NOVEL NEURO-DYNAMIC METHODS FOR SOLVING ROBOTIC PLANNING AND CONTROL PROBLEMS

by

GHASSAN ATMEH

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON
December 2015

To everyone who believed in me.

Especially *S.*

ACKNOWLEDGEMENTS

Completing a PhD requires a lot of work. Much of that work is not possible without the help and support of a large number individuals, some of whom I may fail to mention here but that in no way makes their contribution less important. So let me start out with acknowledging everyone with whom I have met and discussed my life as a graduate student. Thank you for listening, and thank you for supporting me.

I would like to acknowledge the efforts of my research supervisor; Dr. Kamesh Subbarao. He is an amazing teacher, outstanding mentor, and an inspiration. I want to thank him for his guidance and support throughout my graduate school years. I have learned so much from him and will forever be in his debt.

I would also like to acknowledge the effort of my committee members and thank them for overseeing and participating in my work; Dr. Frank Lewis has been instrumental in my understanding of artificial neural networks and intelligent control, I would like to thank him for sharing with me his pioneering work in those areas. I would also like to acknowledge Dr. Atilla Dogan's contribution to my knowledge of control systems and unmanned systems, his teachings have guided much of my work. Another person with whom I have worked closely and gained a wealth of knowledge in dynamical systems from is Dr. Alan Bowling, I would like to thank him for that and for taking the time and putting the effort to be part of my committee. I would also like to thank Dr. Panayiotis Shiakolas for his discussion and feedback regarding my research in robotics.

Dr. Erian Armanios has been a great mentor and has helped me in many different ways throughout my graduate career. He taught me what it means to be a leader. I would like to thank him for his support. Dr. Dan Popa has also been a great source of information and encouragement, and I would like to thank him for his support as well.

A great part of the support I received was from my roommates during my PhD years. I would like to acknowledge Alok Rege, Pavan Nuthi, and Dibesh Joshi for being the best friends and roommates anyone can ask for. Their support as friends and colleagues was overwhelming. I cannot imagine how I would have made it through my PhD without it. I consider them as my brothers.

I would also like to acknowledge my lab mates from the Aerospace Systems Laboratory, especially Laura Suarez Henderson; she is a great friend and wonderful person and I would like to thank her for all that she has done for me. I want to also extend my thanks to all of my colleagues and friends at the University of Texas at Arlington for their support, most notably Isura Ranatunga for the countless hours we spent discussing humanoid robots and controls. I would be remiss if I don't mention the brilliant Andrew Somerville from which I learned a great deal about programming, as well as David Rusbarsky a great robotics engineer, and Patrick Rowe for their many roles in helping me understand of the robotics industry.

My family has been an amazing source of encouragement and support for everything I have done, and I would like to thank my Mom and Dad for being there for me and for their unconditional support. I would like to acknowledge all that they have done for me, I am truly grateful to them. I would also like to thank my brother and two wonderful sisters for being there for me when I needed them and always supporting me.

December 10, 2015

ABSTRACT

NOVEL NEURO-DYNAMIC METHODS FOR SOLVING ROBOTIC PLANNING AND CONTROL PROBLEMS

Ghassan Atmeh, Ph.D.

The University of Texas at Arlington, 2015

Supervising Professor: Kamesh Subbarao

Robotic systems are growing more complex to cater to the tasks they are required to solve. Irrespective of these tasks the problem of planning and control must be solved for all robotic systems. In this dissertation a general description of that problem is given along with the issues that it faces. This problem concerns planning a trajectory in the operational space of the robot, i.e. the space in which a task is to be executed. Then solving the inverse kinematics problem to compute the joint positions that correspond to that trajectory. And lastly, controlling the robot to track those joint trajectories.

One of the issues facing the planning and control problem is singular configurations of a robot; these are joint configurations at which the inverse kinematics problem has multiple solutions. They can also induce large velocities in a robot, and lead to damage in hardware. Another issue to be addressed while solving the planning and control problem is model uncertainties, where the model used to control the robot does not account for all possible parameters that affect its motion. The solution to the planning and control problem presented in this research uses tools from artificial intelligence and tackles these issues to provide a robust framework for planning and control that can be implemented on any robotic chain.

By using dynamic neural networks with intelligent feedback, adaptive operational space trajectories are generated. Two methods for dealing with singular configurations are presented, one uses artificial potential functions to avoid those configurations while another uses neuro-dynamic solutions to render the system robust to their effects. To control the robot, artificial neural networks with online learning are used in a neuro-adaptive controller. This controller allows for tracking the desired joint values calculated from the inverse kinematics solution.

Numerical simulations are used to verify the capabilities of the developed methods for solving the planning and control problem. These methods also tested in a realistic simulation environment of the Atlas humanoid robotic system. In house software is developed to realize an execution of a walking gait using the neuro-dynamic framework. Results show that the developed methods form a singularity robust framework for solving the planning and control problem for a complex robotic system.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

Executive Summary

The work presented in this dissertation is focused on solving the robotic planning and control problem using methods from the field of artificial intelligence. The robotic planning and control problem is broken down into three main sub problems; the operational space trajectory generation problem, the inverse kinematics problem, and the control problem. By using different kinds of artificial neural networks such as recurrent neural networks (RNN), feedforward neural networks (FFNN), and neuro-dynamic systems (Zhang neural network) these problems are solved using a neuro-dynamic framework.

This framework can be applied to any robotic chain that consists of connected links that move relative to each other. After establishing the theoretical contributions of this work, a practical application is used to demonstrate the effectiveness of the framework and verify its capabilities. Using a realistic simulation of the Atlas humanoid robotic system, the developed methods are tested. Results show that the developed neuro-dynamic framework is not only effective in solving the planning and control problem, but is also a practical framework for any robotic system.

1.1 Research Objectives

The following is a list of the objectives of this research.

1. Solve the robotic planning problem using neuro-dynamic methods.
2. Solve the robot control problem using neuro adaptive control techniques.

1.2 Research Contribution

This research contains original contributions to the field of robotics, below is a list of the major contributions of this work.

1. Developed and designed a dynamic neural network (DNN) with intelligent feedback for adaptive trajectory generation.
2. Created a novel artificial potential function method to plan joint trajectories that are singularity free.
3. Derived the neuro-dynamic model for the modified Zhang neural network (M-ZNN) to compute the time-varying solution to the damped-least-squares inverse of a matrix.
4. Used Lyapunov stability theory to prove uniform global asymptotic stability of the M-ZNN neuro-dynamic model.
5. Developed a software framework to implement all developed methods and tools on different robotic systems.

1.3 Dissertation Outline

This Dissertation details the research conducted to solve the robotic planning and control problem. Chapter 2 contains a background of robotic theory, neural networks, and neuro-dynamics. It also contains the state-of-the-art in these fields as well as the motivation for the research and the potential applications in robotics. The research problems to be solved are stated and mathematically detailed in chapter 3, this is directly followed by the solution to the operational space trajectory generation problem using a DNN with intelligent adaptive feedback in chapter 4.

A novel algorithm that uses artificial potential functions (APF) for singularity-free joint trajectory generation is presented in chapter 5. A benchmark numerical simulation of a three-link manipulator is used to verify the capabilities of the algorithm as compared to conventional methods. Chapter 6 introduces the Zhang neural network and its neuro-dynamic model for a continuous-time solution to the Moore-Penrose inverse of a matrix. In that chapter the neuro-dynamic model for the modified ZNN (M-ZNN) are derived to solve the continuous-time, damped-least-squares inverse of a matrix. Stability of the neuro-dynamic model is proven using Lyapunov theory.

The theory behind neuro-adaptive control is detailed in chapter 7, and its application to robotic systems is shown. A numerical simulation of the same system used to test the APF algorithm is used in chapter 7 to verify the ability of the DNN, M-ZNN, and neuro-adaptive controller to solve the same planning and control problem solved in chapter 5.

After establishing the theoretical contributions of the research presented in this document, the practical impact is presented in chapter 8. The Atlas humanoid robot is used as an example robotic system on which the developed neuro-dynamic methods are applied. The chapter explains how the implementation is accomplished in a realistic simulation of the Atlas robot systems using the open source simulation software Gazebo, and the robot operating system (ROS). Results from two simulation cases are presented to highlight the implementability of the neuro-dynamic framework for planning and control of robotic systems.

The document closes with the research conclusions and possibility of future work in chapter 9. This chapter is followed by appendices containing information about the robotic models used for testing the developed methods throughout the document.

CHAPTER 2

Introduction and Motivation

The field of robotics is multidisciplinary, therefore, using robots to execute a task requires solving problems that are diverse in nature. Executing a task using a robot requires moving the robot from one location to the other. To do so one must have knowledge of the robot's current location, and then decide how the robot will move to the final location. This is typically referred to as trajectory planning and control in the robotics community and is the focus of the work developed in this dissertation.

The trajectory planning problem poses a specific challenge since it is a spatio-temporal problem. Solving this problem involves generating a spatial path and a temporal trajectory for executing a task in operational space, i.e the space where tasks are executed. The operational space is typically Cartesian space and defines the position and orientation of a part of the robot, such as the end-effector for a manipulator. The forward kinematics dictate how the position and orientation of that end-effector change given values of the robot joint positions. The inverse kinematics problem, which involves solving for the joint positions given an end-effector position and orientation, is a highly nonlinear and complex problem to solve. It can also suffer from non-unique solutions at points referred to as singularities. Solving this problem however, is critical when attempting to track a desired trajectory.

The dynamics of a robotic system are nonlinear in nature and controlling the motion of a robot requires knowledge in nonlinear analysis and control. The types of nonlinearities in the system dynamics dictate how the control problem can be approached. Proving stability of the closed-loop system is mathematically intricate, but practically necessary. These reasons along with model uncertainties make controlling a nonlinear robotic system a challenging task.

The work presented in this document develops novel neuro-dynamic methods to deal with the above mentioned challenges in order to contribute to the state-of-the-art solutions currently available. Some of the solutions presented here have the advantage of reduced computation, others allow for the use of neuro-adaptive models to deal with uncertainty, while some possess the quality of being continuous in time and therefore are more in sync with the time-varying nature of the robotic system they are implemented on.

2.1 Background and the State-of-the-Art

2.1.1 The N-DOF Robotic Manipulator

The N-degree-of-freedom (DOF) manipulator can be defined as a system of $N+1$ rigid bodies connected by joints that allow for rotational or translational motion relative to each other. This system is introduced here to serve as the application on which the methods developed will be verified. The kinematics and dynamics of such a manipulator are outlined in this section. The theory for characterizing a manipulator's singular configurations along with numerically solving the inverse kinematics problem are also presented.

Forward kinematics and the Jacobian matrix

For a manipulator with N-DOF the vector of DOF, $\mathbf{q} \in \mathrm{I\!R}^{\mathrm{N}\times 1}$, can be defined as $\mathbf{q} = [q_1 \quad q_2 \quad q_3 ... \, q_N]^T$, and the vector of end-effector pose in operational (Cartesian) space is $\mathbf{x}_e \in \mathrm{I\!R}^{\mathrm{M}\times 1}$, where M can have a maximum value of 6. In that case one can define $\mathbf{x}_e = [x_e \quad y_e \quad z_e \quad \phi_e \quad \theta_e \quad \psi_e]^T$ where x_e, y_e, and z_e are the Cartesian position vector components and ϕ_e, θ_e, and ψ_e are the roll, pitch, and yaw angles perspectively of the end-effector. A multi-link manipulator is graphically depicted in Figure 2.1.

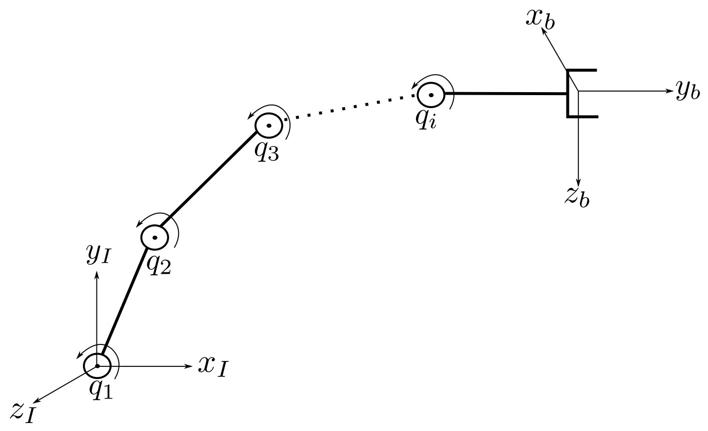

Figure 2.1. Graphical depiction of a multi-link manipulator.

The forward kinematics for the manipulator can be expressed in the following compact, nonlinear form

$$\mathbf{x}_e = f(\mathbf{q}) \tag{2.1}$$

where $\{f(.) : \mathbb{R}^{N\times 1} \rightarrow \mathbb{R}^{M\times 1}\}$. Therefore,

$$\dot{\mathbf{x}}_e = \frac{\partial f(\mathbf{q})}{\partial \mathbf{q}}\dot{\mathbf{q}} \tag{2.2}$$

where $\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{M\times N} \equiv \frac{\partial f(\mathbf{q})}{\partial \mathbf{q}}$ is the Jacobian matrix. In the case where $N > M$ the manipulator is referred to as a redundant one.

The inverse kinematics

The inverse kinematics (IK) problem requires finding the manipulator configuration given an end-effector position and orientation, i.e

$$\mathbf{q} = g(\mathbf{x_e}) \tag{2.3}$$

where $\{g(.) : \mathbb{R}^{\mathrm{M}\times 1} \rightarrow \mathbb{R}^{\mathrm{N}\times 1}\}$ which can be clearly seen to be the inverse of $f(.)$. The existence of the inverse is not guaranteed for all $\mathbf{x}_e \in \mathbb{R}^{\mathrm{M}\times 1}$. Additionally, there exist $\mathbf{x}_e$ where computation of $g\left(\mathbf{x_e}\right)$ is computationally expensive. While analytical solutions exist for a limited set of problems (typically small values of N), numerical solutions are typically sought for larger numbers of links. For this purpose, the differential kinematics is Eq. 2.2 can be rewritten to use numerical integration for solving the IK problem as follows

$$\dot{\mathbf{q}} = \mathbf{J}^{-1}\left(\mathbf{q}\right)\dot{\mathbf{x}}_e \tag{2.4}$$

The numerical solution to the IK problem can be obtained by integrating Eq. 2.4 as follows

$$\mathbf{q}(t) = \int_0^t \dot{\mathbf{q}}(\zeta)d\zeta + \mathbf{q}(0) \tag{2.5}$$

However, since numerical integration suffers from *drift*, the operational space error can be used to acquire a better solution. Define the operational error $\mathbf{e}\left(\mathbf{t}\right) \in \mathbb{R}^{\mathrm{M}\times 1}$ as the error between the desired and actual end-effector position and orientation as follows

$$\mathbf{e}\left(\mathbf{t}\right) = \mathbf{x}_d\left(\mathbf{t}\right) - \mathbf{x}_e\left(\mathbf{t}\right) \tag{2.6}$$

therefore its derivative is

$$\dot{\mathbf{e}}\left(\mathbf{t}\right) = \dot{\mathbf{x}}_d\left(\mathbf{t}\right) - \dot{\mathbf{x}}_e\left(\mathbf{t}\right) \tag{2.7}$$

which according to Eq. 2.2 can be written as

$$\dot{\mathbf{e}}(t) = \dot{\mathbf{x}}_d\left(\mathbf{t}\right) - \mathbf{J}\left(\mathbf{q}\right)\dot{\mathbf{q}}\left(\mathbf{t}\right) \tag{2.8}$$

The evolution of the error over time is given in Eq. 2.8, therefore by choosing a relationship between $\dot{\mathbf{q}}\left(\mathbf{t}\right)$ and $\mathbf{e}\left(\mathbf{t}\right)$ that ensures the error converges to zero, one can solve the IK problem. The following relationship is chosen

$$\dot{\mathbf{q}}\left(\mathbf{t}\right) = \mathbf{J}^{-1}\left(\mathbf{q}\right)\left(\dot{\mathbf{x}}_d\left(\mathbf{t}\right) + \mathbf{K}\mathbf{e}\left(\mathbf{t}\right)\right) \tag{2.9}$$

since it gives the following exponentially stable error system [1]

$$\dot{\mathbf{e}}\left(\mathbf{t}\right)+\mathbf{K}\mathbf{e}\left(\mathbf{t}\right)=0 \tag{2.10}$$

where the matrix $\mathbf{K}=\mathbf{K}^T>0$ is positive definite and is typically selected to be a diagonal matrix. A block diagram depicting the numerical solution to the IK problem is given in Figure 2.2.

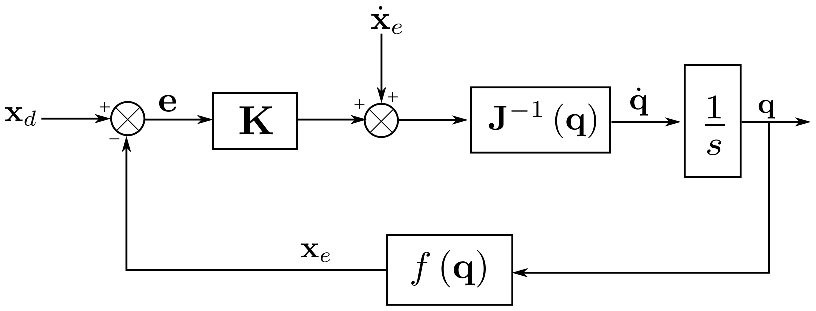

Figure 2.2. Block diagram showing the numerical solution to the IK problem.

Dynamics

The dynamics of a robot manipulator are typically derived using energy methods (Lagrange's equations), the iterative Newton-Euler method, Kane's method and others. Not withstanding the method employed, the governing equations take the following form

$$\mathbf{M}\left(\mathbf{q}\right)\ddot{\mathbf{q}}+\mathbf{V}\left(\mathbf{q},\dot{\mathbf{q}}\right)+\mathbf{G}\left(\mathbf{q}\right)=\boldsymbol{\tau}+\boldsymbol{\tau}_E \tag{2.11}$$

where $\mathbf{M}(\mathbf{q})\in\mathbb{R}^{\mathrm{N}\times\mathrm{N}}$ is symmetric mass matrix, $\mathbf{V}(\mathbf{q},\dot{\mathbf{q}})\in\mathbb{R}^{\mathrm{N}\times 1}$ is the Coriolis/centripetal vector, $\mathbf{G}(\mathbf{q})\in\mathbb{R}^{\mathrm{N}\times 1}$ is the gravity vector, $\boldsymbol{\tau}\in\mathbb{R}^{\mathrm{N}\times 1}$ is the input torque vector, and $\boldsymbol{\tau}_E\in\mathbb{R}^{\mathrm{N}\times 1}$ represents the unmodeled disturbance torques.

2.1.2 Manipulator Kinematic Singularities

Singularities in a robot structure represent joint position configurations at which unpredictable motion is produced, they are configurations where there exist infinite solutions to the inverse kinematics problem. Operating in the neighborhood of a singularity can induce large velocities in the joint space which can cause damage in the physical hardware [1]. This makes avoiding singularities a requirement during operation.

The Jacobian matrix is a function of the manipulator configuration, if there exist a configuration that causes the Jacobian to be rank-deficient that configuration is a singularity. For a square Jacobian $\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{\mathrm{N}\times\mathrm{N}}$, i.e $M = N$, the determinant of the Jacobian, $\mathbf{det}(\mathbf{J}(\mathbf{q}))$, can be used to examine the singularities. However, for redundant manipulators $\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{\mathrm{M}\times\mathrm{N}}$, with $(N > M)$ the determinant the square matrix $\mathbf{J}(\mathbf{q})\,\mathbf{J}^T(\mathbf{q})$, or $\mathbf{det}\left(\mathbf{J}(\mathbf{q})\,\mathbf{J}^T(\mathbf{q})\right)$ can be examined [2].

Since singularities can cause the Jacobian matrix to become non invertible, there exists the need to either avoid the singularity or minimize its effect in planning and control tasks. This is generally done using two approaches, the first involves designing a trajectory that is singularity free. This requires knowledge of the singular configurations a priori. The second on the other hand attempts to make the system robust to singularities; wherein the focus is on ensuring that the Jacobian matrix is well conditioned for inversion despite not having full knowledge of the singular configurations beforehand.

The topic of singularity avoidance has been studied extensively over the past two decades. The concept of path and trajectory planning for robotic manipulators while avoiding singularities has also been studied using various approaches. One approach uses optimal control to formulate the problem; by establishing bounds on the rate of change of the Jacobian matrix and utilizing them as constraints for an optimization problem along with other constraints, an optimal control problem is solved numerically to plan an optimal trajectory for both redundant and non-redundant manipulators while avoiding singularities [3]. To avoid wrist singularity, combined analytical and pseudoinverse inverse kinematic solutions are used to account for angles that do not actively participate in redundancy and others that do [4]. This is done to reduce the computational complexity of inverse kinematic solutions and solve the problem of orienting a robot forearm in order to prevent wrist singularity. Other methods that insure singularity-robust inverse kinematics are ones which utilize performance indices, specifically the reduced minor index. This approach prevents robot manipulators from generating excessive joint torques. The reduced minor index allows for a singularity-robust inverse of the Jacobian matrix [5]. Artificial neural networks (ANN) have also been used for fast inverse kinematics solutions with effective singularity prevention in redundant manipulators; here, bounded geometrical concepts are used to create characteristic matrices which help generate a performance index and a null space vector that is computed using an ANN, this allows for singularity free path planning [6]. ANN are also used for singularity-free trajectory planning and obstacle avoidance for parallel manipulators [7]. Another approach for singularity avoidance is the on-line task modification method (OTMM). This method uses a correction vector that is constructed from the task velocity and a singular vector corresponding to the minimum singular value. The method is based on the observation of the geometric relation between the task velocity and the singular vector at singular configurations [8]. Geometrical inverse kinematics methods have also been used to avoid singularities especially in the case of hyper redundant manipulators. This is done by setting the angles between the adjacent links equal to each other, this renders any two or more joint axes lining up a minimal occurrence [9]. Other techniques are ones that use singularity-robust task-space control schemes. One of which utilizes two controllers; one joint-space position controller and a task-space tracking controller. The closed-loop stability of such a controller is studied and experimental results are used to evaluate its performance [10]. More recently, a simulated annealing algorithm was used to select Bezier curve parameters for avoiding singularities while solving the path planning problem for a free-floating space manipulator [11].

