

[16] X. Jiang and M. Yung, “ Predictive fuzzy logic controller for trajectory tracking of a mobile robot,” *IEEE Mid-Summer Workshop on Soft Computing in Industrial Applications*, pp. 29 – 32,2005

[17] O. Castillo and L. T. Aguilar , “ Fuzzy logic tracking control for unicycle mobile robots,” *Advanced online publications*,2006.

[18] M. K. Bugeja and S. G. Fabri , “ Dual Adaptive Control for Trajectory Tracking of Mobile Robots,” *IEEE international conference on robotics and automation*, pp. 2215 – 2220,2007.

[19] L. Ge, R. Li, D. Yu and L. Zhao, “ A Nonlinear Adaptive Variable Structure Trajectory Tracking Algorithm for Mobile Robots,” *intelligent robotics and applications*, pp. 892-900, 2008

[20] Z. Y. Liu, R. H. Jing, X. Q. Ding and J. H. Li, “ Trajectory Tracking Control of Wheeled Mobile Robots Based on the Artificial Potential Field,” *Internationa conference on neuro computing*, pp. 382-387,2008

[21] F. Pourboghrat and M. P. Karlsson, “ Adaptive control of dynamic mobile robots with nonholonomic constraints,” Carbondale, IL 62901-6603, USA: Department of Electrical and Computer Engineering, Southern Illinois University,2002

[22] C. Dongkyoung, S. Jin, K. Pyojae and J. Young, “ Sliding mode control for trajectory tracking of nonholonomic wheeled mobile robots,” *IEEE Transactions of Robotic and Automation* , 578-587,1999.

[23] D. Gu and H. Hu, “Wavelet Neural Network based predictive control for Mobile robots,” Dept. of Engineering science, University of Essex, 2000.

[24] R. M. DeSantis, “Modeling and path-tracking control of a mobile wheeled robot with differential drive,” *Robotica*, volume 13, pp 401-410, Cambridge university press, 1995.

[25] Rev. E, “ ERA-MOBI user’s manual”, videre design,2006

Appendices

Matlab Functions

Function 1: Dynamic model

```
function u=Dynamics(Tr,Tl,v,omega,theta)

m=10;           % The robot mass
Ra=0.05;        % The radius of the wheels
a=0.05;         % The distance between the center of mass and wheels
axis
L = 0.5;        % The lateral distance of the wheels to the center
J=5;            % The robot mass moment of inertia

M=[m 0 m*a*sin(theta);
    0 m -m*a*cos(theta);
    m*a*sin(theta) -m*a*cos(theta) J];

V=[0 0 m*a*omega*cos(theta);
    0 0 m*a*omega*sin(theta)
    0 0 0];

S=[cos(theta) -a*sin(theta);
    sin(theta) a*cos(theta);
    0 1];

B=[cos(theta)/Ra cos(theta)/Ra;
    sin(theta)/Ra sin(theta)/Ra;
    L/Ra -L/Ra];

Sdot=[-omega*sin(theta) -a*omega*cos(theta);
      omega*cos(theta)   -a*omega*sin(theta);
      0 0];
Mh=S'*M*S;

Vh=[0 -m*a*omega;
     m*a*omega 0];

Bh=S'*B;
T=[Tr;Tl];
ve=[v;omega];
vdot=inv(Mh)*(T-Vh*ve);
u=[vdot(1),vdot(2)];
```

Function 2: Kinematic model

```
function u=Kinematics(v,omega,theta)

m=10;
Ra=0.05;
a=0.05;
L = 0.5;
J=5;

S=[cos(theta) -a*sin(theta);
    sin(theta) a*cos(theta);
    0 1];
ve=[v;omega];
qdot=S*ve;
u=[qdot(1),qdot(2),qdot(3)];
```

Function 3: Self-tuning controller

```
function
u=GainTuner(ex,ey,eth,theta,vr,Jacv11,Jacv12,Jacv21,Jacv22,Jacv31,Jacv3
2)

global Kx_old
global Ky_old
global Kth_old
global etaX;
global etaY;
global etaTh;
global DKx_old;
global DKy_old;
global DKth_old;
global Betax;
global Betay;
global Betath;



ep=[ex;ey;eth];

Te=[cos(theta)   sin(theta) 0;
    -sin(theta)  cos(theta) 0;
    0               0       1];
Jacv=[sign(Jacv11) sign(Jacv12);
      sign(Jacv21) sign(Jacv22);
      sign(Jacv31) sign(Jacv32)];
Der1=[ex     0        0;
      0  vr*ey  vr*sin(eth)];

Der=-ep'*Te*Jacv*Der1;
```

```
DKx=-etaX*Der(1)+Betax*DKx_old;
DKy=-etaY*Der(2)+Betay*DKy_old;
DKth=-etaTh*Der(3)+Betath*DKth_old;

DKx_old=DKx;
DKy_old=DKy;
DKth_old=DKth;


Kx=Kx_old+DKx;
Ky=Ky_old+DKy;
Kth=Kth_old+DKth;

Kx_old=Kx;
Ky_old=Ky;
Kth_old=Kth;

Kx=1;

u=[Kx,Ky,Kth];
```