Singularity-free path planning for closed chains such as the Stewart platform has also been studied; an algorithm that uses the concept of via points connects the points through segment-wise paths and attempts to plan a singularity-free path. The limitations of the algorithm are its lack of confidence in detecting the non-existence of a singularity-free path and its sensitivity to intersections of singularity hypersurfaces [12]. Kinematic redundancy algorithms are also used to find feasible configurations for closed chain manipulators based on an optimization scheme. The cost function in the optimization is designed to avoid the most problematic singularity configurations, where the end-effector can be locally moved even though all actuated joints are locked [13]. Another method of planning singularity-free paths for closed-chain manipulators is done by defining a smooth manifold that maintains a one-to-one correspondence with the singularity-free C-space of the manipulator, then, by using a higher dimensional continuation technique the manifold is explored from one configuration, until the second configuration is found [14]. Singularity avoidance problems for wheeled robots have also been studied and addressed. [15, 16].

2.1.3 Neural Networks and Neuro-Dynamic Systems

An artificial neural network can be defined as a group of interconnected artificial neurons whose connections define the network topology. An artificial neuron is a mathematical model based on the biological processes of information processing, specifically the nervous system's basic unit, the neuron. The basic model of a neuron is shown in Figure 2.3, the output of the neuron $y \in \mathrm{I\!R}$ is computed by

$$y = \sigma\left(\sum_{i=1}^{n} w_i e_i + b\right) \tag{2.12}$$

where $e_i \in \mathrm{I\!R}$ is the ith input to the neuron, $w_i \in \mathrm{I\!R}$ is the synaptic connection strength (weight) of the ith input, $b \in \mathrm{I\!R}$ is the neuron bias, and $\{\sigma(.) : \mathrm{I\!R} \to \mathrm{I\!R}\}$ is the neuron activation function and can be selected to be a linear function (multiplies the sum by one), a hyperbolic tangent function, or a sigmoid [17]. The model can be written in compact form by

$$y = \sigma\left(\mathbf{W}^T \bar{e}\right) \tag{2.13}$$

with $\bar{e} \in \mathrm{I\!R}^{(n+1)\times 1} = [1 \quad e_1 \quad e_2 \, ... \, e_n]^T$, and $\mathbf{W} \in \mathrm{I\!R}^{(n+1)\times 1} = [b \quad w_1 \quad w_2 \, ... \, w_n]$.

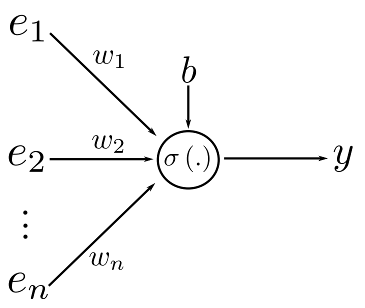

Figure 2.3. Graphical depiction of the artificial neuron model.

The architecture of the connections between the neurons determines the type of neural network. The most famous and widely used type of neural networks is the multi-layer feedforward neural network (FFNN), otherwise known as the multi-layer perceptron (MLP). A FFNN is shown in Figure 2.4, having three layers.

Another example is the recurrent neural network (RNN) which allows for feedback connections between different layers of neurons. This is unlike the FFNN which allows information to flow in one direction through the network. The ability of an

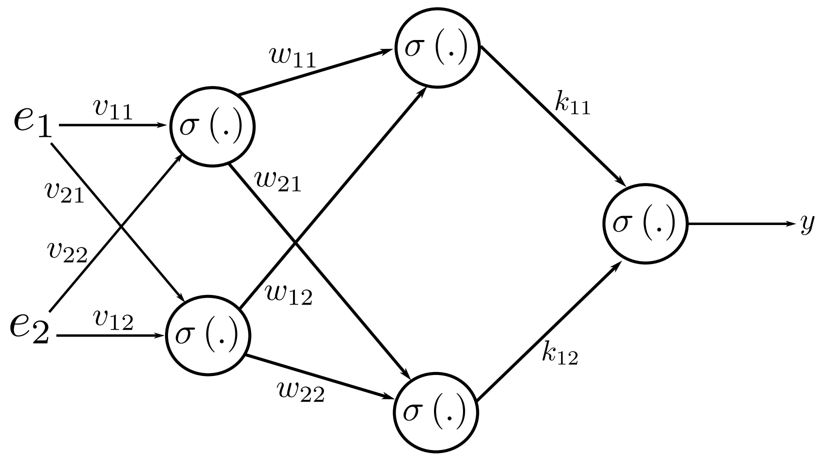

Figure 2.4. Graphical depiction of a three-layer feedforward neural network.

RNN to have connections in any direction throughout the network allows for the introduction of memory, or delay, into the network. Therefore, the RNN can be regarded as a dynamical system. For this reason such networks can be utilized for studying time-varying problems such as dynamical systems analysis and control as well as trajectory generation. The structure of a fully-recurrent neural network is shown in Fig. 2.5.

The dynamics for the i^{th} neuron of such a network can be expressed mathematically using the following differential equation

$$\frac{1}{\tau_i}\dot{x}_i(t) = -x_i(t) + \sum_{j=1}^{n} w_{ij} f_j(x_j(t)) + \sum_{j=1}^{p} \tilde{w}_{ij} e_j(t) \tag{2.14}$$

where $x_i(t) \in \mathbb{R}$ is the state of the ith neuron, $\tau_i \in \mathbb{R}$ is the relaxation time otherwise known as the time constant, $w_{ij} \in \mathbb{R}$ is the connection weight between the jth and ith neuron, $\{f_j(.) : \mathbb{R} \to \mathbb{R}\}$ is a nonlinear activation function, $\tilde{w}_{ij} \in \mathbb{R}$ are weights that scale the inputs of the neuron $e_j(t) \in \mathbb{R}$, $i \in [1, n]$, and n is the number of neurons in the network.

Artificial neural networks have been the topic of research for the past couple of decades. Recently there has been a resurgence in this field that can be attributed

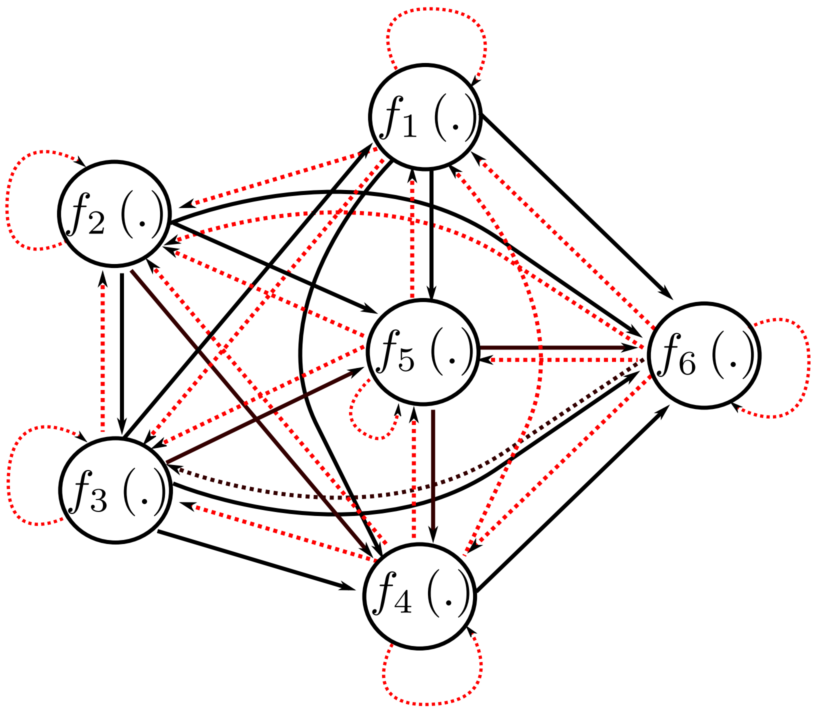

Figure 2.5. Schematic depicting a recurrent neural network structure.

to increase in computational power available. In robotics there are many examples where neural networks have been used; FFNN have been used in a variety of robotic applications, one such application is the localization of a target for a grasping task while solving the required inverse kinematics associated with the task [18]. In other applications the IK solution for a three-link planar manipulator using a FFNN have shown to be fast with acceptable accuracy [19]. Other methods invoking FFNN for solving the IK problem have utilized radial basis function (RBF) neural networks and trained using quadratic programming [20]. Another approach uses parallel RBF neural networks for solving the IK problem, by training the networks using the Levenberg-Marquardt backpropagation algorithm [21]. Examples for using FFNN to solve the IK problem and track different trajectories can be found in [22] and [23]. Other solutions to the IK problem for robotic manipulator involve forming a neural network committee, or a group of neural networks that work together in solving the IK problem [24].

Recurrent neural networks (RNN) have also been used in solving the IK problem, one example is given in [25] where three Elman nets are used in parallel and trained using simulated annealing. Another application of RNN is solving the IK problem for redundant manipulators while avoiding obstacles and singularities; the RNN is used to solve a constrained convex quadratic programming problem in real time to determine the singular configurations of a manipulator, the joint velocities from IK, and the manipulator critical points [26].

Recurrent neural networks have also been used in robotic control applications. They are used for designing a control system for uncertain, nonholonomic dynamical systems, where they learn the system's dynamics and allow a baseline controller to deal with model uncertainties [27]. The echo state neural network (ESNN) has been applied in experiment to controlling a robot manipulator platform [28]. By combining a RNN with nonlinear control methods such as backstepping, identification and control of robotic manipulators is possible [29]. Other types of neural networks have been used for robotic control such as spiking neural networks [30] and the Kalman-neural-network [31].

The Zhang Neural Network (ZNN), which is a special kind of recurrent neural network otherwise referred to as a neuro-dynamic system, was introduced in 2001. It has been used to solve a variety of time-varying problems such as the Sylvester equation with time-varying coefficients [32, 33], time-varying matrix inversion [34], time-varying equation solving [35], time-varying convex quadratic programs [36], time-varying inverse kinematics problem [37], and an online solution to the Moore-Penrose inverse of time-varying full rank matrices problem [38]. Traditional neural networks use a scalar-valued error function, then utilize the gradient of that function to "train" the network and find a solution to a given problem. The ZNN utilizes a matrix valued error function to find the solution to time-varying matrix problems. The matrix valued error function allows for deriving a set of nonlinear dynamical equations for the network, these equations in turn allow for solving the time-varying problem. This set of dynamical equations is known as the neuro-dynamics.

Based on the information available in the recent literature as shown above, neural networks in general have been used to solve a variety of problems in robotic planning and control. However, it is noticeable that there is no one complete system that solves the problem of planning and execution of a specific trajectory for a robotic manipulator. Also, there is not much focus on continuous-time solutions to these problems (using recurrent networks), most of the solutions are of static nature (feedforward networks), which means that they can be applied at specific time instants. Given the time-varying nature of a robotic system these static solutions can lag behind in time, therefore, a continuous-in-time solution that can be implemented online would provide a more in sync solution.

2.2 Research Motivation

Recent events in natural and man-made disasters have highlighted the limitation in man's ability to confine and mitigate damage in such scenarios. The current robotics technology provides a helping hand in disastrous events; however, it has limited capabilities when it comes to navigating environments designed for humans. Therefore, there is an urgent need for robotic technology that can function in all environments and serve as a substitute to humans in disaster scenarios.

Such environments are typically unstructured and uncertain, unlike the laboratories where many robots are designed, built, and testes. This brings up the need for robotic systems that are more immune to such uncertainties and clutter. Typical robot planning and control solutions are based on mathematical models that suffer from multiple problems such as uncertainty and singular solutions. Therefore, the motivation behind the research presented in this document is to develop a system that provides a more robust solution to the robotics planning and control problem that contributes to the state-of-the-art solutions available.

2.3 Research Applications

2.3.1 Robotic manipulators

Robotic manipulators like the one detailed in section 2.1.1 are used in applications of manufacturing, surgery, bomb disposal, handling of radioactive or biohazardous material, and many others. One example is the CanadaArm2 on the international space station as shown in Fig. 2.6. In all applications, a manipulator receives a command for a desired motion and is expected to execute it. This involves solving the path planning and control problems mentioned earlier in this chapter. The work developed in this document solves these problems using novel neuro-dynamic methods that can be applied to any robotic manipulator.

2.3.2 Humanoid robots

The topic of humanoid bipedal robots has been the subject of research since the 1960s [39]. The most studied area on the subject is biped locomotion, or walking. One of the more famous and popular methods for stable walking of humanoids is the Zero-Moment-Point (ZMP) method, or criterion. The ZMP was introduced and implemented by Miomir Vukobratovic in the 1960s and 1970s [40]. The ZMP criterion assumes the presence of a point inside the convex hull of contact points on the ground where the tipping moment is zero, hence the name, zero-moment point. This alludes to the fact that the resultant forces and moments at that point will not contribute

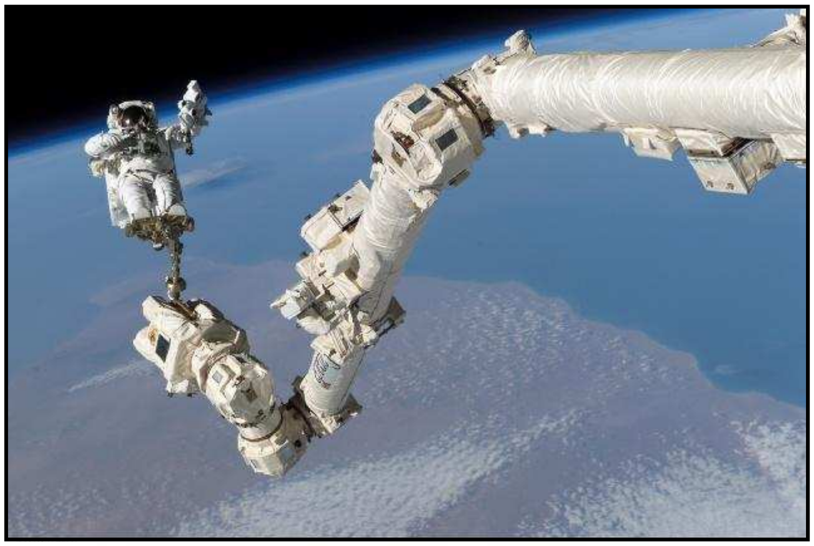

Figure 2.6. Astronaut Stephen K. Robinson anchored to a foot restraint on the International Space Station's Canadarm2.

to the destabilization of the humanoid robot. The convex hull of contact points with the ground is usually referred to as the support polygon. For the robot to be stable the ZMP must remain within the support polygon. Many publications are available on the ZMP; one self-contained document can be found in reference [41].

A typical method, found in the literature, for stable humanoid walking pattern generation is that of designing the feet and Center-of-Mass (COM) trajectories in such a way that the ZMP remains inside the support polygon. This ensures the robot is stable at all times. The motion of a humanoid's foot is usually broken down to two phases: swing foot and support foot. During the swing foot phase, a foot is swung forward along a prescribed trajectory while the other foot is in support phase, i.e. on the ground providing support. And by repeating this two-phase cycle the robot walks. In each swing foot state the humanoid robot can be treated as an N-DOF manipulator, and the path planning and control methods discussed earlier can be applied to control its motion.

When generating a COM (ZMP) or hip trajectory that ensures stability with these footsteps the term walking pattern or gait is used to describe such trajectories [42, 43]. A hip motion trajectory can be used to maintain stability. The inverted pendulum model has been utilized to approximate the robot dynamics for pattern generation with preview control or model predictive control [44]. It has been shown that the inverted pendulum model yield acceptable results for humanoid robot stable walking in the sense of ZMP.

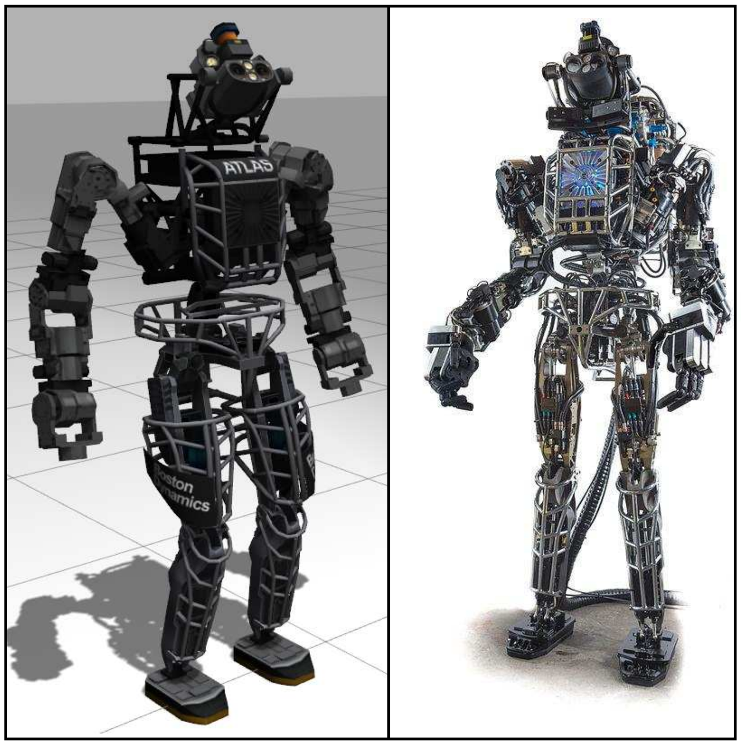

Figure 2.7. The Boston Dynamics Atlas humanoid robot - (left) simulation, (right) actual.

Much of the prior work on humanoid locomotion been geared towards stable gait generation. Despite using sophisticated control methods, the main focus of that work is to generate stable walking trajectories in the sense of ZMP. Executing those trajectories on a system of complex, uncertain dynamical nature as a humanoid robot has not been given as much attention. Humanoid robots are highly complex electro-mechanical systems, such as the Boston Dynamics Atlas humanoid robot shown in Fig. 2.7. Since humanoid robots are expected to one day operate in highly uncertain environments, be in direct contact with human beings, or execute sensitive tasks, one directly foresees a multitude of obstacles that would arise when designing a system to execute trajectories designed for stable walking. The neuro-dynamic system developed in this document will serve as a solution to the path planning and control problems relating to the humanoid walking problem.

CHAPTER 3

Problem Formulation

The problems that are solved throughout this document are outlined and mathematically detailed in this chapter. These problems are sub-problems of the whole robotic planning and control problem that is addressed in this dissertation. Figure 3.1 shows a block diagram like description of the robotic planning and control problem. As outlined by the figure, the robot forward kinematics and dynamics allow for calculating the robot joint positions, joint velocities, end-effector position, and end-effector velocity. To command the robot to execute a specific task the following three problems must be solved. These problems are outlined in red, green, and blue respectively in Fig. 3.1.

1. The operational space trajectory generation problem: requires the generation of a trajectory related to the task to be executed.
2. The inverse kinematics (IK) problem: using the trajectory from problem (1), a trajectory in joint space needs to be calculated based on the kinematics of the robot.
3. The control problem: a control system must be designed to track the joint trajectory generated in problem (2).

It is noted that to calculate the solution for problem (2) the inverse of the Jacobian matrix needs to be computed. As detailed in chapter 2, the Jacobian can suffer from singularities that render it numerically ill-conditioned for inversion. For this reason a singularity-free or singularity robust joint trajectory must be calculated. In the work presented here this is done in two different ways: the first involves designing a joint trajectory that is singularity free. This requires knowledge of the singular configurations a priori. The second on the other hand attempts to calculate the Jacobian inverse in a singularity robust manner.

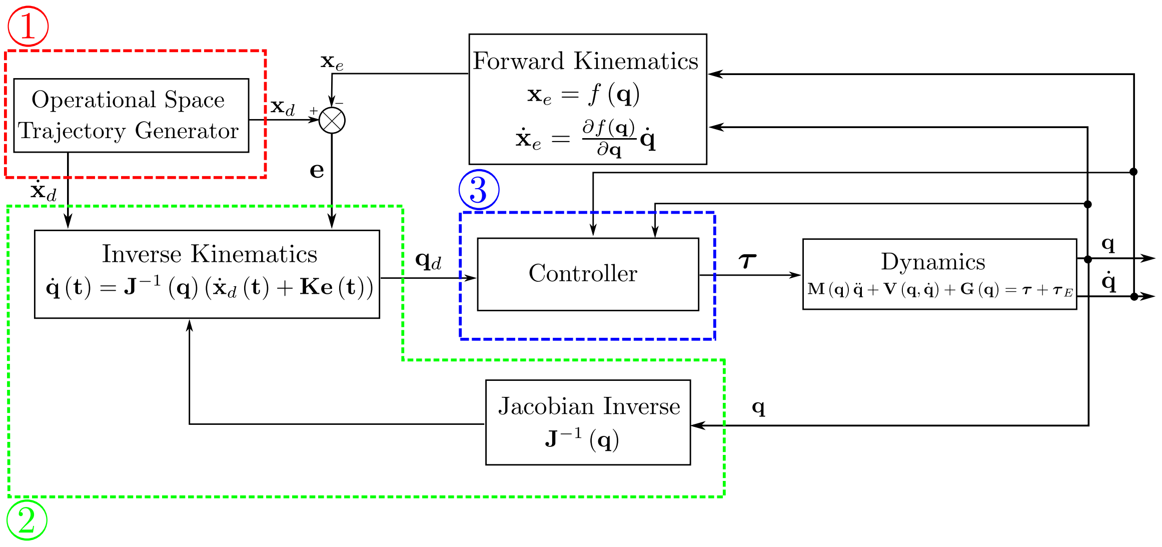
1
Operational Space Trajectory Generator
$\mathbf{x}_d$
$\mathbf{x}_e$
+
−
$\mathbf{e}$
$\dot{\mathbf{x}}_d$
Forward Kinematics
$\mathbf{x}_e = f(\mathbf{q})$
$\dot{\mathbf{x}}_e = \frac{\partial f(\mathbf{q})}{\partial \mathbf{q}} \dot{\mathbf{q}}$
3
Inverse Kinematics
$\dot{\mathbf{q}}(\mathbf{t}) = \mathbf{J}^{-1}(\mathbf{q})(\dot{\mathbf{x}}_d(\mathbf{t}) + \mathbf{K}\mathbf{e}(\mathbf{t}))$
$\mathbf{q}_d$
Controller
$\boldsymbol{\tau}$
Dynamics
$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{V}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau} + \boldsymbol{\tau}_E$
$\mathbf{q}$
$\dot{\mathbf{q}}$
Jacobian Inverse
$\mathbf{J}^{-1}(\mathbf{q})$
$\mathbf{q}$
2

Figure 3.1. Block diagram of planning and control problem - three major subproblems highlighted.