Function 4: Jacobian calculation

```
function u=Jacv(t,Der11,Der12,Der21,Der22,Der31,Der32)

global m;
global J;
global a;
global Kpr;
global Kpl;

Der1=[Der11 Der12;
       Der21 Der22;
       Der31 Der32];

  Jact=[(1/m)*t 0;
          0   (1/(J+m*a^2))*t];


Der4=[Kpr   0;
        0        Kpl];

Jac=Der1*Jact*Der4;

u=[sign(Jac(1,1)),sign(Jac(1,2)),sign(Jac(2,1)),sign(Jac(2,2)),sign(Jac
(3,1)),sign(Jac(3,2))];
```

Function 5: Backstepping controller

```
function u=KinematicControl(Kx,Ky,Kth,vr,wr,ex,ey,eth)

v=vr*cos(eth)+Kx*ex;
w=wr+Ky*vr*ey+Kth*vr*sin(eth);
u=[v,w];
```

Function 6: Jacobian calculation direct model

```
function u=NNJacv(vc,wc,x,y,theta)

global x_old;
global y_old;
global theta_old;
global vc_old;
global wc_old;


Jac11=(x-x_old)/(vc-vc_old);
Jac12=(x-x_old)/(wc-wc_old);
Jac21=(y-y_old)/(vc-vc_old);
Jac22=(y-y_old)/(wc-wc_old);
Jac31=(theta-theta_old)/(vc-vc_old);
Jac32=(theta-theta_old)/(wc-wc_old);

x_old=x
y_old=y
theta_old=theta
vc_old=vc
wc_old=wc

u=[Jac11,Jac12,Jac21,Jac22,Jac31,Jac32];
```

Function 7: Trajectory generation

```
function u=trajcircle(t)

T=50; % duration of the trajectory

if (t<=50)

    xi=0.000001;
xf=2;
vxi=0;
vxf=0;

% initial and final time
t0=0;
tf=T;
%calculate the coefficients for the cubic trajectory of x
coeff=inv([1  t0  t0^2  t0^3
              0  1   2*t0  3*t0^2
              1  tf  tf^2  tf^3
              0  1   2*tf  3*tf^2])*[xi  vxi  xf  vxf]';
```

```
a0=coeff(1)
a1=coeff(2)
a2=coeff(3)
a3=coeff(4)

%evaluate x(t)
x=a0+a1*t+a2*t^2+a3*t^3;

%evaluate y(t) based on the path in the xy plane

y=(1-(x-1)^2)^0.5;
%Calculate the velicites vx and vy
vx=a1+2*a2*t+3*a3*t^2;
vy=-(x-1)*vx/y;
% vy=vx*cos(x);
v=((vx^2)+(vy^2))^0.5;
theta=atan(y/x);

elseif (t<=100)

            xi=2;
            xf=0.00000000001;
            vxi=0;
            vxf=0;

            % initial and final time
            t0=T;
            tf=2*T;

            %calculate the coefficients for the cubic trajectory of x
            coeff=inv([1  t0  t0^2  t0^3
                       0  1   2*t0  3*t0^2
                       1  tf  tf^2  tf^3
                       0  1   2*tf  3*tf^2])*[xi  vxi  xf  vxf]';
            a0=coeff(1)
            a1=coeff(2)
            a2=coeff(3)
            a3=coeff(4)

            x=a0+a1*t+a2*t^2+a3*t^3;
            %evaluate x(t)
            y=-(1-(x-1)^2)^0.5;
            %Calculate the velicites vx and vy
            vx=a1+2*a2*t+3*a3*t^2;
            vy=(x-1)*vx/y;
            % vy=vx*cos(x);
            v=((vx^2)+(vy^2))^0.5;
            theta=atan(y/x);
end
u=[x,y,vx,vy,v,theta];
```