The following sections of this chapter provide the mathematical descriptions of the problems mentioned above.

3.1 Operational Space Trajectory Generation Problem

Given a system governed by the kinematic model

$$\mathbf{x}_e = f(\mathbf{q}) \tag{3.1}$$

where $\mathbf{x}_e \in \mathbb{R}^{\mathrm{M}\times 1}$, $\mathbf{q} \in \mathbb{R}^{\mathrm{N}\times 1}$, with $N \neq M$, and where $\{f(.) : \mathbb{R}^{\mathrm{N}\times 1} \rightarrow \mathbb{R}^{\mathrm{M}\times 1}\}$. Also

$$\dot{\mathbf{x}}_e = \mathbf{J}(\mathbf{q})\,\dot{\mathbf{q}} \tag{3.2}$$

where $\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{\mathrm{M}\times\mathrm{N}} \equiv \frac{\partial f(\mathbf{q})}{\partial \mathbf{q}}$. Also given that the system is governed by the following dynamic model

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{V}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau} + \boldsymbol{\tau}_E \tag{3.3}$$

where $\mathbf{M}(\mathbf{q}) \in \mathbb{R}^{\mathrm{N}\times\mathrm{N}}$ is symmetric mass matrix, $\mathbf{V}(\mathbf{q},\dot{\mathbf{q}}) \in \mathbb{R}^{\mathrm{N}\times 1}$ is the Coriolis/centripetal vector, $\mathbf{G}(\mathbf{q}) \in \mathbb{R}^{\mathrm{N}\times 1}$ is the gravity vector, $\boldsymbol{\tau} \in \mathbb{R}^{\mathrm{N}\times 1}$ is the input torque vector, and $\boldsymbol{\tau}_E \in \mathbb{R}^{\mathrm{N}\times 1}$ represents the unmodeled disturbance torques. Find a trajectory $\mathbf{x}_e^d(t)$.

3.2 Singularity-Free/Robust Joint Trajectory Calculation Problem

For a system governed by the kinematic model in Eq. 3.1, the differential kinematics in Eq. 3.2, and the dynamic model in 3.3; given a trajectory $\mathbf{x}_e^d(t) \in C^2$; i.e $\mathbf{x}_e^d(t)$ is bounded and twice differentiable with respect to time and $\ddot{\mathbf{x}}_e^d(t)$ is also bounded; find $\mathbf{q}_d(t)$ subject to

$$\mathbf{det}\left(\mathbf{J}(\mathbf{q}_d(t))\,\mathbf{J}^T(\mathbf{q}_d(t))\right) \neq 0 \tag{3.4}$$

3.3 Trajectory Tracking Controller Design

Given a system governed by nonlinear dynamics of the form in 3.3, define the tracking error as $\mathbf{e}(t) = \mathbf{q}_d(t) - \mathbf{q}(t)$. A control law, $\boldsymbol{\tau} = \mathbf{g}(\mathbf{q},\dot{\mathbf{q}},\mathbf{q}_d,\dot{\mathbf{q}}_d,\ddot{\mathbf{q}}_d)$, will be designed such that $\|\mathbf{e}(t)\| \leq \varepsilon > 0$ as $t \to \infty$.

CHAPTER 4

Operational Space Trajectory Generation

Many different methods of path planning and trajectory generation have been developed over the past decade or more. This is due to the fact that solving these two problems is essential for many applications. Recently published work on topics related to path planning and trajectory generation show their application variety, one of which is finding a safe path from one point to the other for a six-wheeled rover over rough terrain [45]. Another deals with creating feasible paths to generate the motion of snake-like robots using potential functions [46]. Other applications involve generating real-time trajectories for quadrotors [47], generating trajectories for an automated excavator [48], and generating energy-optimal trajectories for servomotor systems [49].

This wide variety of applications has given birth to different methods for solving the trajectory generation and path planing problems. Such recent methods involve the use of genetic algorithms for path planning [50]. Optimal control theory is also used to generate a minimum time trajectories [51]. Another approach for optimal trajectories uses the double generating-functions method to solve the trajectory generation problem as an optimization problem [52]. Other recent works utilize Bézier curves to generate trajectories [53] and C-splines to connect via-points and formulate a smooth position trajectory [54].

During the last decade there has been a resurgence in the use of artificial neural networks as tools for problem solving. This can be generally attributed to the increase in available computational power. Different kinds of neural networks have been used for path planning and trajectory generation. In the application of legged locomotion, neural oscillators are used to form central pattern generators (CPG) to create walking patterns [55]. To plan a desired trajectory for a flexible robotic manipulator, multi-layer feedforward neural networks are utilized to generate smooth trajectories [56]. This allows for suppressing unwanted vibrations in the manipulator. Artificial neural networks trained using the imperialist competitive algorithm are used to optimally plan paths for an unmanned combat aerial vehicle [57], while guided auto wave pulse coupled neural networks are used for mobile robot path planning [58]. The latter approach is a biologically inspired one that identifies every neuron in the system as an obstacle or free space in the configuration space of the robot. By using variable thresholds for each neuron the proposed method generates optimal paths at improved times compared to the $\mathbf{A}^*$ search algorithm.

Using a discrete time recurrent neural network model combined with obstacle detection and avoidance algorithms, the work in [59] shows the ability to generate real-time trajectories in a dynamic environment for an intelligent vehicle. A similar approach can be seen in the work published by [60], where a continuous-time model of a recurrent neural network combined with the Grossberg obstacle avoidance method is used for trajectory planning for a mobile robot. This approach can also be seen in the work by [61]. In more recent work, cellular neural networks which are a type of recurrent neural networks, have been used to solve a path planning problem for a mobile robot using images from a camera [62]. Also in recent work, pulse couple neural networks are used to plan the shortest path between two points with the presence of obstacles [63].

The problem of trajectory generation is the focus of this chapter. The work presented here deals with this problem in an abstract manner, not related to any specific application. This allows for a generic solution to the trajectory generation problem. In this work a dynamic neural network (DNN), which is a combination of a recurrent neural network (RNN) and feedforward neural networks (FFNN), is used to generate a desired trajectory. The principle behind the solution presented here is based on the fact that the trajectory generation problem can be broken down into two separate steps; the first maps time to an intermediate trajectory, and the second maps that intermediate trajectory to a desired one [64].

4.1 Trajectory Generation Using a Dynamic Neural Network

To solve the operational space trajectory generation problem stated in chapter 3, the problem is broken down into two steps; the first involves generating an intermediate trajectory, $\mathbf{o}(t)$, which is a function of time. This trajectory is designed to have desired temporal characteristics such as a specific frequency. The second step involves shaping the trajectory from the first part to have the desired spatial shape such as a circle, an ellipse, or a figure eight. These two consecutive steps can be accomplished using a DNN, a combination of a RNN and a FFNN. The RNN generates a time varying trajectory through a mapping, $\{\mathbf{h}(\cdot)\ :\ [0,\ T] \rightarrow \mathrm{I\!R}^{\mathrm{k}}\}$, of time t into an intermediate trajectory $\mathbf{o}(t) \in \mathrm{I\!R}^{\mathrm{k}}$, where k is a positive integer. This intermediate trajectory drives the FFNN to generate the desired trajectory $\mathbf{y}(t)$ through a mapping, $\{\mathbf{g}_l(\cdot)\ :\ \mathrm{I\!R}^{\mathrm{k}} \rightarrow \mathrm{I\!R}^{\mathrm{k}}\}$, of $\mathbf{o}(t)$ such that $\mathbf{y}(t) = \mathbf{g}_l(\mathbf{o}(t))$ (see [64]). Figure 4.1 shows a block diagram of such a DNN.

4.1.1 Recurrent Neural Network Design

The RNN is designed to generate a two-dimensional trajectory that does not cross over itself unless a new cycle begins. An example of such a trajectory would be a circle with a period T. This way the RNN is able to drive the FFNN in a continuous manner. To accomplish this, the dynamics of the RNN have to be shaped in such a

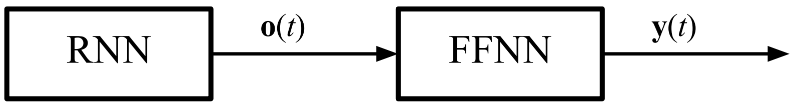

Figure 4.1. Block diagram of dynamic neural network structure for trajectory generation.

way that they globally converge to a stable limit cycle, i.e. a stable trajectory. This is be done by utilizing relay feedback systems [65].

To approximate a limit cycle oscillation with a frequency of ω_l in two dimensions, we introduce two states; the first is $\mathbf{o}_1(t) \approx A\sin(\omega_l t)$ and the second is its derivative $\mathbf{o}_2(t) \approx A\cos(\omega_l t)$, a relay feedback system as the one shown in Fig. 4.2 can be used with the following transfer function [64]:

$$G(s) = \frac{K}{s(s+1)(s+\omega_l^2)} \tag{4.1}$$

with $K = A\pi\omega_l^2(1+\omega_l^2)/4d$, where $d > 0$ is the control signal used to drive the system $G(s)$ such that

$$u(t) = \begin{cases} d & \text{if } e(t) > 0 \\ -d & \text{if } e(t) < 0 \end{cases} \tag{4.2}$$

This relay feedback system can be simulated using an RNN as was shown in the work by [64]. By synthesizing a three neuron RNN, $n = 3$, and one input, $p = 1$, the state-space realization of $G(s)$ can be achieved using Eq. 2.14. The parameters used for the RNN are shown in table 4.1. The input to the RNN is designed as $e_1(t) = f^*(w_{RN}x_3(t))$ where $f^*(z) = 2/(1+e^{-rz}) - 1$ with $r \gg 2$ and $w_{RN} = -1$. With this design of the RNN the state of the third neuron is the output, given as

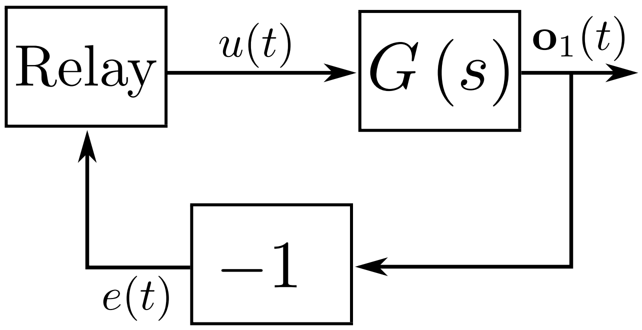

Figure 4.2. Relay feedback control system.

$\mathbf{o}_1(t) = x_3(t) \approx A\cos(\omega_l t)$. The RNN architecture for this system is shown in Fig. 4.3.

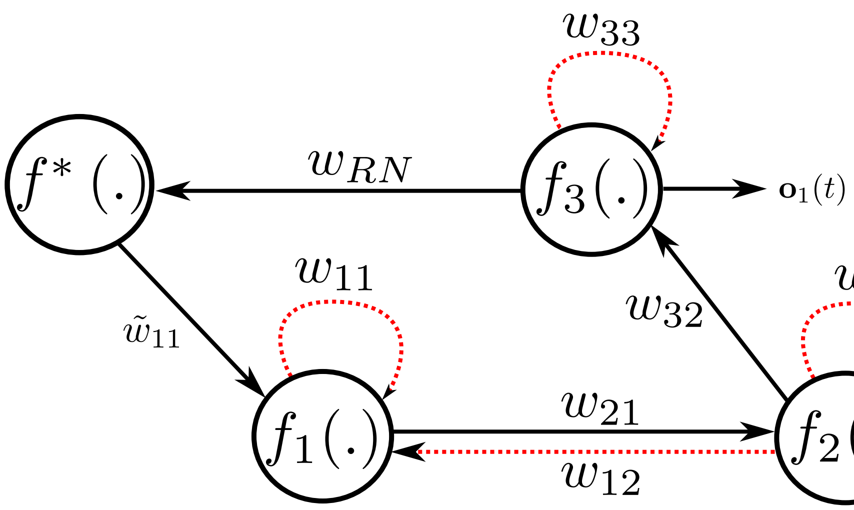

Figure 4.3. Relay feedback system realization using a RNN.

To generate a two-dimensional limit cycle using the system in Fig. 4.3 the state of the second neuron can be used. From the synthesized RNN, $x_2(t) = \dot{x}_3(t)$, therefore, $\mathbf{o}_2(t) = x_2(t)$ and $\mathbf{o}_1(t) = x_3(t)$ give a periodic, two-dimensional trajectory that does not cross over itself unless a new cycle begins. This limit cycle is the output of the RNN in the DNN shown in Fig. 4.1.

Table 4.1. RNN Parameters for relay-feedback state-space realization

Parameter	Value	Parameter	Value
w_{11}	$-\omega_l^2 - 1$	w_{31}	0
w_{12}	$-\omega_l^2$	w_{32}	1
w_{13}	0	w_{33}	0
w_{21}	1	$\tilde{w}_{11}$	K
w_{22}	0	$\tilde{w}_{21}$	0
w_{23}	0	$\tilde{w}_{31}$	0

4.1.2 Feedforward Neural Network Design

In the DNN system, the outputs of the RNN are passed to the FFNN to shape their spatial variation. This is done by training the FFNN using the outputs of the RNN as inputs and desired trajectories as targets. By using any training method of choice, such as the backpropagation algorithm [66], and the training set $\{[\mathbf{o}_1(t)\ \mathbf{o}_2(t)]\,,\ \ [\mathbf{y}_{1d}(t)\ \mathbf{y}_{2d}(t)]\}$ the weights of the FFNN connections can be calculated.

4.2 Dynamic Neural Networks with Intelligent Feedback

In the above mentioned DNN structure, the RNN has no knowledge of the results generated by the FFNN. The RNN only supplies the intermediate trajectory, $\mathbf{o}(t)$, to the FFNN so that it can generate the desired trajectory, $\mathbf{y}(t)$. Adaptive intelligence is introduced using a feedback element. By using another FFNN in feedback, an input signal, $\mathbf{e}\left(t\right)$, can be supplied to the RNN using a mapping, $\{\mathbf{g}_e(\cdot)\ :\ \mathbb{R}^{\mathrm{m}}\ \rightarrow \mathbb{R}^{\mathrm{n}}\ \}$, of the output of the first FFNN $\mathbf{y}(t)$ such that $\mathbf{e}(t) = \mathbf{g}_e\left(\mathbf{y}(t)\right)$. This closes the loop between the desired trajectory being generated and the dynamics of the RNN, making the trajectory generation system adaptive. Figure 4.4 shows a block diagram of a DNN with intelligent feedback.

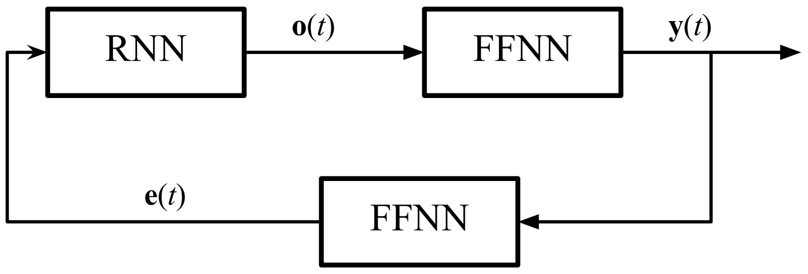

Figure 4.4. Adaptive intelligence system for trajectory generation.

4.2.1 Design of Intelligent Feedback Element

The feedback FFNN is used to generate input to the RNN based on the outputs of the first FFNN which has been trained to generate specific outputs, $[\mathbf{y}_1(t)\ \mathbf{y}_2(t)]$, based on the trajectory generated by the RNN $[\mathbf{o}_1(t)\ \mathbf{o}_2(t)]$. The feedback FFNN is designed by training the network to generate specific input signals that can drive the RNN based on the outputs of the first FFNN. This is done by using the training set $\{[\mathbf{y}_1(t)\ \mathbf{y}_2(t)]\,,\ [\mathbf{e}_d(t)]\}$.

4.3 Trajectory Generation Examples

4.3.1 Circular Trajectory

A DNN with intelligent feedback is used to generate a circular trajectory with a diameter of 0.5 m and a frequency of π rad/s. To ensure that the generated trajectory has the desired frequency an RNN is designed to generate the intermediate trajectory $\mathbf{o}_1(t)$ with a frequency of π rad/s. The first FFNN is trained with the training set $\{[\mathbf{o}_1(t)\ \mathbf{o}_2(t)]\,,\ [\mathbf{x}_d(t)\ \mathbf{y}_d(t)]\}$. This FFNN is a two-layer one with 3 hidden layer neurons using a sigmoid activation function. The feedback FFNN is also a two-layer one with 10 neurons in the hidden layer that use a sigmoid activation activation function. The feedback FFNN is trained using the set $\{[\mathbf{x}(t)\ \mathbf{y}(t)],\ [\mathbf{e}_d(t)]\}$, where $[\mathbf{x}(t)\ \mathbf{y}(t)]$ are the outputs of the first FFNN and $e_d(t) = f^*(w_{RN}x_3(t))$.

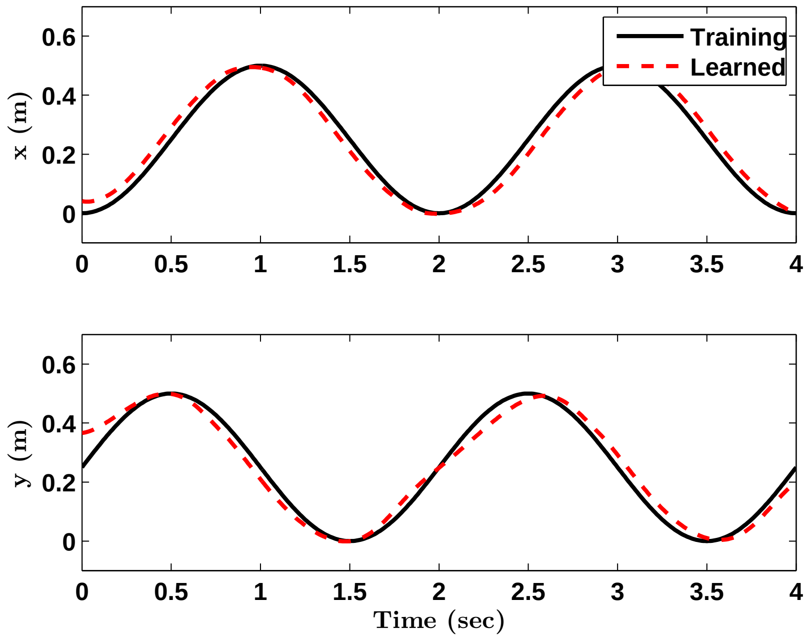

Figure 4.5. Desired trajectory (training) and DNN generated trajectory (learned) as a function of time - circle.

After training the system, a simulation is conducted to verify the results. The RNN dynamics are simulated numerically and used as an input to the FFNN. The desired trajectories and the outputs of the DNN, are shown in Figs. 4.5 and 4.6. From Fig. 4.5 one can clearly see the periodic and sinusoidal nature of the DNN generated (learned) trajectories $\mathbf{x}(t)$ and $\mathbf{y}(t)$ with a period of approximately 2 second, and therefore a frequency of approximately π rad/s. The figure also shows that the DNN generated trajectories are in close agreement with the desired (training) trajectories $\mathbf{x}_d(t)$ and $\mathbf{y}_d(t)$.The circular trajectory is shown with the two-dimensional plot in Fig. 4.6. This figure shows a circle of diameter 0.5 m to serve as the desired trajectory along with the trajectory generated by the DNN. Clearly the generated trajectory fulfills the spatial requirements of a circle 0.5 m in diameter. The ($\times$) in the figure indicates the starting point of trajectory generation, which is dictated by the initial conditions of the RNN. These results indicate the capability of the DNN with feedback to generate circular trajectories.

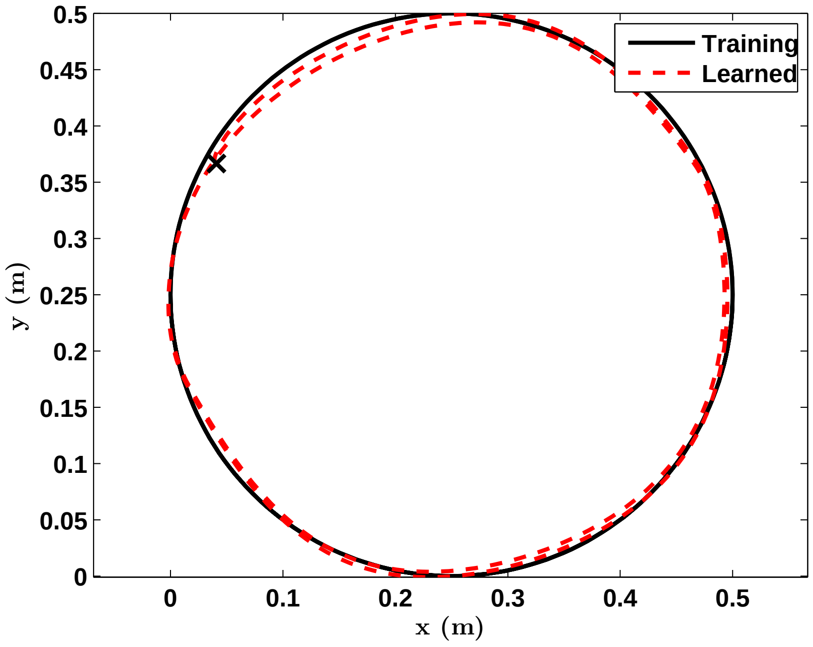

Figure 4.6. Desired trajectory (training) and DNN generated trajectory (learned) - circle.