Function 8: Rectangular trajectory generation

```
function u=trajrectangle(t)

T1=50; % duration of the trajectory
T2=50;
if (t<=T1)

xi=0.0000000001;
xf=1.8;
vxi=0;
vxf=0;

% initial and final time
t0=0;
tf=T1;
%calculate the coefficients for the cubic trajectory of x
coeff=inv([1  t0  t0^2  t0^3
           0  1   2*t0  3*t0^2
           1  tf  tf^2  tf^3
           0  1   2*tf  3*tf^2])*[xi  vxi  xf  vxf]';
a0=coeff(1)
a1=coeff(2)
a2=coeff(3)
a3=coeff(4)

%evaluate x(t)
x=a0+a1*t+a2*t^2+a3*t^3;

%evaluate y(t) based on the path in the xy plane

y=0;
%Calculate the velicites vx and vy
vx=a1+2*a2*t+3*a3*t^2;
vy=0;
% vy=vx*cos(x);
v=((vx^2)+(vy^2))^0.5;
% theta=0;
theta=atan(y/x);

elseif (t<=T1+T2)
        yi=0;
        yf=1.8;
        vyi=0;
        vyf=0;

        % initial and final time
        t0=T1;
        tf=T1+T2;

        %calculate the coefficients for the cubic trajectory of x
```

```
        coeff=inv([1  t0  t0^2  t0^3
                   0  1   2*t0  3*t0^2
                   1  tf  tf^2  tf^3
                   0  1   2*tf  3*tf^2])*[yi  vyi  yf  vyf]';
        a0=coeff(1)
        a1=coeff(2)
        a2=coeff(3)
        a3=coeff(4)

        %evaluate x(t)
        y=a0+a1*t+a2*t^2+a3*t^3;

        %evaluate y(t) based on the path in the xy plane

        x=1.8;
        %Calculate the velicites vx and vy
        vy=a1+2*a2*t+3*a3*t^2;
        vx=0;
        % vy=vx*cos(x);
        v=((vx^2)+(vy^2))^0.5;
%         theta=pi/2;
        theta=atan(y/x);

elseif (t<=T1+T2+T1)
            xi=1.8;
            xf=0;
            vxi=0;
            vxf=0;

            % initial and final time
            t0=T1+T2;
            tf=T1+T2+T1;

            %calculate the coefficients for the cubic trajectory of x
            coeff=inv([1  t0  t0^2  t0^3
                       0  1   2*t0  3*t0^2
                       1  tf  tf^2  tf^3
                       0  1   2*tf  3*tf^2])*[xi  vxi  xf  vxf]';
            a0=coeff(1)
            a1=coeff(2)
            a2=coeff(3)
            a3=coeff(4)

            %evaluate x(t)
            x=a0+a1*t+a2*t^2+a3*t^3;

            %evaluate y(t) based on the path in the xy plane

            y=1.8;
            %Calculate the velicites vx and vy
            vx=a1+2*a2*t+3*a3*t^2;
            vy=0;
            % vy=vx*cos(x);
            v=((vx^2)+(vy^2))^0.5;
%             theta=pi;
```

```
            theta=atan(y/x);

elseif (t<=2*T1+2*T2)

        yi=1.8;
        yf=0;
        vyi=0;
        vyf=0;

        % initial and final time
        t0=T1+T2+T1;
        tf=2*T1+2*T2;

        %calculate the coefficients for the cubic trajectory of x
        coeff=inv([1  t0  t0^2  t0^3
                   0  1   2*t0  3*t0^2
                   1  tf  tf^2  tf^3
                   0  1   2*tf  3*tf^2])*[yi  vyi  yf  vyf]';
        a0=coeff(1)
        a1=coeff(2)
        a2=coeff(3)
        a3=coeff(4)

        %evaluate x(t)
        y=a0+a1*t+a2*t^2+a3*t^3;

        %evaluate y(t) based on the path in the xy plane

        x=0;
        %Calculate the velicites vx and vy
        vy=a1+2*a2*t+3*a3*t^2;
        vx=0;
        % vy=vx*cos(x);
        v=((vx^2)+(vy^2))^0.5;
%         theta=pi/2;
        theta=atan(y/x);
end

u=[x,y,vx,vy,v,theta];
```