The performace of the feedback FFNN is shown in Fig. 4.7 where the input to the RNN is shown. This FFNN is trained to generate an input to the RNN to control its states that are used to generate the trajectories in Figs. 4.5 and 4.6. It is clear from the result in Fig. 4.7 that the feedback FFNN learnes how to generate this input based on the trajectories generated by the DNN, and that this learned input matches the training data used. The results shown here verify the ability of a DNN with intelligent feedback to generate a circular trajectory.

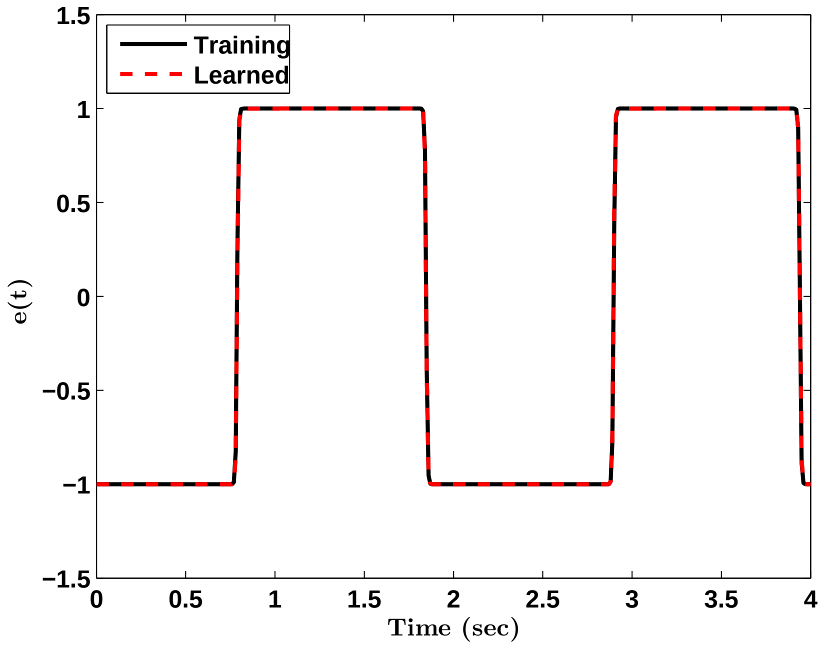

Figure 4.7. Desired DNN input (training) and feedback FFNN generated input (learned) - circle.

4.3.2 Figure Eight Trajectory

A more complicated trajectory compared to the circular one shown in the previous example is the figure eight trajectory. This problem has become a benchmark against which solutions for the trajectory generation problem are tested. This is due to the many-to-one mapping that occurs when the trajectory overlaps. The desired trajectory to be generated is $\mathbf{x}_d(t) = 0.5\sin(\omega_1 t + \delta)$ and $\mathbf{y}_d(t) = -0.5\sin(\omega_2 t)$, where $\omega_1 = 0.5$ rad/s, $\omega_2 = 1$ rad/s, and $\delta = \frac{\pi}{2}$. An RNN is designed to generate the intermediate trajectory $\mathbf{o}_1(t)$ with a frequency of 0.8 rad/s. Similar to the previous example, the first FFNN is trained with the training set $\{[\mathbf{o}_1(t)\ \mathbf{o}_2(t)]\,,\ [\mathbf{x}_d(t)\ \mathbf{y}_d(t)]\}$. This FFNN is a two-layer one with 15 hidden layer neurons using a sigmoid activation function. The feedback FFNN is a three-layer one with 15 neurons that use a sigmoid activation function in the first hidden layer, and 8 neurons that use a log-sigmoid activation function in the second hidden layer. The feedback FFNN is trained using the set $\{[\mathbf{x}(t)\ \mathbf{y}(t)]\,,\ [\mathbf{e}_d(t)]\}$, where $[\mathbf{x}(t)\ \mathbf{y}(t)]$ are the outputs of the first FFNN and $e_d\,(t) = f^*\,(w_{RN}x_3\,(t))$.

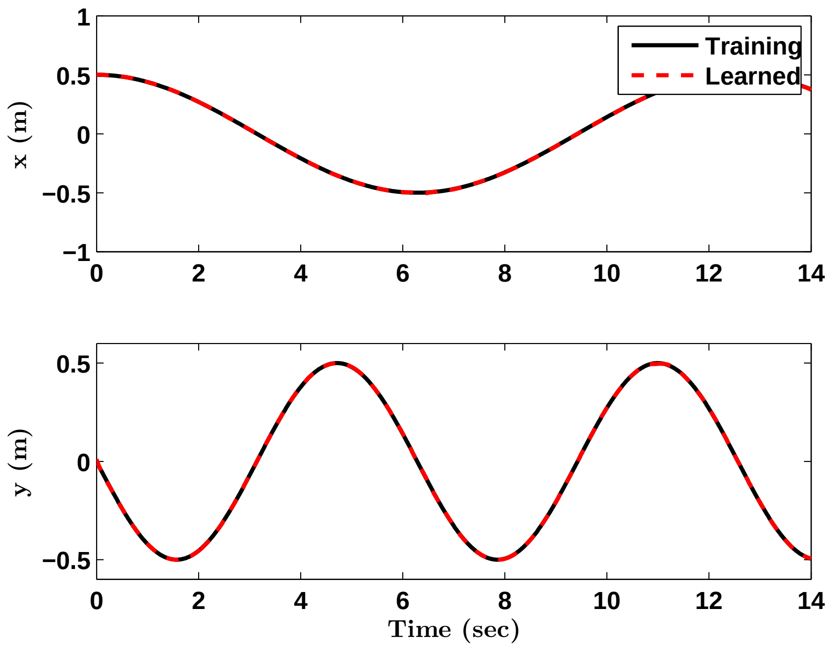

Figure 4.8. Desired trajectory (training) and DNN generated trajectory (learned) as a function of time - figure eight.

Using a numerical simulation similar to the one used for generating the circular trajectory earlier, the outputs of the DNN for generating a figure eight trajectory are shown in Figs. 4.8 and 4.9. From Fig. 4.8 it is clear that the generated (learned) trajectory $\mathbf{x}(t)$ has a frequency which is half that of $\mathbf{y}(t)$ this is expected since the

FFNN was trained to generate such trajectories to produce a figure eight. The desired (training) trajectories are also shown in the figure, here we can see that the generated trajectories are almost identical to the desired ones. The figure eight trajectory generated by the DNN along with a desired trajectory are shown in a two-dimensional figure in Fig. 4.9. This figure shows that a DNN with intelligent feedback as the one designed here is capable of producing such a complex trajectory with high accuracy. As can be seen in the figure the learned trajectory is almost identical to the training one. The $\times$ in the figure indicates the starting point of trajectory generation.

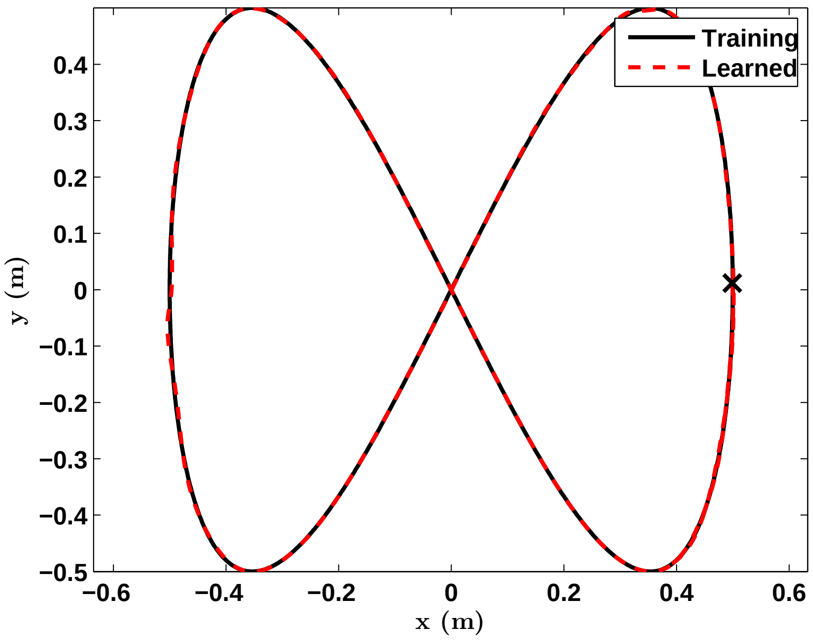

Figure 4.9. Desired trajectory (training) and DNN generated trajectory (learned) - figure eight.

The performance of the feedback FFNN for generating a figure eight trajectory is shown in Fig. 4.10. This FFNN is trained to generate an input to the RNN to control its states that are used to generate the figure eight trajectory in Fig. 4.6. It is clear from the result in Fig. 4.7 that the feedback FFNN learns how to generate this input based on the trajectories generated by the DNN, and that this learned input matches the training data used. The results shown here verify the ability of a DNN with intelligent feedback to generate a complex trajectory such as the figure eight trajectory.

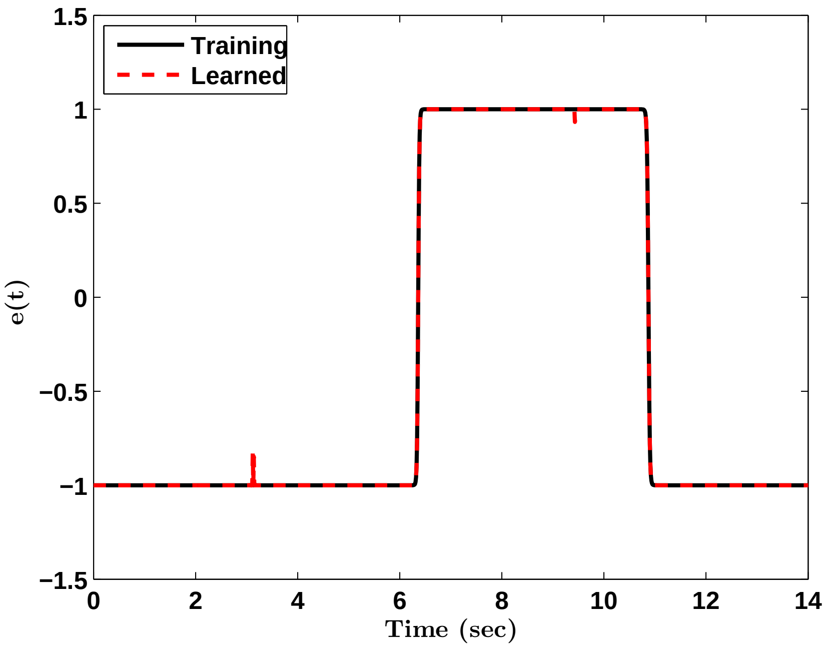

Figure 4.10. Desired DNN input (training) and feedback FFNN generated input (learned) - figure eight.

CHAPTER 5

Singularity-Free Joint Trajectory Generation

In this chapter a method for planning singularity-free paths for N-DOF manipulators is developed. The singular configurations of the manipulator need to be identified, then a path planning algorithm will use that information to plan a path while avoiding these configurations. Also, the implementation aspect of the path planner on robotic manipulators using conventional control methods is established.

The motivation behind the work presented in this chapter stems from the need to calculate the inverse (pseudoinverse) of the Jacobian matrix when solving the IK problem numerically. If the configuration is singular, or near singular, this inversion might not be accurate or even feasible. Therefore, by planning paths that can execute a required task while avoiding singular configurations the Jacobian inverse is always computable. The proposed idea is to utilize Artificial Potential Functions (APF) in joint space rather than Cartesian space while treating singularity configurations as obstacles. The goal is to develop a singularity-free path-planning technique that allows for connecting two points (configurations) in joint space while avoiding singular configurations. This work presents a novel approach to solving the singularity-free path planning problem for a robotic manipulator using conventional methods in path planning.

APFs are functions that have been widely used in path planning with obstacle avoidance in operational space. The concept developed here adopts a similar strategy in joint space. The robotic manipulator kinematic structure is first analyzed, and the singular configurations are identified. By using a special class of curves, the superquadrics, repulsive potentials are created and centered at the singular configuration in joint space. A global potential function is generated by a superposition of multiple repulsive and attractive potentials. The gradient of that function is used to plan a singularity-free path from the current configuration to a final, desired, one.

5.1 Artificial Potential Functions

Artificial potential functions are mathematical representations of potential energies in a system [67]. When a function is formed, it is typically a superposition of multiple potentials that can be attractive or repulsive. In the field of robotics such functions have been used, in combination with controllers, to solve problems in obstacle avoidance [68], dynamic path planning with obstacles in Cartesian space [69], and robotic motion planning [70]. A class of APF that has also been used for obstacle avoidance is the superquadric APF [71]. The main quality of these potential functions is that they prevent the creation of local minima, a problem that APF suffer from when multiple repulsive and attractive potentials are superimposed to create a global artificial potential function.

The repulsive potentials are analogous to hills or peaks, they are centered around locations that typically need to be avoided. A repulsive potential function must have the following characteristics [71]

1. The potential should have spherical symmetry at large distances to avoid the creation of local minimum when this potential is added to an attractive well.
2. The potential of an object to be avoided should have a limited range of influence.
3. The potential and the gradient of the potential must be continuous.

A class of APFs that fits the above mentioned characteristics, and therefore is used in the proposed method, is the superquadric potential function.

5.1.1 Superquadric Artificial Potential Functions

The superquadric potential is a function that has isopotential surfaces shaped like superquadrics. The value of the potential energy at each surface is determined by the potential energy function. Superquadrics, enclosing the origin in an $\mathbb{R}^{\mathrm{N}}$ space, can be mathematically described as follows [71]

$$\left(\frac{q_1}{f_1\left(q_1,...,q_N\right)}\right)^{2n}+\left(\frac{q_2}{f_2\left(q_1,...,q_N\right)}\right)^{2n}...+\left(\frac{q_N}{f_N\left(q_1,...,q_N\right)}\right)^{2n}=1 \tag{5.1}$$

where $q_1, q_2, ..., q_N$ are coordinates in $\mathbb{R}^{\mathrm{N}}$ space, $f_i\left(q_1,...,q_N\right) \in \mathbb{R}$ are scaling functions, and $n \in \mathbb{R}$. The superquadric equation can be further simplified when selecting the scaling functions to be constant, which gives the following representation

$$\left(\frac{q_1}{c_1}\right)^{2n}+\left(\frac{q_2}{c_2}\right)^{2n}+...+\left(\frac{q_N}{c_N}\right)^{2n}=1 \tag{5.2}$$

Equation 5.2 is the equation of an n-ellipse, which can also be expressed as

$$\left(\frac{q_1-q_1^s}{c_1}\right)^{2n}+\left(\frac{q_2-q_2^s}{c_2}\right)^{2n}+...+\left(\frac{q_N-q_N^s}{c_N}\right)^{2n}=1 \tag{5.3}$$

when the point $\mathbf{q}^s \in \mathbb{R}^{\mathrm{N}} = \left[q_1^s \quad q_2^s \; ... \; q_N^s\right]^T$ is enclosed by the n-ellipse.

Since the superquadric potential functions were originally developed for navigating around objects in operational space, two constraints are imposed on the n-ellipse for it to be useful as a potential function: first, the ellipse must touch the corners of the surrounded object and second, the area between the object (assumed to be rectangular in shape) and the ellipse must be minimal. These constraints can be mathematically stated for the i-th coordinate as

$$c_i = \frac{w_i}{2}\left(2^{\frac{1}{2}n}\right) \tag{5.4}$$

where w_i is the width of the hyper-rectangle surrounding the object. In the case of the work presented here, the obstacle to be avoided is actually a point and has no dimensions. However, w_i is used as a parameter to create a buffer zone around the point to be avoided. As w_i increases the buffer zone expands.

To define the isopotential contours a pseudo-distance from each singular point is defined

$$K^{s_j} = \left[\left(\frac{q_1 - q_1^{s_j}}{c_1^{s_j}} \right)^{2n} + \left(\frac{c_2^{s_j}}{c_1^{s_j}} \right)^2 \left(\frac{q_2 - q_2^{s_j}}{c_2^{s_j}} \right)^{2n} + \left(\frac{c_N^{s_j}}{c_1^{s_j}} \right)^2 \left(\frac{q_N - q_N^{s_j}}{c_N^{s_j}} \right)^{2n} \right]^{\frac{1}{2n}} - 1 \quad (5.5)$$

The value of $K^{s_j} \in \mathbb{R}$ is the pseudo-distance from point $\mathbf{q}^{s_j} \in \mathbb{R}^{\mathrm{N}}$ and varies from zero at the point to infinity at a far enough position away from $\mathbf{q}^{s_j}$. The subscript $j = 1, 2, 3, ...$ represents the number of points that are to be enclosed by isopotential contours.

The relationship between K^{s_j} and n is such that as K^{s_j} goes form zero to infinity as n goes from infinity to one. Therefore, we can define n as

$$n = \frac{1}{1 - e^{-\alpha \beta_n K^{s_j}}} \quad (5.6)$$

where α and β_n are adjustable, scalar parameters. It should be noted that this is not the only way to define n, other definitions are possible.

The repulsive potential centered at a point $\mathbf{q}^{s_j} \in \mathbb{R}^{\mathrm{N}}$ can be defined as a function of the pseudo-distance $U_{rep}^{s_j}(K^{s_j})$. It must go to infinity at $K^{s_j} = 0$ and decay exponentially as the distance from the singular point increases. The following definition is used

$$U_{rep}^{s_j} = A \frac{e^{-\alpha K^{s_j}}}{K^{s_j}} \quad (5.7)$$

where $A \in \mathbb{R}$ is an overall scaling factor for the potential.

5.1.2 Attractive Potentials

The attractive potential used in this work is the quadratic attractive well. This potential is generally a bowl shaped energy well which drives the system to its center if the environment is unobstructed. An attractive potential, with a minimum at the origin, can be mathematically expressed as

$$U_{att} = \frac{\kappa}{2}\mathbf{q}^T\mathbf{q} \tag{5.8}$$

where $\mathbf{q} \in \mathbb{R}^{\mathrm{N}}$ and $\kappa \in \mathbb{R}$ is a scaling factor. For the case when the minimum is required to be at a point $\mathbf{q}_f \in \mathbb{R}^{\mathrm{N}}$, the attractive potential can be written as

$$U_{att}^{\mathbf{q}_f} = \frac{\kappa}{2}\left(\mathbf{q} - \mathbf{q}_f\right)^T\left(\mathbf{q} - \mathbf{q}_f\right) \tag{5.9}$$

Now that the definitions for repulsive and attractive potentials are established, they can be superimposed to form a global artificial potential function. For example, for an m number of points to be avoided, m-repulsive potentials and an attractive potential can be superimposed to create the following global potential

$$U_{global} = U_{att}^{\mathbf{q}_f} + U_{rep}^{s_1} + U_{rep}^{s_2} + \ldots + U_{rep}^{s_m} \tag{5.10}$$

5.2 Singularity-Free Path Planning using APF

Analysis of the manipulator structure, more specifically the Jacobian matrix, can reveal the location of singularity points in joint space. Using superquadric potential functions, which were described earlier, repulsive potentials are centered at a singularity point in joint space. A hyper-rectangle in $\mathbb{R}^{\mathrm{N}}$, is sized around the singularity point to act as a buffer zone. The larger the hyper-rectangle the more conservative the the planning in the vicinity of the singularity. An attractive well potential is created such that its minimum is located at the desired final location in joint space $\mathbf{q}_f \in \mathbb{R}^{\mathrm{N}}$.

After the artificial potential function has been created for a specific robotic manipulator of N-DOF, it is possible to calculate the gradient of that function at any location in joint space $\nabla U_{global}\left(\mathbf{q}_i\right)$. The path planner is initialized at the current manipulator configuration $\mathbf{q}_i \in \mathbb{R}^{\mathrm{N}}$, afterwards the gradient of the potential function at that initial location is calculated. The next configuration is calculated using the negative of the gradient as follows: $\mathbf{q}_{i+1} = \mathbf{q}_i - \nabla U_{global}(\mathbf{q}_i)$. This process is repeated until the norm of the error between the current and desired configuration is less than a set tolerance $\|\mathbf{q}_{i+1} - \mathbf{q}_f\| < \varepsilon$. A flow chart illustrating this process is shown in Figure 5.1.

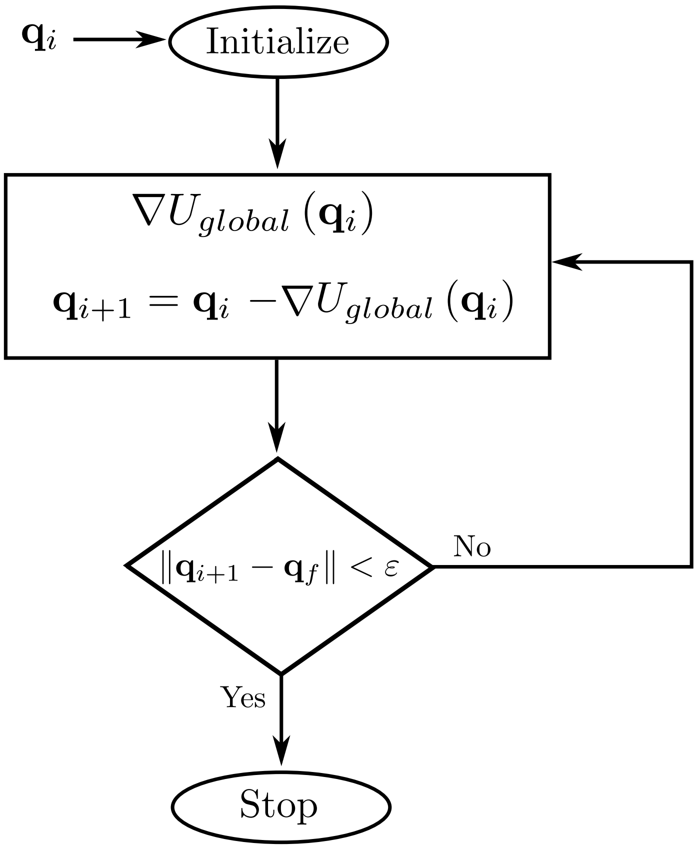
$\mathbf{q}_i$
Initialize
$\nabla U_{global}(\mathbf{q}_i)$
$\mathbf{q}_{i+1} = \mathbf{q}_i - \nabla U_{global}(\mathbf{q}_i)$
$\|\mathbf{q}_{i+1} - \mathbf{q}_f\| < \varepsilon$
No
Yes
Stop

Figure 5.1. Flow chart of the path planning algorithm using artificial potential functions.