Function 9: Neural network direct model (Online training)

```
function u=NNDirectModel(v,w,x,y,theta)

%********************** INITIALIZATION
*****************************%
global Whi;
global Woh;
global Ni;
global No;
global Nh;
```

```
global eta;
global beta;


syms x1

fx=1/(1+exp(-x1));                          %The sigmoid function
fx1=x1;
fx_p=fx*(1-fx);                            %The derivative of the
sigmoid function

 %********************** Forward Propagation ************************%
 I=[1;v;w];

 Outh=subs(fx,x1,Whi*I);                                    %The output
of the hidden layer neurons
 Outh=[1;Outh];
 Outo=subs(fx1,x1,Woh*Outh);                                %The output
of the network

 Lo=subs(fx_p,x1,Woh*Outh);                                 %The output
of the derivative of the second sigmoid function
 Lh=subs(fx_p,x1,Whi*I);                                    %The output
of the derivative of the first sigmoid function

 Ex=x-Outo(1);                                             %Computing
the error of X
 Ex_sq=0.5*Ex*Ex;

 Ey=y-Outo(2);                                             %Computing
the error of Y
 Ey_sq=0.5*Ey*Ey;

 Eth=theta-Outo(2);
%Computing the error of Y
 Eth_sq=0.5*Eth*Eth;

 E_sq=Ex_sq+Ey_sq+Eth_sq;
 Em=[Ex;Ey;Eth];

 Dold=0;
 Doldi=0;
 %********************** BACK PROPAGATION
*****************************%
for k=1:No
for m=1:Nh+1

    DWoh(k,m)=Em(k)*1*Outh(m,1)*eta+beta*Dold;
%     DWoh(k,m)=Em(k,Np*(j-1)+i)*Lo(k)*Outh(m,1)*eta+beta*Dold;
    Dold=DWoh(k,m);
    Woh_old(k,m)=Woh(k,m);
    Woh(k,m)=Woh(k,m)+DWoh(k,m);

end
```

```
end

%*******Updating the weights between the input & hidden layer*********%

for p=1:Nh
for k=1:Ni+1

DWhi(p,k)=eta*Lh(p,1)*I(k,1)*[1*Em(1)*Woh_old(1,p+1)+1*Em(2)*Woh_old(2,
p+1)+1*Em(3)*Woh_old(3,p+1)]+beta*Doldi;
% DWhi(p,k)=eta*Lh(p,1)*I(k,1)*[Lo(1)*Em(1,Np*(j-
1)+i)*Woh_old(1,p+1)+Lo(2)*Em(2,Np*(j-
1)+i)*Woh_old(2,p+1)+Lo(3)*Em(3,Np*(j-1)+i)*Woh_old(3,p+1)]+beta*Doldi;
Doldi=DWhi(p,k);
Whi(p,k)=Whi(p,k)+DWhi(p,k);

end
end

u=[Outo(1),Outo(2),Outo(3),E_sq];
```

Function 10: Neural network inverse model (Online training)

```
function u=NNInversemodel(v,w,Tr,Tl)

%************************ INITIALIZATION
*****************************%
global Ni;
global No;
global Nh;
global eta;
global beta;
global Whi;
global Woh;

syms x
% fx=(exp(x)-exp(-x))/(exp(x)+exp(-x));     %The Hyperbolic tangent
function
% fx1=x;
% fx_p=1-fx^2;                              %The derivative of the
hyperbolic tangent function

fx=1/(1+exp(-x));                          %The sigmoid function
fx1=x;
fx_p=fx*(1-fx);                            %The derivative of the
sigmoid function

%********************** Forward Propagation ************************%
 I=[1;v;w];

 Outh=subs(fx,x,Whi*I);                                %The output
of the hidden layer neurons
 Outh=[1;Outh];
```

```
 Outo=subs(fx1,x,Woh*Outh);                                    %The output
of the network

%  Lo=subs(fx_p,x,Woh*Outh);                                     %The output
of the derivative of the second sigmoid function
 Lh=subs(fx_p,x,Whi*I);                                        %The output
of the derivative of the first sigmoid function

 Er=Tr-Outo(1);                                                %Computing
the error of X
 Er_sq=0.5*Er*Er;

 El=Tl-Outo(2);                                                %Computing
the error of Y
 El_sq=0.5*El*El;

 E_sq=Er_sq+El_sq;
 Em=[Er;El];

 Dold=0;
 Doldi=0;
 Whi
 %********************** BACK PROPAGATION
*****************************%
for k=1:No
for m=1:Nh+1

    DWoh(k,m)=Em(k)*1*Outh(m,1)*eta+beta*Dold;
%     DWoh(k,m)=Em(k,Np*(j-1)+i)*Lo(k)*Outh(m,1)*eta+beta*Dold;
    Dold=DWoh(k,m);
    Woh_old(k,m)=Woh(k,m);
    Woh(k,m)=Woh(k,m)+DWoh(k,m);

end
end

%*******Updating the weights between the input & hidden layer*********%

for p=1:Nh
for k=1:Ni+1

DWhi(p,k)=eta*Lh(p,1)*I(k,1)*[1*Em(1)*Woh_old(1,p+1)+1*Em(2)*Woh_old(2,
p+1)]+beta*Doldi;
% DWhi(p,k)=eta*Lh(p,1)*I(k,1)*[Lo(1)*Em(1,Np*(j-
1)+i)*Woh_old(1,p+1)+Lo(2)*Em(2,Np*(j-
1)+i)*Woh_old(2,p+1)+Lo(3)*Em(3,Np*(j-1)+i)*Woh_old(3,p+1)]+beta*Doldi;
Doldi=DWhi(p,k);
Whi(p,k)=Whi(p,k)+DWhi(p,k);

end
end
Whi
u=[Outo(1),Outo(2),E_sq,Whi(1,1),Woh(1,1)];
```