5.3 Example - Three-Link Manipulator

To illustrate the implementability and benefit of the singularity-free APF path planner joint trajectories are generated to track a circular trajectory with a three-link manipulator end-effector in operational space. Details on the manipulator can be found in appendix A.

5.3.1 Trajectory Generation

The trajectory generation problem is solved using two methods; i) the APF method, ii) the linear interpolation method. The earlier is the described in detail in this chapter, while the latter creates a trajectory by linearly interpolating between the current joint configuration and the joint configuration at the next via point. The linear interpolation method will generate a trajectory between the two desired configurations but is not aware of the kinematic singularities. The results from both methods are compared. The following steps are used to generate the trajectories:

1. Set the points to be visited by the end-effector (via points) in operational space and arrange them in the order they are to be visited.
2. Initialize the robot joints with acceptable values.
3. Solve the IK problem at the first via point.
4. Plan a trajectory in joint space between the current configuration and the configuration calculated from solving the IK problem.
5. Solve the IK at the next via point.
6. Iterate again from step 4 until the via points are exhausted.

For each method of trajectory generation a dynamic simulation of the three-link manipulator with a computed torque controller is used to track the planned trajectory. The dynamic simulations are both identical such that the system parameters used for both are the same, and the system is initialized from the same configuration for both.

The only difference is the method of trajectory planning in step 4 of the steps given above.

5.3.2 Simulations

Two numerical simulations are conducted using the same robot kinematics, dynamics, and initial conditions. The first utilizes the APF planning method while the second uses the linear interpolation planning method. Both simulations are required to track a desired trajectory with the manipulator end-effector in operational space. The trajectory to be tracked in operational space is a circular one with a radius of $0.5\ m$ and a frequency of $2\pi\ rad/sec$. By dividing that desired trajectory into a number of via points and solving the IK at each point, the planners generate joint trajectories between those points in joint space. This trajectory is tracked using a computed torque controller [72]. Both simulations run for 50 seconds.

5.3.3 Results

Results for the operational space trajectories from the first simulation, using APF for planning joint trajectories, are given in Fig. 5.2. The via points over the circular trajectory are shown as black squares and connecting these points are solid black lines which are the planned trajectories. These results allow for the conclusion that the APF planner is capable of generating joint trajectories which allow for tracking the desired circular trajectory in operational space. It can also be seen from the same figure that the manipulator end-effector tracks the desired circular trajectory more than once over the 50 seconds of simulation.

The joint angles that generate this motion of the manipulator end-effector are given in Figs. 5.3, 5.4, and 5.5. These figures show the joint trajectories planned by the APF planner, represented by solid black lines. The computed torque controller used is able to track these joint trajectories exactly, as can be seen in all three figures. These results show that the generated joint trajectories by the APF planner are smooth, and easily tracked by a conventional control method.

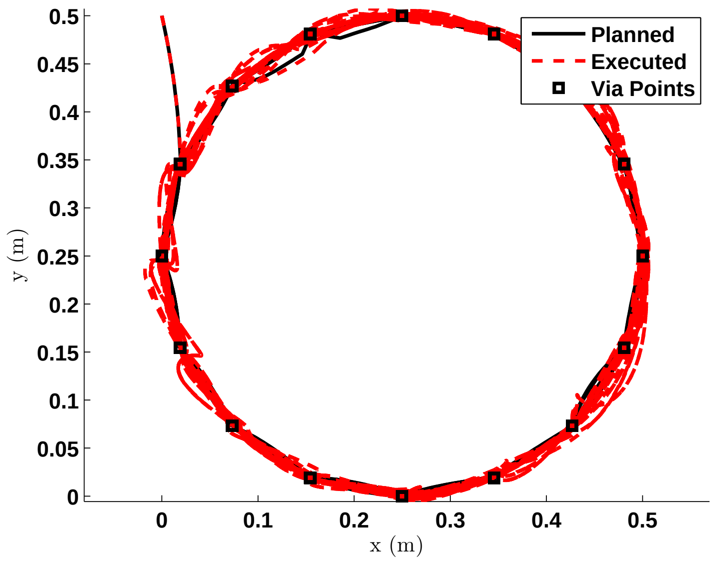

Figure 5.2. APF end-effector trajectory - planned and executed.

Results for the operational space trajectories from the second simulation, using linear interpolation for planning joint trajectories, are given in Fig. 5.6. The via points over the circular trajectory are shown as black squares and connecting these points are solid black lines which are the planned trajectories. The results in this figure clearly show that the system planned and executed trajectories are not satisfactory in the context of tracking a circular trajectory. This erratic and unpredictable motion is evidence of singularities affecting the IK solution and therefore the planning in joint space. Clearly the system in this simulation fails to track the desired trajectory.

The joint angles that generate this motion of the manipulator end-effector from simulation two are given in Figs. 5.7, 5.8, and 5.9. In Fig. 5.7 a large jump in the

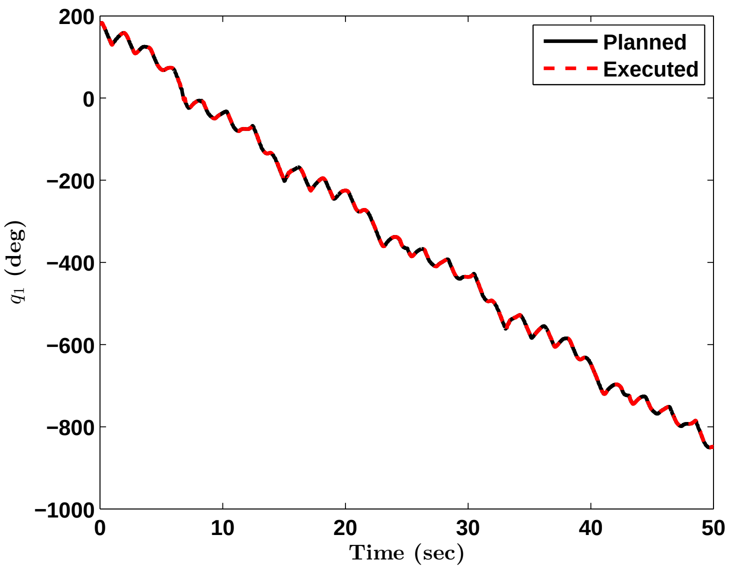

Figure 5.3. APF joint one trajectory - planned and executed.

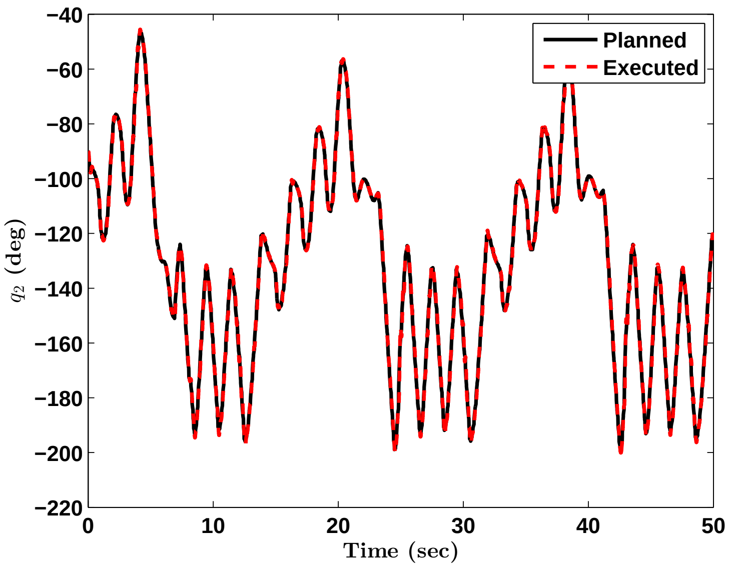

Figure 5.4. APF joint two trajectory - planned and executed.

planned joint trajectory can be seen before the 10 and 40 second mark. Similar jumps can be seen in Figs. 5.8, and 5.9 at the same time intervals. Despite the fact that there are large changes in the planned joint trajectories, the controller shows great

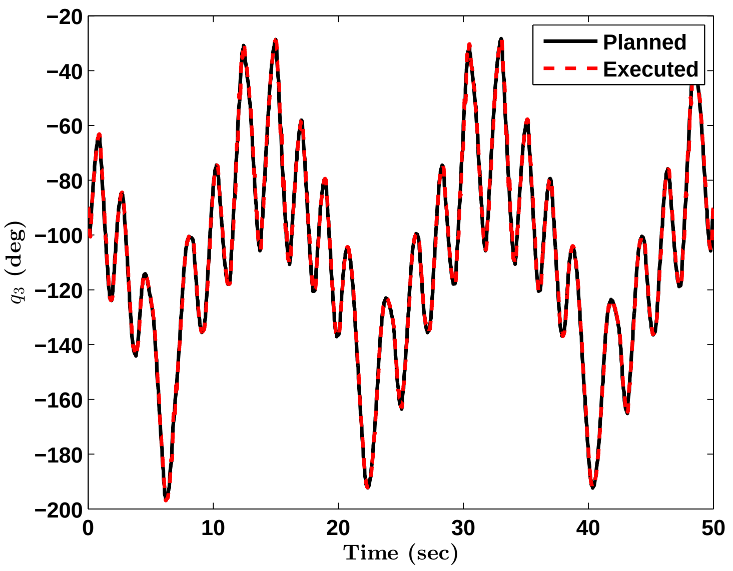

Figure 5.5. APF joint three trajectory - planned and executed.

capability in tracking those values. This comes at the price of large velocities as will be discussed later, however it is worthy of noting here that such results on an actual physical system may not be possible, therefore, the results in operational space may be substantially worse than in simulation.

The reason for these large jumps in desired joint trajectories can be explained by examining in Fig. 5.8; the analysis of the manipulator Jacobian given in appendix A shows that a singularity configuration occurs when joint number two is at a value of $n\pi$, where n is an integer. From Fig. 5.8 it can be seen that before the 10 and 40 second marks in simulation, joint two is approaching a value of $-\pi$. Despite the fact that this value appears in other locations along the joint trajectory, these locations are not via points, i.e. the IK problem was not solved at these points. This is the reason why the joint trajectories in the vicinity of these points does not include any large or abrupt changes in trajectory.

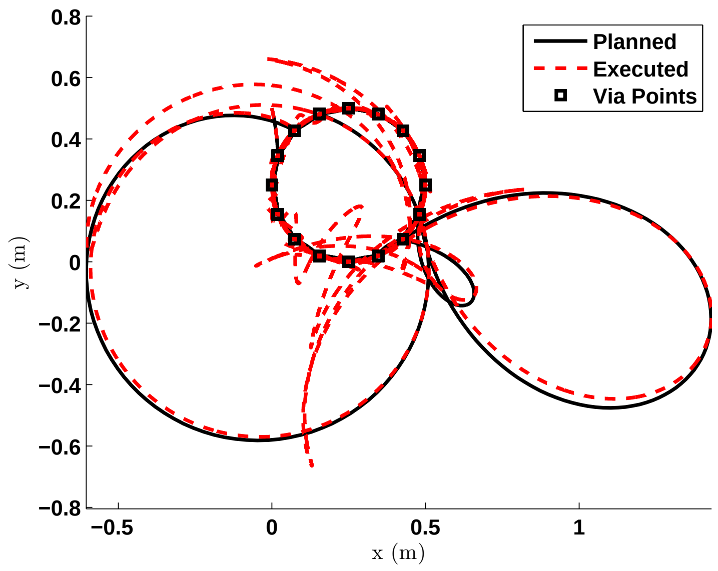

Figure 5.6. Linear interpolation end-effector trajectory - planned and executed.

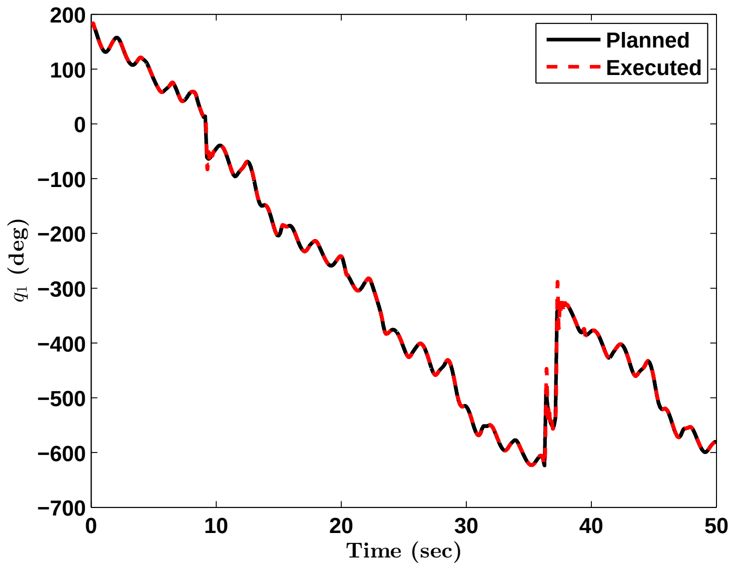

Figure 5.7. Linear interpolation joint one trajectory - planned and executed.

The effect of the singularities can also be seen when examining the manipulator joint velocities given in Figs. 5.10, 5.11, and 5.12. These figures show the joint velocities from both simulations, where the velocities from the simulation that uses

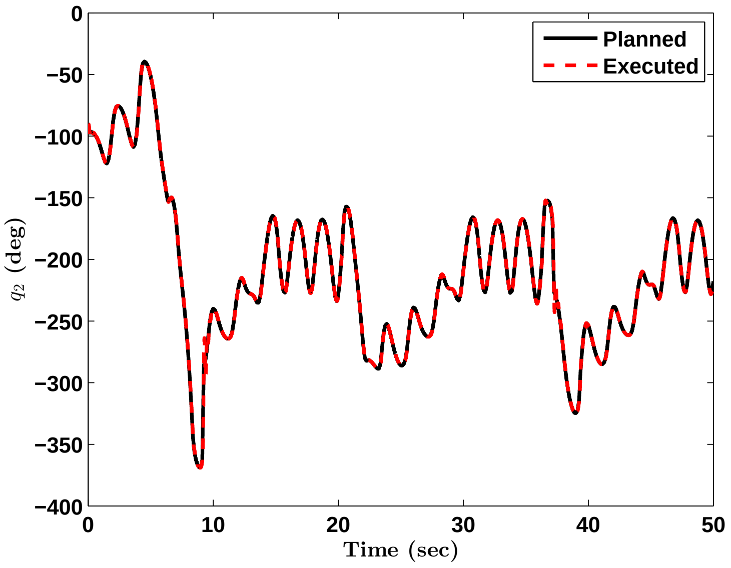

Figure 5.8. Linear interpolation joint two trajectory - planned and executed.

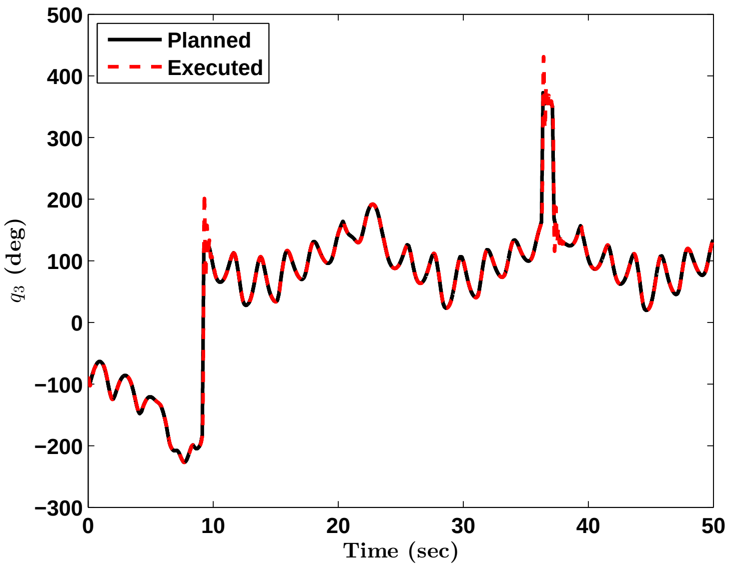

Figure 5.9. Linear interpolation joint three trajectory - planned and executed.

the APF planner is given in solid black and the velocities from the other simulation are given in dashed red. Again here in these results it is seen that the velocities for the manipulator in simulation two (linear interpolation planning) suffers from large magnitudes at the same time stamps where the joint positions exhibited large changes. These velocity jumps are classical symptoms of singularities. Such large joint velocities are not preferred in practice, and during implementation they can damage the system. From these results it can be concluded that the linear interpolation planner, which has no knowledge of the singular configurations, plans trajectories in joint space that lead to singular joint values at via points. When the IK is calculated at these via points the Jacobian inverse is numerically ill-conditioned and leads to such large joint velocities.

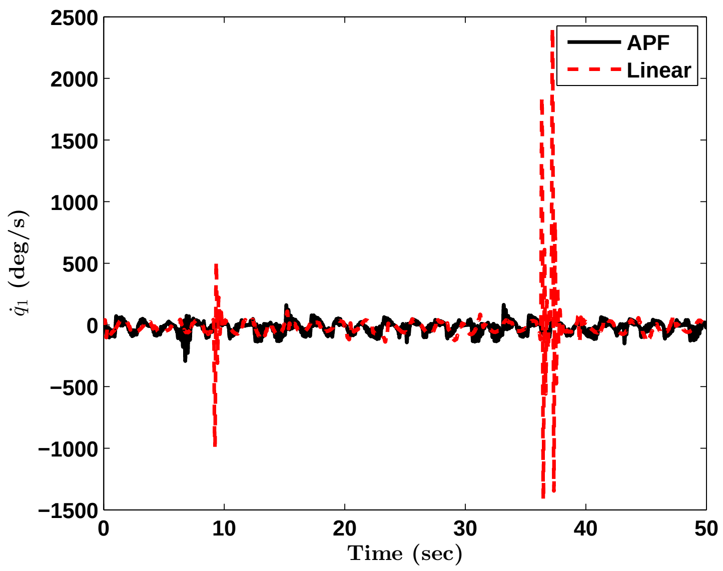

Figure 5.10. APF planning example - velocities of joint one.

A measure of the accuracy of computing a matrix inverse is the condition number C_n; this number is defined as the ratio between the maximum and minimum singular values of the matrix. The larger the condition number, the less accurate the matrix inverse is, in other words the matrix is ill-conditioned. For both simulations the condition number for the Jacobian matrix was calculated. Results are shown on

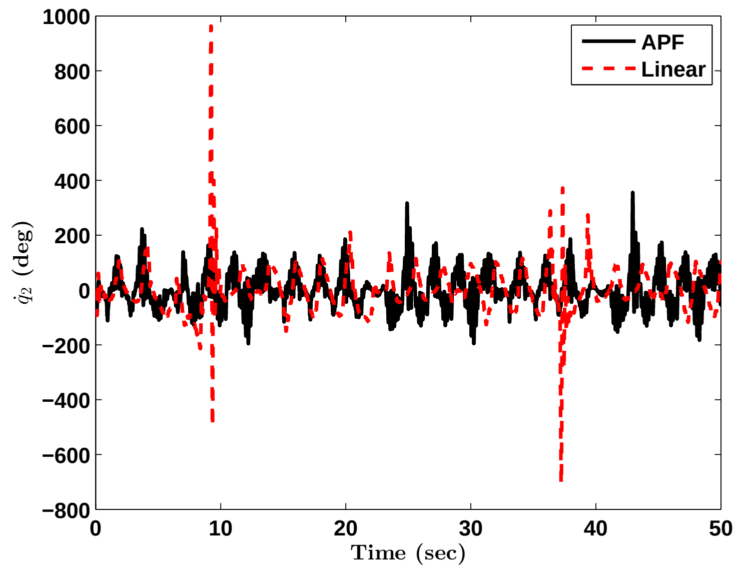

Figure 5.11. APF planning example - velocities of joint two.

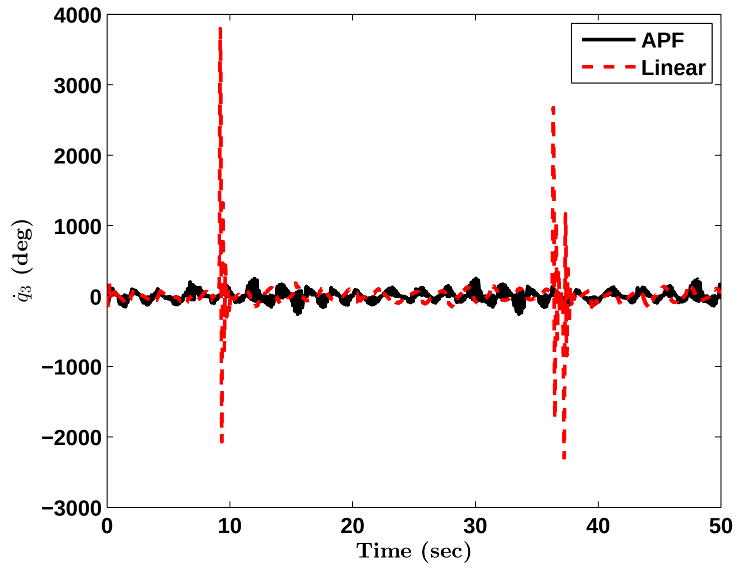

Figure 5.12. APF planning example - velocities of joint three.

the semi-logarithmic plot in Fig. 5.13. These results show that in the case of using the linear interpolation planner for joint trajectories the condition number of the Jacobian is substantially higher at approximately 10 and 38 seconds. These correspond to the singular configurations. This leads to the conclusion that the Jacobian matrix is more numerically conditioned for inversion over the joint trajectories planned by the APF planner, and therefore, the IK solution is more accurate and the resulting operational space trajectory is satisfactory.