Matlab Codes

Matlab Code 1: Adaptive self-tuning controller parameters

```
global Kx_old;
global Ky_old;
global Kth_old;
global etaX;
global etaY;
global etaTh;
global m;
global J;
global a;
global Kpr;
global Kpl;
global Kdr;
global Kdl;
global Kir;
global Kil;
global DKx_old;
global DKy_old;
global DKth_old;
global Betax;
global Betay;
global Betath;
global Whi;
global Woh;
global Ni;
global No;
global Nh;
global eta;
global beta;
global x_old;
global y_old;
global theta_old;
global vc_old;
global wc_old;

x_old=0.001;
y_old=0.001;
theta_old=0.001;
vc_old=0.001;
wc_old=0.001;

Ni=2;
No=3;
Nh=20;
eta=0.9;
beta=0.7;

Whi =[-5.0318   -2.9556   -3.8485
```

```
   -5.2619   -3.8675   -1.5007
   -5.5622   -3.2744   -3.8412
   -6.9284   -3.4300   -2.1201
   -5.8478   -3.9825   -3.1088
   -5.6619   -3.9356   -2.8302
   -8.7575   -5.7148   -4.6528
   -7.1010   -6.5355   -4.3837
   -5.9365   -5.8208   -5.2414
   -5.5203   -3.4045   -2.8530
   -5.3069   -3.9712   -3.5233
   -5.6921   -3.7886   -2.5344
   -5.6470   -2.7121   -1.9751
   -7.9421   -6.3812   -3.2674
   -7.0419   -3.5427   -2.7927
   -5.7710   -3.4467   -3.5285
   -6.0289   -3.7720   -3.0979
   -5.8048   -5.2921   -3.8871
   -6.2009   -4.2881   -3.0127
   -5.1301   -3.1392   -2.1619];

Woh=[0.0327   -1.9629    0.6331    0.4982   -0.5891   -0.4417    1.0797
0.3703   -0.1929   -1.1648   -0.0405   -0.9488    0.3543   -0.6168
0.5646   -0.1698   -1.3475   -1.5264   -1.6573   -1.8202   -0.2604;
    -0.1249    1.1736    1.2860    1.5819    0.7271    1.1625    1.5804
3.1456    2.3546    3.3608    2.4768    2.6300    2.9675    0.7665
1.5268    2.0678    1.7300    4.0162    2.3752    2.4177    2.9063;
     6.3975    3.4400    1.4960   -0.2497   -2.1630   -0.7627   -2.3545
-4.7428   -4.6930   -4.1545   -4.8565   -4.7872   -5.0756   -5.6613 -
3.5709   -4.8572   -5.0920   -4.6393   -5.5487   -3.9060   -5.1640];



m=10;
J=5;
a=0.05;
Kpr=40;
Kpl=40;
Kdr=2;
Kdl=2;
Kir=0;
Kil=0;

Xi=0;
Yi=0;
Thi=0;

DKx_old=0;
DKy_old=0;
DKth_old=0;

Kx_old=0;
Ky_old=0;
Kth_old=0;

gx=1;
```

```
gy=50;
gth=1;

etaX=0.9;
etaY=0.9;
etaTh=0.9;

Betax=0.1;
Betay=0.1;
Betath=0.1;
```