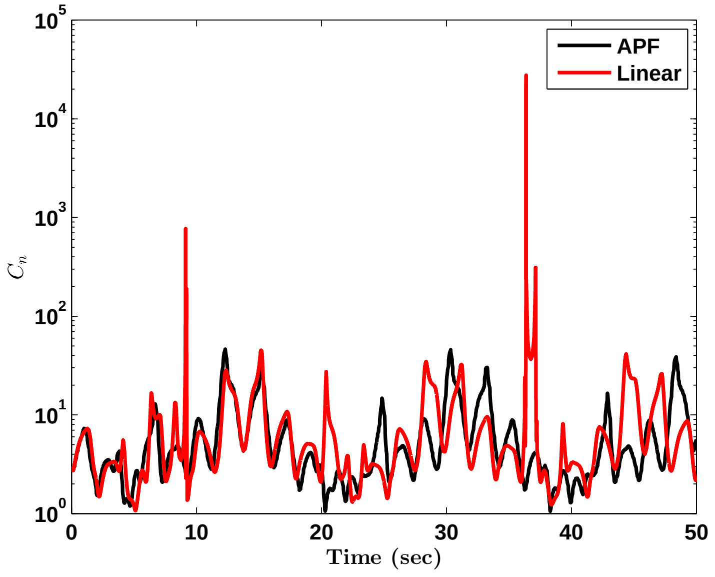

Figure 5.13. Jacobian condition number for APF example simulations.

CHAPTER 6

Singularity Robust Jacobian Inversion

As stated in chapter 2, the inverse kinematics (IK) problem requires finding the manipulator configuration given an end-effector position and orientation. While analytical solutions exist for a limited set of problems (typically small number of links), numerical solutions are typically sought for more complicated systems.

It is also shown in chapter 2 that the numerical solution to the IK problem requires inverting the Jacobian. For a square Jacobian matrix (the number of DOF are equal to the operational space coordinates) a straightforward matrix inversion can be done. However, in the case of a non-square Jacobian matrix, a problem that arises for redundant manipulators, methods such as the Moore-Penrose (MP) inverse or the utilization of a the matrix singular value decomposition can be used. In either case the solution is a static one, which means that the inversion can be applied at specific time instants and supplied to the IK solver. Given the time-varying nature of a robotic system and the therefore the IK problem these static solutions can lag behind in time. A continuous-in-time solution to the Jacobian inverse that can be implemented online would be more suited for solving the IK problem.

A major problem that arises when inverting the Jacobian matrix as seen from the simulation example in chapter 5 is the existance of singular configurations. At these configurations the Jacobian matrix is numerically ill-conditioned for inversion. To solve the IK problem over a joint trajectory that does not cause the Jacobian to suffer such ill-conditioning, singularity-free joint trajectories are generated in chapter 5. That solution however requires a priori knowledge of singular configurations, a requirement that might be difficult and tedious to meet when dealing with more complicated, higher DOF systems. For this reason, a method to calculate the Jacobian inverse in a singularity robust way is needed. This implies a method that is able to calculate the Jacobian matrix inverse, with acceptable accuracy, near or at singular configurations. A novel method for solving this problem is developed in this chapter using a modified model of the neuro-dynamic system also known as the The Zhang neural network (ZNN).

6.1 Matrix Generalized Inverse

6.1.1 The Moore-Penrose Inverse

Consider a matrix $\mathbf{A} \in \mathbb{R}^{\mathrm{n}\times\mathrm{m}}$ which is nonzero and full rank, if there exist another matrix $\mathbf{X} \in \mathbb{R}^{\mathrm{m}\times\mathrm{n}}$ that satisfies the following conditions

$$\begin{aligned} \mathbf{AXA} &= \mathbf{A} \\ \mathbf{XAX} &= \mathbf{X} \\ (\mathbf{AX})^T &= \mathbf{AX} \\ (\mathbf{XA})^T &= \mathbf{XA} \end{aligned} \tag{6.1}$$

then $\mathbf{X}$ is said to be the Moore-Penrose (MP) generalized inverse of $\mathbf{A}$ [73]. Moore-Penrose generalized inverse of $\mathbf{A}$ can be identified with the superscript $^+$, i.e. $\mathbf{A}^+$.

Proposition 6.1: Let $\mathbf{A} \in \mathbb{R}^{\mathrm{n}\times\mathrm{m}}$. If $\mathbf{A}$ is right invertible, then $\mathbf{A}$ has full row rank i.e $\mathbf{AA}^T$ is invertible and

$$\mathbf{A}^+ = \mathbf{A}^T(\mathbf{AA}^T)^{-1} \tag{6.2}$$

and $\mathbf{A}^+$ is the right inverse of $\mathbf{A}$. If $\mathbf{A}$ is left invertible, then $\mathbf{A}$ has full column rank i.e $\mathbf{A}^T\mathbf{A}$ is invertible and

$$\mathbf{A}^+ = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T \tag{6.3}$$

Proof. It is sufficient to verify the four conditions in 6.1 with $\mathbf{X} = \mathbf{A}^+$.

6.1.2 Damped-Least-Squares Inverse

The Levenberg-Marquardt method, otherwise known as the damped-least-squares (DLS) method, has been typically utilized to avoid the problem of inverting numerically ill-conditioned matrices at singular configurations [74]. To solve the problem of IK using the DLS method it is typically posed as an optimization problem, where $\dot{\mathbf{q}}$ must be selected so that the following quantity is minimized

$$\|\mathbf{J}(\mathbf{q})\,\dot{\mathbf{q}} - \dot{\mathbf{x}}_e\|^2 + \lambda\|\dot{\mathbf{q}}\|^2 \tag{6.4}$$

where λ is a non-zero constant. The quantity in Eq. 6.4 can be written as

$$\|[\mathbf{J}(\mathbf{q}) \quad \lambda\mathbf{I}]^T\,\dot{\mathbf{q}} - [\dot{\mathbf{x}}_e \quad \mathbf{0}]^T\|^2 \tag{6.5}$$

where $\mathbf{I}$ is the identity matrix. The unique solution is given as

$$\dot{\mathbf{q}} = \mathbf{J}^T(\mathbf{q})\left(\mathbf{J}(\mathbf{q})\,\mathbf{J}^T(\mathbf{q}) + \lambda^2\mathbf{I}\right)^{-1}\dot{\mathbf{x}}_e \tag{6.6}$$

By comparing Eq. 6.6 to the differential kinematics in Eq. 2.4, the DLS inverse of the Jacobian can be defined as

$$\mathbf{J}^\dagger(\mathbf{q}) \equiv \mathbf{J}^T(\mathbf{q})\left(\mathbf{J}(\mathbf{q})\,\mathbf{J}^T(\mathbf{q}) + \lambda^2\mathbf{I}\right)^{-1} \tag{6.7}$$

Proposition 6.2: Let $\mathbf{A} \in \mathbb{R}^{\mathrm{n}\times\mathrm{m}}$. The DLS inverse of $\mathbf{A}$ is defined as

$$\mathbf{A}^\dagger = \mathbf{A}^T\left(\mathbf{A}\mathbf{A}^T + \lambda^2\mathbf{I}\right)^{-1} \tag{6.8}$$

Proof. As shown in equations 6.4 - 6.6.

6.2 Problem Statement

Given a matrix $\mathbf{A} \in \mathbb{R}^{\mathrm{n}\times\mathrm{m}}$, it is required to find the equation that describes the dynamics of a system whose solution $\mathbf{X}(t)$ converges to $\mathbf{A}^\dagger$ as $t \to \infty$. The system dynamics can be expressed in the following form

$$\dot{\mathbf{X}}(t) = \mathcal{H}\left(\mathbf{A}, \dot{\mathbf{A}}\right) \tag{6.9}$$

6.3 The Zhang Neural Network Model for MP Inverse of a Matrix

The ZNN, which is a special kind of recurrent neural networks otherwise referred to as a neuro-dynamic system, was introduced in 2001. It has been used to solve a variety of time-varying problems [32]-[37], one of which is the online solution of the MP inverse of time-varying full rank matrices [38]. Unlike traditional neural networks that define a scalar-valued error function the ZNN utilizes a matrix valued error function to find the solution to time-varying matrix problems.

Using the definition of a right inverse for a time-varying matrix (Eq. 6.2) a matrix-based error function can be defined as

$$\mathbf{E}(\mathbf{X}(t),t) = \mathbf{X}(t)(\mathbf{A}(t)\mathbf{A}^T(t)) - \mathbf{A}^T(t) \tag{6.10}$$

where $\mathbf{X}(t) \in \mathbb{R}^{\mathrm{m}\times\mathrm{n}}$ is the MP right inverse we want to find an expression for, and clearly when $\mathbf{X}(t) = \mathbf{A}^{+}(t)$ the error $\mathbf{E}(\mathbf{X}(t),t) = 0$.

The work in [38] derives the following ZNN model for the right MP inverse of a matrix as

$$\begin{aligned}\dot{\mathbf{X}}(t)\mathbf{A}(t)\mathbf{A}^T(t) = & -\gamma\mathcal{F}\left(\mathbf{X}(t)\mathbf{A}(t)\mathbf{A}^T(t) - \mathbf{A}^T(t)\right)...\\ & ... - \mathbf{X}(t)\left(\dot{\mathbf{A}}(t)\mathbf{A}^T(t) + \mathbf{A}(t)\dot{\mathbf{A}}^T(t)\right)...\\ & ... + \dot{\mathbf{A}}^T(t)\end{aligned} \tag{6.11}$$

where $\gamma \in \mathbb{R}$ is a scaling parameter that is greater than zero and used to control the convergence rate of the neural network. The function $\{\mathcal{F} : \mathbb{R}^{\mathrm{m}\times\mathrm{n}} \rightarrow \mathbb{R}^{\mathrm{m}\times\mathrm{n}}\}$ is a matrix-to-matrix activation function array. For the ijth element of $\mathcal{F}(\cdot)$ any monotonically-increasing function can be used such as a linear or power sigmoid function. The function used in [38] will also be used in the work presented here and is defined as follows

$$f\left(e_{ij}\right) = \begin{cases} e_{ij}^{p}, & \text{if } |e_{ij}| \geqslant 1 \\ \frac{1+\exp(-\zeta)}{1-\exp(-\zeta)} \cdot \frac{1+\exp(-\zeta e_{ij})}{1-\exp(-\zeta e_{ij})} & \text{otherwise} \end{cases} \tag{6.12}$$

where $e_{ij} \in \mathbb{R}$ are the elements of the error matrix $\mathbf{E}(\mathbf{X}(t), t) \in \mathbb{R}^{\mathrm{m}\times\mathrm{n}}$, $p \in \mathbb{R} \geq 3$, and $\zeta \in \mathbb{R} \geq 2$. The ZNN model given in 6.11 was proven to globally converge, given any initial state $\mathbf{X}(0)$, to the time-varying theoretical right MP inverse of a matrix $\mathbf{A} \in \mathbb{R}^{\mathrm{n}\times\mathrm{m}}$, i.e., $\mathbf{A}^{+}$ $\mathbb{R}^{\mathrm{m}\times\mathrm{n}}$ [38].

Similarly, using the definition of a left inverse for a time-varying matrix (Eq. 6.3) a matrix-based error function can be defined as

$$\mathbf{E}(\mathbf{X}(t), t) = \mathbf{A}^T(t)\mathbf{A}(t)\mathbf{X}(t) - \mathbf{A}^T(t) \tag{6.13}$$

where $\mathbf{X}(t) \in \mathbb{R}^{\mathrm{m}\times\mathrm{n}}$ is the left MP left inverse we want to find an expression for, and clearly when $\mathbf{X}(t) = \mathbf{A}^{+}(t)$ the error $\mathbf{E}(\mathbf{X}(t), t) = 0$.

The work in [38] derives the following ZNN model for the left MP inverse of a matrix as

$$\begin{aligned}\mathbf{A}^T(t)\mathbf{A}(t)\dot{\mathbf{X}}(t) = &-\gamma\mathcal{F}\left(\mathbf{A}^T(t)\mathbf{A}(t)\mathbf{X}(t) - \mathbf{A}^T(t)\right)...\\ &...-\left(\dot{\mathbf{A}}^T(t)\mathbf{A}(t) + \mathbf{A}^T(t)\dot{\mathbf{A}}(t)\right)\mathbf{X}(t)...\\ &...+\dot{\mathbf{A}}^T(t)\end{aligned} \tag{6.14}$$

The neuro-dynamics given in Eq. 6.11 and Eq. 6.14 are a continuous-time solution to the matrix inversion problem. They allow for solving the matrix inverse problem online and therefore are good candidates for acquiring the inverse of the Jacobian matrix for solving the IK problem numerically. However, to derive the neuro-dynamics that solve for the matrix inverse, the assumption is made that the matrix is never singular, i.e. the matrix is always full row or column rank. When solving the IK problem for an N-DOF robotic manipulator, that condition is not always met unless explicitly accounted for in the path planning phase as is done in chapter 5. This implies that while providing an online method to solve for the Jacobian inverse in a continuous-time manner, the neuro-dynamics given in Eq. 6.11 and Eq. 6.14 will not be able to solve the matrix inverse problem at, or near, singular configurations. Therefore, it is proposed in this work to use a different matrix-based error function and derive new neuro-dynamics for a ZNN capable of handling singular, or near singular cases.

6.4 The Modified Zhang Neural Network (M-ZNN) Model

The DLS, has been typically utilized to avoid the problem of inverting numerically ill-conditioned matrices at singular configurations [74]. Using the definition in of the DLS inverse of the matrix $\mathbf{A} \in \mathbb{R}^{\mathrm{n}\times\mathrm{m}}$ as given in Eq. 6.8, one can define the following matrix-based error function

$$\mathbf{E}(\mathbf{X}(t), t) = \mathbf{X}(t)\left(\mathbf{A}(t)\mathbf{A}(t)^T + \lambda^2\mathbf{I}\right) - \mathbf{A}^T(t) \tag{6.15}$$

where $\mathbf{X}(t) \in \mathbb{R}^{\mathrm{m}\times\mathrm{n}}$ is the DLS inverse we want to find an expression for, and clearly when $\mathbf{X}(t) = \mathbf{A}^{\dagger}(t)$ the error $\mathbf{E}(\mathbf{X}(t), t) = 0$. In order for the error in Eq. 6.15 to converge to zero its time derivative can be set to the following form

$$\begin{aligned}\dot{\mathbf{E}}(\mathbf{X}(t), t) &= -\gamma\mathcal{F}\left(\mathbf{E}(\mathbf{X}(t), t)\right) \\ &= -\gamma\mathcal{F}\left(\mathbf{X}(t)\left(\mathbf{A}(t)\mathbf{A}(t)^T + \lambda^2\mathbf{I}\right) - \mathbf{A}^T(t)\right)\end{aligned} \tag{6.16}$$

where $\gamma \in \mathbb{R}$ is a scaling parameter that is greater than zero and used to control the convergence rate of the neural network. By taking the time derivative of 6.15 one gets

$$\begin{aligned}\dot{\mathbf{E}}(\mathbf{X}(t), t) = \dot{\mathbf{X}}(t)&\left(\mathbf{A}(t)\mathbf{A}(t)^T + \lambda^2\mathbf{I}\right) ... \\ ... + \mathbf{X}(t)&\left(\dot{\mathbf{A}}(t)\mathbf{A}(t)^T + \mathbf{A}(t)\dot{\mathbf{A}}(t)^T\right) - \dot{\mathbf{A}}^T(t)\end{aligned} \tag{6.17}$$

Combining Eq. 6.16 and Eq. 6.17

$$\begin{aligned}\dot{\mathbf{X}}(t)\left(\mathbf{A}(t)\mathbf{A}(t)^T + \lambda^2\mathbf{I}\right) = &-\gamma\mathcal{F}\left(\mathbf{X}(t)\left(\mathbf{A}(t)\mathbf{A}(t)^T + \lambda^2\mathbf{I}\right) - \mathbf{A}^T(t)\right) ... \\ &... - \mathbf{X}(t)\left(\dot{\mathbf{A}}(t)\mathbf{A}(t)^T + \mathbf{A}(t)\dot{\mathbf{A}}(t)^T\right) ... \\ &... + \dot{\mathbf{A}}^T(t)\end{aligned} \tag{6.18}$$

comparing Eq. 6.11 to the newly derived equation given in Eq. 6.18, it is clearly seen that a difference manifests in Eq. 6.18 due to the appearance of the term $\lambda^2\mathbf{I}$, where λ is a non-zero constant and $\mathbf{I}$ is the identity matrix. This term is due to the newly defined matrix error function in Eq. 6.15, and constitutes the damping constant in the DLS matrix inverse definition. The dynamics in Eq. 6.18 can be written as

$$\dot{\mathbf{X}}(t)\mathcal{M}(t) = \dot{\mathbf{A}}^T(t) - \gamma\mathcal{F}\left(\mathbf{X}(t)\mathcal{M}(t) - \mathbf{A}^T(t)\right) - \mathbf{X}(t)\mathcal{G}(t) \tag{6.19}$$

where

$$\mathcal{M}(t) = \mathbf{A}(t)\mathbf{A}(t)^T + \lambda^2\mathbf{I}$$

$$\mathcal{G}(t) = \dot{\mathbf{A}}(t)\mathbf{A}(t)^T + \mathbf{A}(t)\dot{\mathbf{A}}(t)^T$$

Equation 6.19 represents the neuro-dynamics for calculating the DLS inverse of a matrix, which can be represented in the following form by adding $\dot{\mathbf{X}}(t)$ to both sides of the equation

$$\dot{\mathbf{X}}(t) = \dot{\mathbf{X}}(t)\left(\mathbf{I} - \mathcal{M}(t)\right) + \dot{\mathbf{A}}^T(t) - \gamma\mathcal{F}\left(\mathbf{X}(t)\mathcal{M}(t) - \mathbf{A}^T(t)\right) - \mathbf{X}(t)\mathcal{G}(t) \tag{6.20}$$

The neuro-dynamic model given in Eq. 6.20 is that of the modified Zhang neural network (M-ZNN). One can also write down the ijth neuron dynamic equation explicitly as

$$\dot{x}_{ij} = \sum_{l=1}^{m} \dot{x}_{il}\left(\delta_{lj} - m_{lj}\right) - \gamma f\left(\sum_{l=1}^{m} x_{il}m_{lj} - \alpha_{ij}\right) - \sum_{l=1}^{m} x_{il}g_{lj} + \dot{\alpha}_{ij} \tag{6.21}$$

where $x_{ij} \in \mathbb{R}$ is the ijth element of $\mathbf{X}(t) \in \mathbb{R}^{\mathrm{m}\times\mathrm{n}}$ which is also the state of the ijth neuron of the M-ZNN, m_{ij} is the ijth element of $\mathcal{M}(t) \in \mathbb{R}^{\mathrm{n}\times\mathrm{n}}$ and is the term responsible for making the neuro-dynamics robust to singularities, α_{ij} is the ijth element of $\mathbf{A}^T(t) \in \mathbb{R}^{\mathrm{m}\times\mathrm{n}}$, g_{ij} is the ijth element of $\mathcal{G}(t) \in \mathbb{R}^{\mathrm{n}\times\mathrm{n}}$, and δ_{lj} is the Kronecker delta function. The structure of the M-ZNN given by Eq. 6.21 is shown in Fig. 6.1 where the modifications to the original ZNN used to calculate the MP inverse are shown in red.

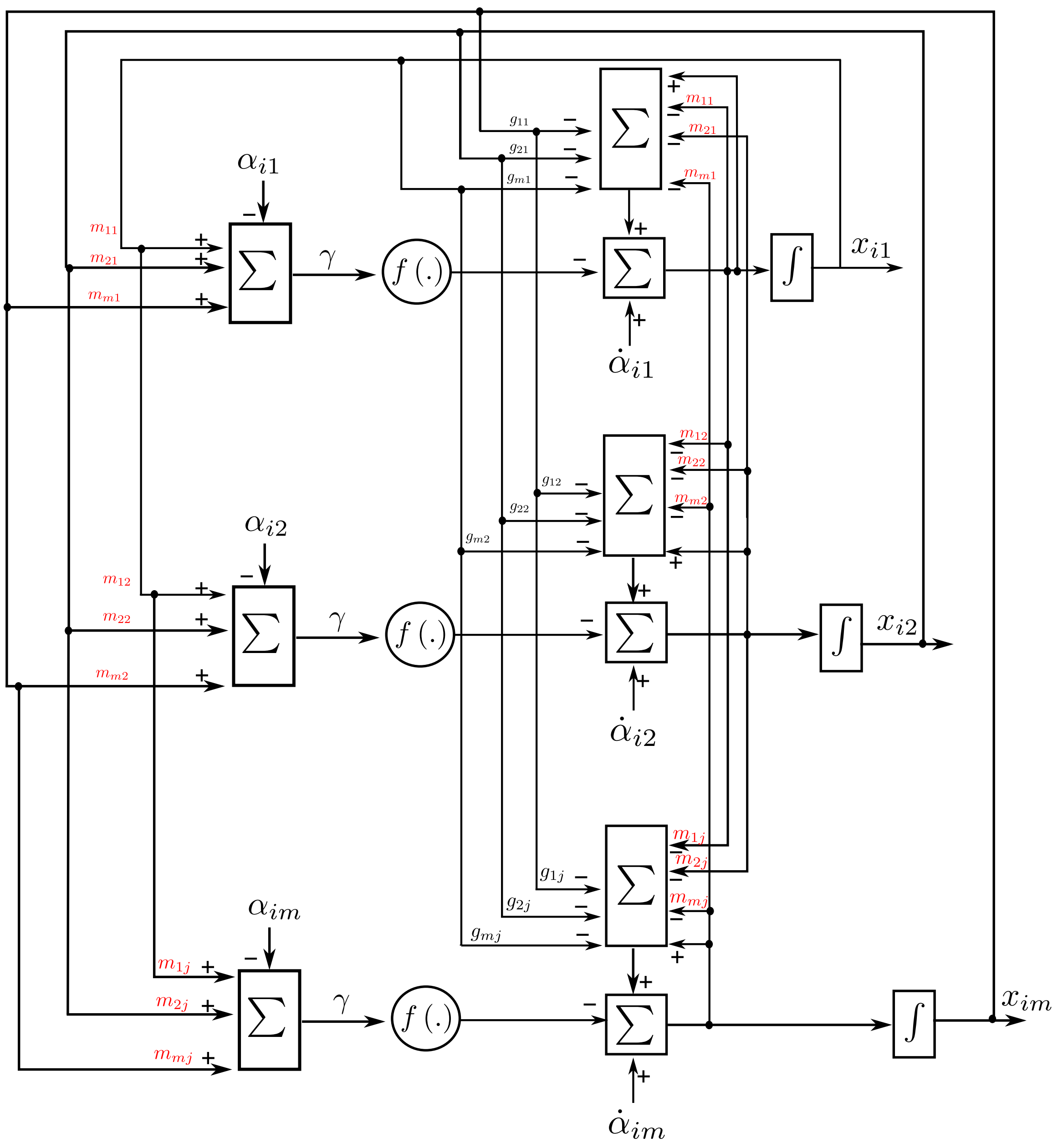

Figure 6.1. Schematic depicting the modified Zhang neural network structure.