Matlab Code 2: Adaptive self-tuning controller comparison plotting

```
figure(1);
plot(Xdes.signals.values,Ydes.signals.values,'r',Xk.signals.values,Yk.s
ignals.values,'m',Xgt.signals.values,Ygt.signals.values,'b:','linewidth
',3);
xlabel('X(m)');
ylabel('y(m)');
title(' The robot trajectory VS reference trajectory in xy plane
Comparison Between Backstepping and Adaptive gain tuning controller');
grid on;
legend('The reference trajectory','The robot actual trajectory with
Backstepping controller','The robot actual trajectory with Adaptive
gain tuning controller');
%*****************************************************************
***%
figure(2);
subplot(3,1,1);
plot(Xdes.time,Xdes.signals.values,'r',Xk.time,Xk.signals.values,'m',Xg
t.time,Xgt.signals.values,'b:','linewidth',3);
xlabel('Time(s)');
ylabel('X(m)');
title(' The robot X(t) response VS reference Xref(t) trajectory
Comparison Between Backstepping and Adaptive gain tuning controller ');
grid on;
legend('The reference trajectory','The robot actual trajectory with
backstepping controller','The robot actual trajectory with Adaptive
gain tuning controller');

subplot(3,1,2);
plot(Ydes.time,Ydes.signals.values,'r',Yk.time,Yk.signals.values,'m',Yg
t.time,Ygt.signals.values,'b:','linewidth',3);
xlabel('Time(s)');
ylabel('Y(m)');
title(' The robot Y(t) response VS reference Yref(t) trajectory
Comparison Between Backstepping and Adaptive gain tuning controller ');
grid on;
% legend('The reference trajectory','The robot actual trajectory with
backstepping controller','The robot actual trajectory with Adaptive
gain tuning controller');
```

```
subplot(3,1,3);
plot(ThetaDes.time,ThetaDes.signals.values,'r',Thk.time,Thk.signals.val
ues,'m',Thgt.time,Thgt.signals.values,'b:','linewidth',3);
xlabel('Time(s)');
ylabel('Theta(rad)');
title(' The robot Theta(t) response VS reference Thetaref(t) trajectory
Comparison Between Backstepping and Adaptive gain tuning controller ');
grid on;
% legend('The reference trajectory','The robot actual trajectory with
backstepping controller','The robot actual trajectory with Adaptive
gain tuning controller');
%**********************************************************************
***%
figure(3);
subplot(3,1,1);
plot(Exk.time,Exk.signals.values,'r',Exgt.time,Exgt.signals.values,'b:'
,'linewidth',3);
xlabel('Time(s)');
ylabel('ErrorX(m)');
title(' The Error Xref(t)-X(t) Comparison Between Backstepping and
Adaptive gain tuning controller');
grid on;
legend('ErrorX(t) with backstepping controller','ErrorX(t) with
Adaptive gain tuning controller');

subplot(3,1,2);
plot(Eyk.time,Eyk.signals.values,'r',Eygt.time,Eygt.signals.values,'b:'
,'linewidth',3);
xlabel('Time(s)');
ylabel('ErrorY(m)');
title(' The Error Yref(t)-Y(t) Comparison Between Backstepping and
Adaptive gain tuning controller');
grid on;
% legend('ErrorY(t) with backstepping controller','ErrorY(t) with
Adaptive gain tuning controller');

subplot(3,1,3);
plot(Ethk.time,Ethk.signals.values,'r',Ethgt.time,Ethgt.signals.values,
'b:','linewidth',3);
xlabel('Time(s)');
ylabel('ErrorTheta(Rad)');
title(' The Error Thetaref(t)-Theta(t) Comparison Between Backstepping
and Adaptive gain tuning controller');
grid on;
% legend('ErrorTheta(t) with backstepping controller','ErrorTheta(t)
with Adaptive gain tuning controllerl');
%**********************************************************************
***%
figure(4);
subplot(2,1,1);
plot(Vk.time,Vk.signals.values,'r',Vgt.time,Vgt.signals.values,'b:','li
newidth',3);
xlabel('Time(s)');
ylabel('V(m/s)');
title(' The robot forward velocity (m/s) Comparison Between
Backstepping and Adaptive gain tuning controller ');
```

```
grid on;
legend('V(m/s) with backstepping controller','V(m/s) with Adaptive gain
tuning controller');

subplot(2,1,2);
plot(wk.time,wk.signals.values,'r',wgt.time,wgt.signals.values,'b:','li
newidth',3);
xlabel('Time(s)');
ylabel('w(Rad/s)');
title(' The robot angular velocity (Rad/s)) Comparison Between
Backstepping and Adaptive gain tuning controller ');
grid on;
legend('w(rad/s) with backstepping controller','w(rad/s) with Adaptive
gain tuning controller');
%*********************************************************************
***%
figure(5);
subplot(2,1,1);
plot(Ecvk.time,Ecvk.signals.values,'r',Ecvgt.time,Ecvgt.signals.values,
'b:','linewidth',3);
xlabel('Time(s)');
ylabel('ErrorV(m/s)');
title(' The Error Vc(m/s)-V(m/s) Comparison Between Backstepping and
Adaptive gain tuning controller');
grid on;
legend('Ecv(m/s) with backstepping controller','Ecv(m/s) with Adaptive
gain tuning controller');

subplot(2,1,2);
plot(Ecwk.time,Ecvk.signals.values,'r',Ecwgt.time,Ecvgt.signals.values,
'b:','linewidth',3);
xlabel('Time(s)');
ylabel('Errorw(rad/s)');
title(' The Error wc(rad/s)-V(rad/s) Comparison Between Backstepping
and Adaptive gain tuning controller');
grid on;
legend('Ecw(rad/s) with backstepping controller','Ecw(rad/s) with
Adaptive gain tuning controller');
%*********************************************************************
***%
figure(6);
subplot(2,1,1);
plot(Trk.time,Trk.signals.values,'r',Trgt.time,Trgt.signals.values,'b:'
,'linewidth',3);
xlabel('Time(s)');
ylabel('Tr(N.m)');
title(' The right wheel Torque(N.m)) Comparison Between Backstepping
and Adaptive gain tuning controller');
grid on;
legend('Tr(N.m) with backstepping controller','Tr(N.m) with Adaptive
gain tuning controller');

subplot(2,1,2);
plot(Tlk.time,Tlk.signals.values,'r',Tlgt.time,Tlgt.signals.values,'b:'
,'linewidth',3);
xlabel('Time(s)');
```

```
ylabel('Tl(N.m)');
title(' The left wheel Torque(N.m)) Comparison Between Backstepping and
Adaptive gain tuning controller');
grid on;
legend('Tl(N.m) with backstepping controller','Tl(N.m) with Adaptive
gain tuning controller');
%*********************************************************************
***%
figure(7);
subplot(3,1,1);
plot(Kxgt.time,Kxgt.signals.values,'linewidth',3);
xlabel('Time(s)');
ylabel('Kx');
title(' The kinematic controller gain Kx change');
grid on;
legend('Kx');

subplot(3,1,2);
plot(Kygt.time,Kygt.signals.values,'linewidth',3);
xlabel('Time(s)');
ylabel('Ky');
title(' The kinematic controller gain Ky change');
grid on;
legend('Ky');

subplot(3,1,3);
plot(Kthgt.time,Kthgt.signals.values,'linewidth',3);
xlabel('Time(s)');
ylabel('Kth');
title(' The kinematic controller gain Kth change');
grid on;
legend('Kth');
```

Matlab Code 3: Neural network offline training general

```
%********************** INITIALIZATION
*****************************%

Ni=2;                                  % No of Input
No=3;                                  % No of Out put
Nh=20;                                 % No of neuron in hidden layer
eta=.3;                                % Learning ratio
beta=0.7;                              % Momentum rate
Nc=500;                                 % Number of cycles
Whi =[-6.4855   -4.0100   -4.5498
   -6.5890   -4.7984   -2.1504
   -6.5339   -3.9266   -4.3257
   -7.4260   -3.7515   -2.3720
   -6.3217   -4.2807   -3.3511
   -6.1753   -4.2071   -3.1082
   -8.9702   -5.8552   -4.7596
   -7.3099   -6.6322   -4.5009
```

```
   -6.2155   -5.9031   -5.4120
   -5.9862   -3.5501   -3.1355
   -5.9129   -4.1946   -3.8804
   -6.3110   -4.0644   -2.8849
   -6.2365   -2.9500   -2.3163
   -8.2328   -6.5682   -3.4148
   -7.1986   -3.6066   -2.8832
   -5.9831   -3.4436   -3.6779
   -6.3094   -3.8544   -3.2696
   -6.2915   -5.4654   -4.1758
   -6.5584   -4.4558   -3.2127
   -5.8346   -3.3768   -2.5838];

Woh=[0.0188   -1.9734    0.6255    0.4932   -0.5925   -0.4435    1.0799
0.3705   -0.1926   -1.1629   -0.0350   -0.9390    0.3647   -0.6056
0.5726   -0.1639   -1.3403   -1.5193   -1.6494   -1.8132   -0.2483;
    -0.0693    1.2123    1.3129    1.6006    0.7402    1.1716    1.5867
3.1500    2.3577    3.3629    2.4780    2.6302    2.9674    0.7661
1.5265    2.0676    1.7297    4.0159    2.3748    2.4173    2.9054;
     6.3216    3.3388    1.4826   -0.2754   -2.1694   -0.7430   -2.2963
-4.7001   -4.6418   -4.0895   -4.7556   -4.6468   -4.9112   -5.4908 -
3.4376   -4.7539   -4.9947   -4.5363   -5.4131   -3.7736   -4.9652];

 syms x x1 x2
fx=1/(1+exp(-x));                          %The sigmoid function
fx1=x;
fx_p=fx*(1-fx);                            %The derivative of the
sigmoid function
Np=length(Tl);                             %Number of patterns in one
set of input data
SSE_old=0;


%******************* The loop for the number of cycles Nc
****************%
for j=1:Nc;

SSE=0;
SSE_old=0;

%********** The loop for the number of patterns in one cycle Np
**********%

for i=1:Np

    %********************** Forward Propagation
************************%
    I=[1;Tl(i);Tr(i)];
    Dx(Np*(j-1)+i)=X(i);
    Dy(Np*(j-1)+i)=Y(i);
    Dth(Np*(j-1)+i)=Theta(i);
    D(:,Np*(j-1)+i)=[X(i);Y(i);Theta(i)];
%     D(Np*(j-1)+i)=subs(Out,{x1,x2},{I(2,:),I(3,:)});         %The
desired output of the system with the generated data as the input
```

```
    Outh=subs(fx,x,Whi*I);                                          %The
output of the hidden layer neurons
    Outh=[1;Outh];                                                  %Adding
one row of 1 for the biases
    Outo(:,Np*(j-1)+i)=subs(fx1,x,Woh*Outh);                        %The
output of the network
    Lo=subs(fx_p,x,Woh*Outh);                                       %The
output of the derivative of the second sigmoid function
    Lh=subs(fx_p,x,Whi*I);                                          %The
output of the derivative of the first sigmoid function
    %********************** Finding the Error
************************%
    Em(:,Np*(j-1)+i)=D(:,Np*(j-1)+i)-Outo(:,Np*(j-1)+i);

    Ex(Np*(j-1)+i)=Dx(Np*(j-1)+i)-Outo(1,Np*(j-1)+i);
%Error between the desired output and the network output
    Ex_sq(Np*(j-1)+i)=.5*(Ex(Np*(j-1)+i)*Ex(Np*(j-1)+i)');