Theorem 5.1 *Given a smoothly time-varying matrix* $\mathbf{A}(t) \in \mathbb{R}^{\mathrm{n}\times\mathrm{m}}$*, the state matrix of the neuro-dynamics of the M-ZNN given in Eq. 6.19,* $\mathbf{X}(t) \in \mathbb{R}^{\mathrm{m}\times\mathrm{n}}$*, starting from any initial state will globally convergence to the DLS inverse of* $\mathbf{A}(t)$*,* $\mathbf{A}^{\dagger}(t) \in \mathbb{R}^{\mathrm{m}\times\mathrm{n}}$*, if an odd, monotonically-increasing activation function array is used for* $\mathcal{F}(.)$*.*

Proof. Define the difference between the DLS inverse solution from the M-ZNN model $\mathbf{X}(t) \in \mathbb{R}^{\mathrm{m}\times\mathrm{n}}$ and the theoretical solution $\mathbf{A}^{\dagger}(t) \in \mathbb{R}^{\mathrm{m}\times\mathrm{n}}$ as $\tilde{\mathbf{X}}(t) \in \mathbb{R}^{\mathrm{m}\times\mathrm{n}}$ $= \mathbf{X}(t) - \mathbf{A}^{\dagger}(t)$, therefore

$$\mathbf{X}(t) = \tilde{\mathbf{X}}(t) + \mathbf{A}^{\dagger}(t) \tag{6.22}$$

Substituting $\mathbf{X}(t)$ from Eq. 6.22 into Eq. 6.19 gives

$$\begin{aligned} &\dot{\tilde{\mathbf{X}}}(t)\mathcal{M}(t) + \dot{\mathbf{A}}^{\dagger}(t)\mathcal{M}(t) + \tilde{\mathbf{X}}(t)\mathcal{G}(t) + \mathbf{A}^{\dagger}(t)\mathcal{G}(t) - \dot{\mathbf{A}}^{T}(t)... \\ &\qquad ... + \gamma\mathcal{F}\left(\tilde{\mathbf{X}}(t)\mathcal{M}(t) + \mathbf{A}^{\dagger}(t)\mathcal{M}(t) - \mathbf{A}^{T}(t)\right) = 0 \end{aligned} \tag{6.23}$$

from the definition of $\mathbf{A}^{\dagger}(t)$ in Eq. 6.8 and the definition of $\mathcal{M}(t) = \mathbf{A}(t)\mathbf{A}(t)^{T} + \lambda^{2}\mathbf{I}$, the term $\mathbf{A}^{\dagger}(t)\mathcal{M}(t) - \mathbf{A}^{T}(t) = 0$, therefore,

$$\mathbf{E}(\tilde{\mathbf{X}}(t), t) = \tilde{\mathbf{X}}(t)\mathcal{M}(t) \tag{6.24}$$

and therefore Eq. 6.23 simplifies to

$$\dot{\tilde{\mathbf{X}}}(t)\mathcal{M}(t)+\dot{\mathbf{A}}^{\dagger}(t)\mathcal{M}(t)+\tilde{\mathbf{X}}(t)\mathcal{G}(t)+\mathbf{A}^{\dagger}(t)\mathcal{G}(t)-\dot{\mathbf{A}}^{T}(t)+\gamma\mathcal{F}\left(\tilde{\mathbf{X}}(t)\mathcal{M}(t)\right) = 0 \tag{6.25}$$

Now, from the definition of the DLS inverse in Eq.6.8 one can write

$$\mathbf{A}^{\dagger}(t)\left(\mathbf{A}(t)\mathbf{A}^{T}(t) + \lambda^{2}\mathbf{I}\right) - \mathbf{A}^{T}(t) = 0 \tag{6.26}$$

taking the time derivative of Eq. 6.26 yields,

$$\dot{\mathbf{A}}^{\dagger}(t)\left(\mathbf{A}(t)\mathbf{A}^{T}(t) + \lambda^{2}\mathbf{I}\right) + \mathbf{A}^{\dagger}(t)\left(\dot{\mathbf{A}}(t)\mathbf{A}^{T}(t) + \mathbf{A}(t)\dot{\mathbf{A}}^{T}(t)\right) - \dot{\mathbf{A}}^{T} = 0 \tag{6.27}$$

therefore

$$\dot{\mathbf{A}}^{\dagger}(t)\mathcal{M}(t) + \mathbf{A}^{\dagger}(t)\mathcal{G}(t) - \dot{\mathbf{A}}^{T}(t) = 0 \tag{6.28}$$

using equation 6.28, Eq. 6.25 becomes

$$\dot{\tilde{\mathbf{X}}}(t)\mathcal{M}(t) + \tilde{\mathbf{X}}(t)\mathcal{G}(t) + \gamma\mathcal{F}\left(\tilde{\mathbf{X}}(t)\mathcal{M}(t)\right) = 0 \tag{6.29}$$

By taking the time derivative of Eq. 6.24 one gets

$$\dot{\mathbf{E}}(\tilde{\mathbf{X}}(t), t) = \dot{\tilde{\mathbf{X}}}(t)\mathcal{M}(t) + \tilde{\mathbf{X}}(t)\mathcal{G}(t) \tag{6.30}$$

then by substituting Eq. 6.30 into Eq. 6.29 one arrives at

$$\dot{\mathbf{E}}(\tilde{\mathbf{X}}(t), t) + \gamma\mathcal{F}\left(\mathbf{E}(\tilde{\mathbf{X}}(t), t)\right) = 0 \tag{6.31}$$

The dynamics given in Eq. 6.31 are the error dynamics of the neural network that is used to solve for the DLS inverse of a matrix. These dynamics can be written in element form as a set of $m \times n$ differential equations as

$$\dot{e}_{ij} + \gamma f\left(e_{ij}\right) = 0 \tag{6.32}$$

Let the following be a candidate Lyapunov function for the ijth system in Eq. 6.32

$$v_{ij} = \frac{e_{ij}^2}{2} \tag{6.33}$$

Note that $v_{ij} > 0$ for $e_{ij} \neq 0$ and $v_{ij} = 0$ for $e_{ij} = 0$ therefore, it is positive definite. The time derivative of Eq. 6.33 is

$$\dot{v}_{ij} = e_{ij}\dot{e}_{ij} = -\gamma e_{ij} f\left(e_{ij}\right) \tag{6.34}$$

If $f(.)$ is an odd, monotonically-increasing function then

$$f\left(e_{ij}\right) = \begin{cases} > 0, & \text{if } e_{ij} > 0 \\ = 0, & \text{if } e_{ij} = 0 \\ < 0, & \text{if } e_{ij} < 0 \end{cases} \tag{6.35}$$

therefore, $\dot{v}_{ij} < 0$ for $e_{ij} \neq 0$ and $\dot{v}_{ij} = 0$ for $e_{ij} = 0$ and so $\dot{v}_{ij}$ is said to be negative definite. Based on the properties of v_{ij} and $\dot{v}_{ij}$, v_{ij} can be considered a valid Lyapunov function, which is also decrescent, therefore according to Lyapunov stability theory the origin of the system in Eq. 6.32 is globally uniformly asymptotically stable [75]. This implies that $\mathbf{E}(\tilde{\mathbf{X}}(t), t) \to 0$ as $t \to \infty$, therefore, $\tilde{\mathbf{X}}(t) \to 0$ as $t \to \infty$, which implies that $\mathbf{X}(t) \to \mathbf{A}^{\dagger}$ as as $t \to \infty$.

6.5 Solving the Neuro-Dynamic Equations

The neuro-dynamic model given in Eq. 6.19 is a nonlinear matrix differential equation. The solution to such an equation can be obtained using numerical integration, however, it must be put in a form that allows for using numerical integration methods. This can be done using the Kronecker product and vectorization techniques [32]. Therefore, Eq. 6.19 can be written in the following vector differential equation format

$$\begin{aligned}\left(\mathcal{M}^T(t) \otimes \mathbf{I}\right) \operatorname{vec}\left(\dot{\mathbf{X}}(t)\right) = \\ -\gamma \mathcal{F}\left(\left(\mathcal{M}^T(t) \otimes \mathbf{I}\right) \otimes \operatorname{vec}\left(\mathbf{X}(t)\right) - \operatorname{vec}\left(\mathbf{A}^T(t)\right)\right) \ldots \\ \ldots \quad - \left(\mathcal{G}^T(t) \otimes \mathbf{I}\right) \operatorname{vec}\left(\mathbf{X}(t)\right) + \operatorname{vec}\left(\dot{\mathbf{A}}^T(t)\right)\end{aligned} \tag{6.36}$$

where $\otimes$ is the Kronecker product and vec(.) is the vectorization operator, both are defined below using matrices $\mathbf{M}, \mathbf{N} \in \mathbb{R}^{n \times n}$.

This vector form of the neuro-dynamics for the M-ZNN can be used to calculate the inverse of the Jacobian matrix for any N-DOF manipulator, i.e. the Jacobian is numerically conditioned for inversion over the joint trajectory. This solution relaxes the requirement of having a priori knowledge of the manipulator kinematic singularities. An example of implementing the neuro-dynamic system on a three-link manipulator can be found in chapter 7.

$$\operatorname{vec}(\mathbf{M}) = \begin{bmatrix} m_{11} \\ m_{21} \\ \vdots \\ m_{n1} \\ m_{12} \\ \vdots \\ m_{nn} \end{bmatrix} \qquad \text{where} \qquad \mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \cdots & m_{nn} \end{bmatrix} \tag{6.37}$$

$$\mathbf{M} \otimes \mathbf{N} = \begin{bmatrix} m_{11}\mathbf{N} & m_{12}\mathbf{N} & \cdots & m_{1n}\mathbf{N} \\ m_{21}\mathbf{N} & m_{22}\mathbf{N} & \cdots & m_{2n}\mathbf{N} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1}\mathbf{N} & m_{n2}\mathbf{N} & \cdots & m_{nn}\mathbf{N} \end{bmatrix} \tag{6.38}$$

CHAPTER 7

Neuro-Adaptive Learning Control

In chapter 4 a method for generating operational space trajectories using DNNs was detailed. Two methods for generating joint space trajectories along which the Jacobian matrix is invertible were presented in chapters 5 and 6. These joint trajectories need to be tracked in order for the operational space trajectory to be executed. To do so a control system must be designed to ensure that the robot achieves the desired joint trajectories. That control system is the topic of this chapter.

The design of a model-free, adaptive, joint controller for an N-DOF robotic manipulator is discussed in this chapter. Using a FFNN that is capable of online learning the nonlinear dynamics of the robot can be approximated in real-time. This approximation along with a filtered error signal are used to generate input torques to the joints of the robot. The controller tracks desired joint trajectories.

Model-free, adaptive control allows for a control systems that account for uncertainty. When deriving a model for the robot dynamics it is impossible to account for every detail, therefore, modeling uncertainty is inevitable. Controllers designed based on these models will not be able to account for any dynamics that occur during implementation and are not included in the mode. To avoid this problem, the work presented here utilizes a FFNN to approximate the nonlinear dynamics of the manipulator. Using a two-layer FFNN that is capable of online learning, i.e. the FFNN weights are updated online in an adaptive manner, a neuro-adaptive controller is designed. The approximated nonlinear dynamics are used in an inner loop of the control architecture. This inner loop cancels the system nonlinearity and renders a

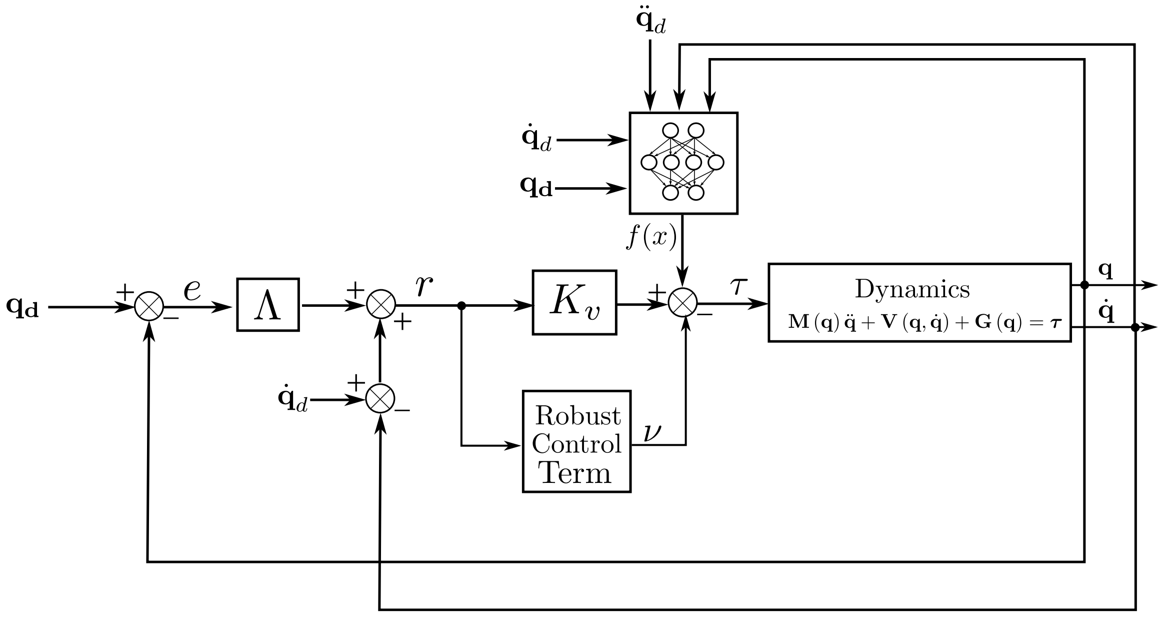

Figure 7.1. Block diagram illustrating the neuro-adaptive control structure.

tracking outer loop that can be dealt with easily through a filtered error term. The neuro-adaptive control law structure is shown in the block diagram in Figure 7.1.

This controller is based on the seminal work of [76, 77, 17, 78], which has recently been implemented on a 28-DOF humanoid robot [79].

7.1 Approximation of Robot Nonlinear Dynamics

The dynamics of a robot manipulator are typically derived using energy methods (Lagrange's equations), the iterative Newton-Euler method, Kane's method and others. Not withstanding the method employed, the governing equations take the following form

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}+\mathbf{V}(\mathbf{q},\dot{\mathbf{q}})+\mathbf{G}(\mathbf{q})=\boldsymbol{\tau}+\boldsymbol{\tau}_E \tag{7.1}$$

where $\mathbf{q}\in\mathbb{R}^{\mathrm{N}\times 1}$, $\mathbf{M}(\mathbf{q})\in\mathbb{R}^{\mathrm{N}\times\mathrm{N}}$ is symmetric mass matrix, $\mathbf{V}(\mathbf{q},\dot{\mathbf{q}})\in\mathbb{R}^{\mathrm{N}\times 1}$ is the Coriolis/centripetal vector, $\mathbf{G}(\mathbf{q})\in\mathbb{R}^{\mathrm{N}\times 1}$ is the gravity vector, $\boldsymbol{\tau}\in\mathbb{R}^{\mathrm{N}\times 1}$ is the input torque vector, and $\boldsymbol{\tau}_E\in\mathbb{R}^{\mathrm{N}\times 1}$ represents the unmodeled disturbance torques.

This dynamic model is nonlinear in nature, and the parameters associated with it are not always easily quantifiable. Therefore, having a controller that is capable of estimating this model in an adaptive manner based on the system's operational state, and generate signals for controlling it reduces the uncertainty that would accompany a model-based controller.

If there exist a bounded trajectory $\mathbf{q}_d(t) \in C^2$; i.e $\mathbf{q}^d(t)$ is bounded and twice differentiable with respect to time and $\ddot{\mathbf{q}}^d(t)$ is also bounded, the filtered error $\mathbf{r} \in \mathbb{R}^{\mathrm{N}\times 1}$ can be defined as

$$\mathbf{r}(t) = \dot{\mathbf{e}}(t) + \boldsymbol{\Lambda}\mathbf{e}(t) \tag{7.2}$$

where the tracking error $\mathbf{e}(t) \in \mathbb{R}^{\mathrm{N}\times 1}$ is defined as

$$\mathbf{e}(t) = \mathbf{q}_d(t) - \mathbf{q}(t) \tag{7.3}$$

and $\boldsymbol{\Lambda} = \boldsymbol{\Lambda}^T > 0$ is a positive definite design parameter matrix.

The manipulator dynamics given in Eq. 7.1 can be expressed as a function of the filtered error in Eq. 7.2 as

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{r}} = -\mathbf{V}(\mathbf{q},\dot{\mathbf{q}})\mathbf{r} + \mathbf{M}(\mathbf{q})\mathbf{r}_a + \mathbf{V}(\mathbf{q},\dot{\mathbf{q}})\mathbf{r}_b + \mathbf{G}(\mathbf{q}) - \boldsymbol{\tau} - \boldsymbol{\tau}_E \tag{7.4}$$

where $\mathbf{r}_a = \ddot{\mathbf{q}}_d + \boldsymbol{\Lambda}\dot{\mathbf{e}}$ and $\mathbf{r}_b = \dot{\mathbf{q}}_d + \boldsymbol{\Lambda}\mathbf{e}$. The dynamics in Eq. 7.4 can be expressed in a compact form as

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{r}} = -\mathbf{V}(\mathbf{q},\dot{\mathbf{q}})\mathbf{r} + \mathbf{f}(\mathbf{x}) - \boldsymbol{\tau} - \boldsymbol{\tau}_E \tag{7.5}$$

where $\mathbf{x} \in \mathbb{R}^{5\mathrm{N}\times 1} = \begin{bmatrix}\mathbf{e}^T & \dot{\mathbf{e}}^T & \mathbf{q}_d{}^T & \dot{\mathbf{q}}_d^T & \ddot{\mathbf{q}}_d^T\end{bmatrix}$ and $\{\mathbf{f}(.) : \mathbb{R}^{5\mathrm{N}\times 1} \to \mathbb{R}^{\mathrm{N}\times 1}\}$ is defined as

$$\mathbf{f}(\mathbf{x}) = \mathbf{M}(\mathbf{q})\mathbf{r}_a + \mathbf{V}(\mathbf{q},\dot{\mathbf{q}})\mathbf{r}_b + \mathbf{G}(\mathbf{q}) \tag{7.6}$$

The matrices in Eq. 7.6 that are functions of $\mathbf{q}$ and $\dot{\mathbf{q}}$, this can be abbreviated with a subscript m and Eq. 7.6 can be rewritten as

$$\mathbf{f}(\mathbf{x}) = \mathbf{M}_m\mathbf{r}_a + \mathbf{V}_m\mathbf{r}_b + \mathbf{G}_m \tag{7.7}$$

The universal approximation property of FFNN can be used to approximate the nonlinear function $\mathbf{f}(\mathbf{x})$ [17]. A two-layer FFNN can be used for this purpose and can be expressed mathematically as

$$\mathbf{f}(\mathbf{x}) = \mathbf{W}^T \sigma\left(\mathbf{V}^T \mathbf{x}\right) + \varepsilon \tag{7.8}$$

where $\sigma(.)$ is a set of basis functions, for the purpose of this implementation these functions are chosen to be sigmoid functions whose higher-order terms in a Taylor series are assumed to be bounded. The approximation error ε is assumed bounded on a compact set with a known bound ε_N, i.e. $\|\varepsilon\| < \varepsilon_N$. $\mathbf{V} \in \mathrm{I\!R}^{5\mathrm{N}\times\mathrm{L}}$ and $\mathbf{W} \in \mathrm{I\!R}^{\mathrm{L}\times\mathrm{N}}$ are ideal weights for the first and second layer of the FFNN respectively, that produce the exact approximation of $\mathbf{f}(\mathbf{x})$, but they are unknown. L is the number of neurons in the first layer.

Unless the network is minimal the ideal weights may not be unique but the best weights can be defined as those which minimize the supremum norm of ε over a compact simply connected set. Define the matrix $\mathbf{Z} \in \mathrm{I\!R}^{(5\mathrm{N}+\mathrm{L})\times(\mathrm{N}+\mathrm{L})}$ of both weight matrices $\mathbf{V}$ and $\mathbf{W}$ as follows

$$\mathbf{Z} = \begin{bmatrix} \mathbf{W} & \mathbf{0} \\ \mathbf{0} & \mathbf{V} \end{bmatrix} \tag{7.9}$$

The ideal weights $\mathbf{V}$ and $\mathbf{W}$ are assumed to be bounded through $\|\mathbf{Z}\|_F \leq Z_b$. Where there $\|.\|_F$ is the Frobenius norm operator. $Z_b \in \mathrm{I\!R}$ is assumed to be known and is selected by the designer based on a trial and error experiment.

The estimate of the nonlinear function $\mathbf{f}(\mathbf{x})$ is now given by

$$\hat{\mathbf{f}}(\mathbf{x}) = \hat{\mathbf{W}}^T \sigma\left(\hat{\mathbf{V}}^T \mathbf{x}\right) \tag{7.10}$$

where $\hat{\mathbf{W}} \in \mathrm{I\!R}^{\mathrm{L}\times\mathrm{N}}$ and $\hat{\mathbf{V}} \in \mathrm{I\!R}^{5\mathrm{N}\times\mathrm{L}}$ are estimated values of the FFNN weights for which update laws are derived.