    Ey(Np*(j-1)+i)=Dy(Np*(j-1)+i)-Outo(2,Np*(j-1)+i);
%Error between the desired output and the network output
    Ey_sq(Np*(j-1)+i)=.5*(Ey(Np*(j-1)+i)*Ey(Np*(j-1)+i)');

    Eth(Np*(j-1)+i)=Dth(Np*(j-1)+i)-Outo(3,Np*(j-1)+i);
%Error between the desired output and the network output
    Eth_sq(Np*(j-1)+i)=.5*(Eth(Np*(j-1)+i)*Eth(Np*(j-1)+i)');

    E_sq(Np*(j-1)+i)=Ex_sq(Np*(j-1)+i)+Ey_sq(Np*(j-1)+i)+Eth_sq(Np*(j-
1)+i);

    SSE=SSE_old+E_sq(Np*(j-1)+i);
    SSE_old=SSE;

    loops=0;
    Sum_DWoh=0;
    Sum_DWhi=0;
    Dold=0;
    Doldi=0;

%********************** Back Propagation ************************%
%*******Updating the weights between the hiden layer & output*********%
for k=1:No
for m=1:Nh+1

    DWoh(k,m)=Em(k,Np*(j-1)+i)*1*Outh(m,1)*eta+beta*Dold;
%     DWoh(k,m)=Em(k,Np*(j-1)+i)*Lo(k)*Outh(m,1)*eta+beta*Dold;
    Dold=DWoh(k,m);
    Woh_old(k,m)=Woh(k,m);
    Woh(k,m)=Woh(k,m)+DWoh(k,m);

end
end

%*******Updating the weights between the input & hidden layer*********%
for p=1:Nh
for k=1:Ni+1
```

```
DWhi(p,k)=eta*Lh(p,1)*I(k,1)*[1*Em(1,Np*(j-
1)+i)*Woh_old(1,p+1)+1*Em(2,Np*(j-1)+i)*Woh_old(2,p+1)+1*Em(3,Np*(j-
1)+i)*Woh_old(3,p+1)]+beta*Doldi;
% DWhi(p,k)=eta*Lh(p,1)*I(k,1)*[Lo(1)*Em(1,Np*(j-
1)+i)*Woh_old(1,p+1)+Lo(2)*Em(2,Np*(j-
1)+i)*Woh_old(2,p+1)+Lo(3)*Em(3,Np*(j-1)+i)*Woh_old(3,p+1)]+beta*Doldi;
Doldi=DWhi(p,k);
Whi(p,k)=Whi(p,k)+DWhi(p,k);

end
end



end

MSSE(j)=SSE/Np;

end



t=1:Np*Nc;
figure(1)
plot(Ex_sq,'linewidth',3);
xlabel(' number of input data values ');
ylabel( ' The error of X squared ');
title( ' The plot of the Error of X squared ');
grid on;

figure(2)
plot(Ey_sq,'linewidth',3);
xlabel(' number of input data values ');
ylabel( ' The error of Y squared ');
title( ' The plot of the Error of Y squared ');
grid on;

figure(3)
plot(Eth_sq,'linewidth',3);
xlabel(' number of input data values ');
ylabel( ' The error of Theta squared ');
title( ' The plot of the Error of Theta squared ');
grid on;

figure(4)
plot(E_sq,'linewidth',3);
xlabel(' number of input data values ');
ylabel( ' The total error squared ');
title( ' The plot of the total Error squared ');
grid on;

figure(5)
plot(t,Outo(1,:),'r',t,Dx,'b','linewidth',3);
```

```
xlabel(' number of input data values ');
ylabel( ' The output X ');
title( ' The plot of the real and approximated output X ');
legend( 'The approximated output using NN','The real output');
grid on;

figure(6)
plot(t,Outo(2,:),'r',t,Dy,'b','linewidth',3);
xlabel(' number of input data values ');
ylabel( ' The output Y ');
title( ' The plot of the real and approximated output Y ');
legend( 'The approximated output using NN','The real output');
grid on;

figure(7)
plot(t,Outo(3,:),'r',t,Dth,'b','linewidth',3);
xlabel(' number of input data values ');
ylabel( ' The output Theta ');
title( ' The plot of the real and approximated output Theta ');
legend( 'The approximated output using NN','The real output');
grid on;

figure(8)
plot(MSSE,'linewidth',3);
xlabel(' number of input data values ');
ylabel( ' Mean Sum Squared Error ');
title( ' The plot of the Mean Sum Squared Error ');
grid on;

figure(9)
plot(Outo(1,Np*(Nc-1)+1:Np*Nc-1),Outo(2,Np*(Nc-1)+1:Np*Nc-
1),'r',X,Y,'b','linewidth',3);
xlabel(' X(m) ');
ylabel( ' Y(m) ');
title( ' The plot of the robot model trajectory VS NN approximated
model trajectory ');
legend(' The trajectory of the NN approximated model ', 'The trajectory
of the robot model ');
grid on;
```