7.2 Control Law and Error Dynamics

Using the universal approximation property of the FFNN [17], the nonlinear function $\mathbf{f}(\mathbf{x})$ can be approximated as given in Eq. 7.10, and the following control law can be stated

$$\boldsymbol{\tau} = \hat{\mathbf{W}}^T \sigma\left(\hat{\mathbf{V}}^T \mathbf{x}\right) + \mathbf{K}_v \mathbf{r} - \boldsymbol{\nu} \tag{7.11}$$

where $\mathbf{K}_v = \mathbf{K}_v^T > 0$ is a positive definite gain matrix, and $\boldsymbol{\nu} \in \mathbb{R}^{\mathrm{N}\times 1}$ function that allows for robustness in the face of higher-order terms in a Taylor series expansion that will be shown later. This robustifying term can be expressed as

$$\boldsymbol{\nu} = -\mathbf{K}_z \left(\|\hat{\mathbf{Z}}\|_F + Z_b\right) \mathbf{r} \tag{7.12}$$

Substituting the control law from Eq. 7.11 into the filtered error dynamics in Eq. 7.5 and using Eq. 7.8 for the ideal approximation of the nonlinear function $\mathbf{f}(\mathbf{x})$ gives the closed-loop filtered error dynamics as

$$\mathbf{M}_m \dot{\mathbf{r}} = -\left(\mathbf{V}_m + \mathbf{K}_v\right)\mathbf{r} + \mathbf{W}^T \sigma\left(\mathbf{V}^T \mathbf{x}\right) - \hat{\mathbf{W}}^T \sigma\left(\hat{\mathbf{V}}^T \mathbf{x}\right) + \boldsymbol{\nu} + \varepsilon - \boldsymbol{\tau}_E \tag{7.13}$$

by adding and subtracting $\mathbf{W}^T \sigma\left(\mathbf{V}^T \mathbf{x}\right)$ from the right hand side

$$\mathbf{M}_m \dot{\mathbf{r}} = -\left(\mathbf{V}_m + \mathbf{K}_v\right)\mathbf{r} + \tilde{\mathbf{W}}^T \hat{\sigma} - \mathbf{W}^T \tilde{\sigma} + \boldsymbol{\nu} + \varepsilon - \boldsymbol{\tau}_E \tag{7.14}$$

where $\hat{\sigma} = \sigma\left(\hat{\mathbf{V}}^T \mathbf{x}\right)$, $\tilde{\sigma} = \sigma - \hat{\sigma}$, $\sigma = \sigma\left(\mathbf{V}^T \mathbf{x}\right)$, and $\tilde{\mathbf{W}} = \mathbf{W} - \hat{\mathbf{W}}$. By further adding and subtracting $\hat{\mathbf{W}}^T \tilde{\sigma}$ from the right hand side

$$\mathbf{M}_m \dot{\mathbf{r}} = -\left(\mathbf{V}_m + \mathbf{K}_v\right)\mathbf{r} + \tilde{\mathbf{W}}^T \hat{\sigma} + \hat{\mathbf{W}}^T \tilde{\sigma} + \tilde{\mathbf{W}}^T \tilde{\sigma} + \boldsymbol{\nu} + \varepsilon - \boldsymbol{\tau}_E \tag{7.15}$$

Given that $\tilde{\sigma} = \sigma - \hat{\sigma} \equiv \sigma\left(\mathbf{V}^T \mathbf{x}\right) - \sigma\left(\hat{\mathbf{V}}^T \mathbf{x}\right)$, then Taylor series expansion for $\sigma\left(\mathbf{V}^T \mathbf{x}\right)$ can be expressed as

$$\sigma\left(\mathbf{V}^T \mathbf{x}\right) = \sigma\left(\hat{\mathbf{V}}^T \mathbf{x}\right) + \sigma'\left(\hat{\mathbf{V}}^T \mathbf{x}\right)\tilde{\mathbf{V}}^T \mathbf{x} + O\left(\tilde{\mathbf{V}}^T \mathbf{x}\right)^2 \tag{7.16}$$

where $\tilde{\mathbf{V}} = \mathbf{V} - \hat{\mathbf{V}}$ and

$$\sigma'\left(\hat{\mathbf{V}}^T\mathbf{x}\right) = \frac{\mathrm{d}\sigma\left(\mathbf{V}^T\mathbf{x}\right)}{\mathrm{d}\left(\mathbf{V}^T\mathbf{x}\right)}|_{\left(\mathbf{V}^T\mathbf{x}\right)=\left(\hat{\mathbf{V}}^T\mathbf{x}\right)} \tag{7.17}$$

and $O\left(\tilde{\mathbf{V}}^T\mathbf{x}\right)^2$ denotes terms of order 2. By denoting $\hat{\sigma}' = \sigma'\left(\hat{\mathbf{V}}^T\mathbf{x}\right)$ one can express $\tilde{\sigma}$ as

$$\tilde{\sigma}\left(\mathbf{V}^T\mathbf{x}\right) = \hat{\sigma}'\tilde{\mathbf{V}}^T\mathbf{x} + O\left(\tilde{\mathbf{V}}^T\mathbf{x}\right)^2 \tag{7.18}$$

The expression in Eq. 7.18 can be used in Eq. 7.15 to express $\tilde{\sigma}$ as a linear function of $\tilde{\mathbf{V}}$ plus higher order terms. This approximation gives the following closed loop filtered error dynamics

$$\mathbf{M}_m\dot{\mathbf{r}} = -\left(\mathbf{V}_m + \mathbf{K}_v\right)\mathbf{r} + \tilde{\mathbf{W}}^T\hat{\sigma} + \hat{\mathbf{W}}^T\hat{\sigma}'\tilde{\mathbf{V}}^T\mathbf{x} + \boldsymbol{\nu} + \mathbf{w}_1 \tag{7.19}$$

where $\mathbf{w}_1$ is a term that sums all the disturbance terms as follows

$$\mathbf{w}_1 = \tilde{\mathbf{W}}^T\hat{\sigma}'\tilde{\mathbf{V}}^T\mathbf{x} + \mathbf{W}^T O\left(\tilde{\mathbf{V}}^T\mathbf{x}\right)^2 + \varepsilon - \boldsymbol{\tau}_E \tag{7.20}$$

Using the error system in Eq. 7.19 does not yield a compact set outside which a certain Lyapunov function derivative is negative. The error system can be written as

$$\mathbf{M}_m\dot{\mathbf{r}} = -\left(\mathbf{V}_m + \mathbf{K}_v\right)\mathbf{r} + \tilde{\mathbf{W}}^T\left(\hat{\sigma} - \hat{\sigma}'\hat{\mathbf{V}}^T\mathbf{x}\right) + \hat{\mathbf{W}}^T\hat{\sigma}'\tilde{\mathbf{V}}^T\mathbf{x} + \boldsymbol{\nu} + \mathbf{w} \tag{7.21}$$

with

$$\mathbf{w} = \tilde{\mathbf{W}}^T\hat{\sigma}'\mathbf{V}^T\mathbf{x} + \mathbf{W}^T O\left(\tilde{\mathbf{V}}^T\mathbf{x}\right)^2 + \varepsilon - \boldsymbol{\tau}_E \tag{7.22}$$

It is noted here that the NN approximation error, the disturbance torque, and the Taylor series higher order terms all influence the error system in a similar manner, a disturbance. And the total disturbance in the error system can be shown to be bounded from above by [77]

$$\|\mathbf{w}\| \leq c_0 + c_1\|\tilde{\mathbf{Z}}\|_F + c_2\|\tilde{\mathbf{Z}}\|_F\|\mathbf{r}\| \tag{7.23}$$

where c_i are constants and $\tilde{\mathbf{Z}} = \mathbf{Z} - \hat{\mathbf{Z}}$.

7.3 FFNN Weight Update Laws

Now that the control law has been established, and the closed-loop system disturbances are shown to be bounded, the update laws for the weights in Eq. 7.11 can be derived starting from a candidate Lyapunov function given by

$$L = \frac{1}{2}\mathbf{r}^T\mathbf{M}_m\mathbf{r} + \frac{1}{2}\text{tr}\left(\tilde{\mathbf{W}}^T\mathbf{F}^{-1}\tilde{\mathbf{W}}\right) + \frac{1}{2}\text{tr}\left(\tilde{\mathbf{V}}^T\mathbf{G}^{-1}\tilde{\mathbf{V}}\right) \tag{7.24}$$

Given the desired trajectory $\mathbf{q}_d(t) \in C^2$; i.e $\mathbf{q}_d(t)$ is bounded and twice differentiable with respect to time and the control law given in Eq. 7.11, the FFNN weight update laws are derived such that the time derivative of L is negative definite, i,e $\dot{L} < 0$ outside a compact set. The update laws are summarized as follows

$$\dot{\hat{\mathbf{W}}} = \mathbf{F}\hat{\sigma}\mathbf{r}^T - \mathbf{F}\hat{\sigma}'\hat{\mathbf{V}}\mathbf{x}\mathbf{r}^T - \kappa\mathbf{F}\|\mathbf{r}\|\hat{\mathbf{W}} \tag{7.25}$$

$$\dot{\hat{\mathbf{V}}} = \mathbf{G}\mathbf{x}\left(\hat{\sigma}'^T\hat{\mathbf{W}}\mathbf{r}\right)^T - \kappa\mathbf{G}\|\mathbf{r}\|\hat{\mathbf{V}} \tag{7.26}$$

for any constant matrix $\mathbf{F} = \mathbf{F}^T > 0$, constant matrix $\mathbf{G} = \mathbf{G}^T > 0$, and scalar design parameter κ. Then the filtered error and the weight estimates are uniformly ultimately bounded for a large enough gain $\mathbf{K}_v$ [76, 77, 17, 78].

Proof [77]. Taking the time derivative of Eq. 7.24 and substituting the the dynamics from Eq. 7.21 gives

$$\begin{aligned}\dot{L} = &- \mathbf{r}^T\mathbf{K}_v\mathbf{r} + \frac{1}{2}\mathbf{r}^T\left(\dot{\mathbf{M}}_m - 2\mathbf{V}_m\right)\mathbf{r}... \\ &... + \text{tr}\,\tilde{\mathbf{W}}^T\left(\mathbf{F}^{-1}\dot{\tilde{\mathbf{W}}} + \hat{\sigma}\mathbf{r}^T - \hat{\sigma}'\hat{\mathbf{V}}^T\mathbf{x}\mathbf{r}^T\right)... \\ &... + \text{tr}\,\tilde{\mathbf{V}}^T\left(\mathbf{G}^{-1}\dot{\tilde{\mathbf{V}}} + \mathbf{x}\mathbf{r}^T\hat{\mathbf{W}}^T\hat{\sigma}'\right) + \mathbf{r}^T(\boldsymbol{\nu} + \mathbf{w})\end{aligned} \tag{7.27}$$

For robot dynamics it is known that the matrix $\dot{\mathbf{M}}_m - 2\mathbf{V}_m$ is skew symmetric [72], therefore the second term in Eq. 7.27 is zero. By substituting the update laws from Eqs. 7.25 and 7.26 into Eq. 7.27 one gets

$$\dot{L} = -\mathbf{r}^T\mathbf{K}_v\mathbf{r} + \kappa\|\mathbf{r}\|\text{tr}\,\tilde{\mathbf{W}}^T\left(\mathbf{W} - \hat{\mathbf{W}}\right) + \kappa\|\mathbf{r}\|\text{tr}\,\tilde{\mathbf{V}}^T\left(\mathbf{V} - \hat{\mathbf{V}}\right) + \mathbf{r}^T(\boldsymbol{\nu} + \mathbf{w}) \tag{7.28}$$

By using $\tilde{\mathbf{Z}}$, Eq. 7.28 can be written as

$$\dot{L} = -\mathbf{r}^T\mathbf{K}_v\mathbf{r} + \kappa\|\mathbf{r}\|\mathrm{tr}\ \tilde{\mathbf{Z}}^T\left(\mathbf{Z}-\hat{\mathbf{Z}}\right) + \mathbf{r}^T\left(\boldsymbol{\nu}+\mathbf{w}\right) \tag{7.29}$$

Since $\mathrm{tr}\ \tilde{\mathbf{Z}}^T\left(\mathbf{Z}-\hat{\mathbf{Z}}\right) = \langle\tilde{\mathbf{Z}},\mathbf{Z}\rangle_F - \|\tilde{\mathbf{Z}}\|_F^2 \leq \|\tilde{\mathbf{Z}}\|_F\|\mathbf{Z}\|_F - \|\tilde{\mathbf{Z}}\|_F^2$ then

$$\begin{aligned}\dot{L} \leq & -\sigma_{min}\left(\mathbf{K}_v\right)\|\mathbf{r}\|^2 + \kappa\|\mathbf{r}\|\|\tilde{\mathbf{Z}}\|_F\left(Z_b - \|\tilde{\mathbf{Z}}\|_F\right)\ldots\\ & \ldots - \mathbf{K}_z\left(\|\hat{\mathbf{Z}}\|_F + Z_b\right)\|\mathbf{r}\|^2 + \|\mathbf{r}\|\|\mathbf{w}\|\end{aligned} \tag{7.30}$$

were $\sigma_{min}\left(.\right)$ is the minimum singular value, and by using the disturbance bound in Eq. 7.23

$$\begin{aligned}\dot{L} \leq & -\sigma_{min}\left(\mathbf{K}_v\right)\|\mathbf{r}\|^2 + \kappa\|\mathbf{r}\|\|\tilde{\mathbf{Z}}\|_F\left(Z_b - \|\tilde{\mathbf{Z}}\|_F\right) - \mathbf{K}_z\left(\|\hat{\mathbf{Z}}\|_F + Z_b\right)\|\mathbf{r}\|^2\ldots\\ & \ldots + \|\mathbf{r}\|\left[c_0 + c_1\|\tilde{\mathbf{Z}}\|_F + c_2\|\tilde{\mathbf{Z}}\|_F\|\mathbf{r}\|\right]\end{aligned} \tag{7.31}$$

and therefore

$$\dot{L} \leq -\|\mathbf{r}\|\left[\sigma_{min}\left(\mathbf{K}_v\right)\|\mathbf{r}\| + \kappa\|\tilde{\mathbf{Z}}\|_F\left(\|\tilde{\mathbf{Z}}\|_F - Z_b\right) - c_0 - c_1\|\tilde{\mathbf{Z}}\|_F\right] \tag{7.32}$$

clearly $\dot{L}$ is negative when the term in braces is positive.

Define $C_3 = Z_b + C_1/\kappa$ and by completing the square

$$\begin{aligned}&\sigma_{min}\left(\mathbf{K}_v\right)\|\mathbf{r}\| + \kappa\|\tilde{\mathbf{Z}}\|_F\left(\|\tilde{\mathbf{Z}}\|_F - C_2\right) - C_0\\ &\qquad = \kappa\left(\|\tilde{\mathbf{Z}}\|_F - c_3/2\right)^2 - \kappa c_3^2/4 + \sigma_{min}\left(\mathbf{K}_v\right)\|\mathbf{r}\| - c_0\end{aligned} \tag{7.33}$$

this is guaranteed positive if

$$\|\mathbf{r}\| > \frac{\kappa c_3^2/4 + c_0}{\sigma_{min}\left(\mathbf{K}_v\right)} \tag{7.34}$$

or

$$\|\tilde{\mathbf{Z}}\|_F > c_3/2 + \sqrt{c_3^2/4 + c_0/\kappa} \tag{7.35}$$

This makes $\dot{L}$ negative outside a compact set. The result from Eq. 7.34 shows that the filtered error is uniformly ultimately bounded for a large enough gain $\mathbf{K}_v$, which shows that the tracking error $\mathbf{e}\left(t\right)$ also remains bounded for any trajectory that starts within the set.

7.4 Example - Three-Link Manipulator

To verify the capability of the DNN with intelligent feedback for operational space trajectory generation which is developed in chapter 4, to test the robustness of the M-ZNN derived in chapter 6 to singularities, and to verify the neuro-adaptive controller's ability to execute a task, a numerical simulation of the three-link manipulator in appendix A is conducted. The simulation is identical to the one used to test the APF planner in chapter 5, were it is desired to track a circular trajectory with a three-link manipulator end-effector in operational space.

7.4.1 Simulations

A numerical simulation is conducted using the same robot kinematics, dynamics, and initial conditions as used in the example in chapter 5. In this simulation however, a DNN with intelligent feedback is used to generate a desired trajectory for the manipulator end-effector in operational space. The trajectory to be tracked in operational space is a circular one with a diameter of 0.5 m and a frequency of 2π rad/sec. The M-ZNN is used to calculate the inverse of the Jacobian in a singularity robust manner while the neuro-adaptive controller detailed in this chapter is used to track the desired joint angles. The simulation runs for 50 seconds.

7.4.2 Results

The operational space trajectory planned and executed in this simulation are given in Fig. 7.2, a referce circle (blue triangles) of the intended trajectory is given in the figure for the sake of comparison. The results in this figure clearly show that when executing the exact same simulation as the one in chapter 5, the combination of the DNN with intelligent feedback, M-ZNN, and neuro-adaptive controller delivers satisfactory results. It is capable of generating and tracking the desired circular reference trajectory with the manipulator end-effector.

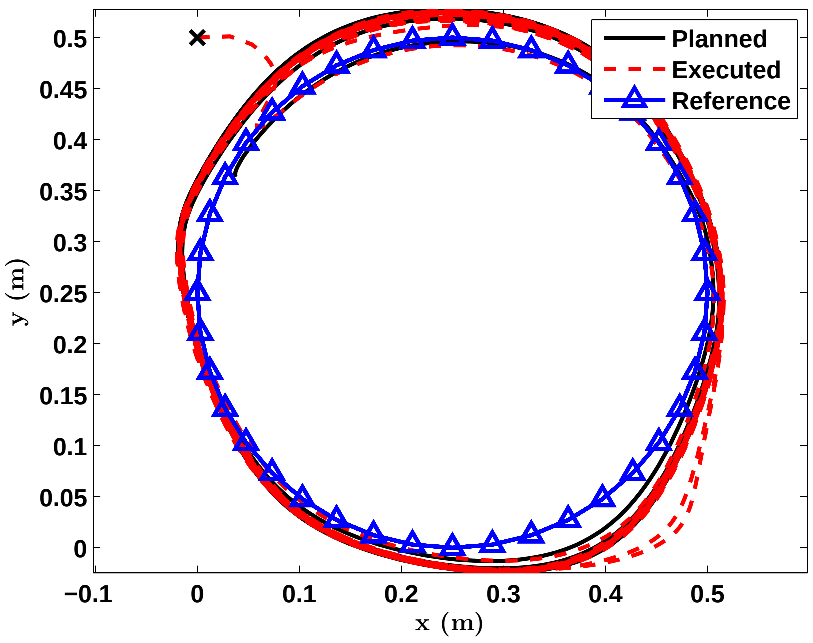

Figure 7.2. End-effector trajectories, desired and actual - MZNN.

The planned and executed joint trajectories for this simulation are given in Figs. 7.3, 7.4, and 7.5. Three main results can be seen in these figures; the first is that there are no abrupt changes in the value of the planned joint trajectories, i.e. the desired trajectories are smooth. These smooth trajectories are a result of the system being robust to singular configurations, which implies that the Jacobian is numerically well conditioned for inversion. This can also be seen in Fig. 7.6, were the Jacobian's condition number (C_n) is given for the simulation example given here as well as both simulations given in chapter 5; it is clear that when the no method is used to account for singularities (linear planner) the inverse of the Jacobian it is ill

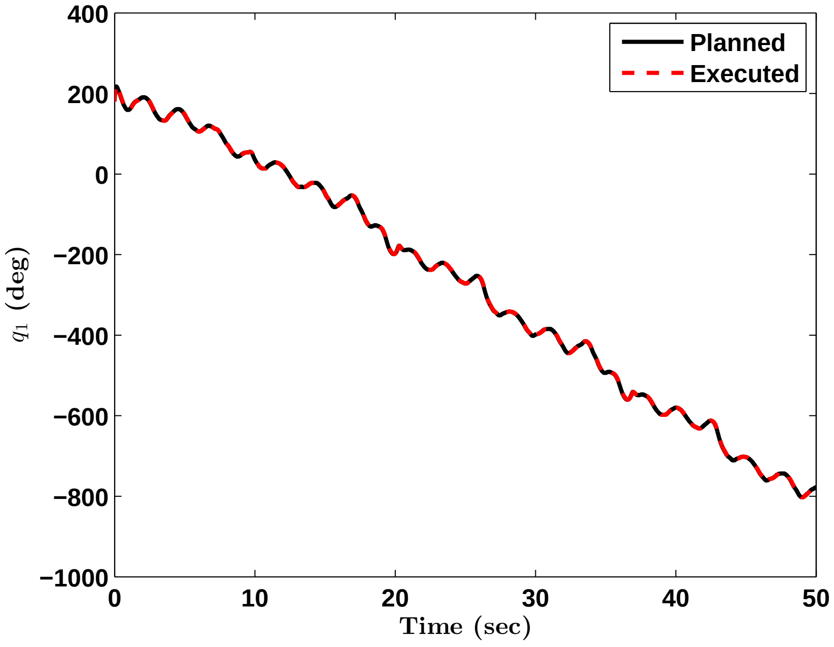

Figure 7.3. Joint one trajectories, desired and actual - MZNN.

conditioned (high condition number). Peaks of large C_n can be seen in the figure at the same time stamps at which large jumps in joint values are seen in Figs. 5.7, 5.8, and 5.9. However, when using the APF planner these jumps are avoided. The results from the simulation using an M-ZNN show that the Jacobian is very well conditioned for inversion, clearly from Fig. 7.6 it is the most well conditioned. This is due to the fact that the M-ZNN uses the DLS method to calculate the Jacobian inverse over the joint trajectory, and there makes the system robust to singularities.

The second result that can be seen in figures 7.4, and 7.5 is that a periodic pattern in the desired joint trajectories emerges after the first 10 seconds of the simulation. This can be explained by the adaptive nature of the intelligence used to generate these trajectories; the operational space trajectory generator (DNN) is adaptively intelligent, and due to the repeating nature of the circular trajectory, the

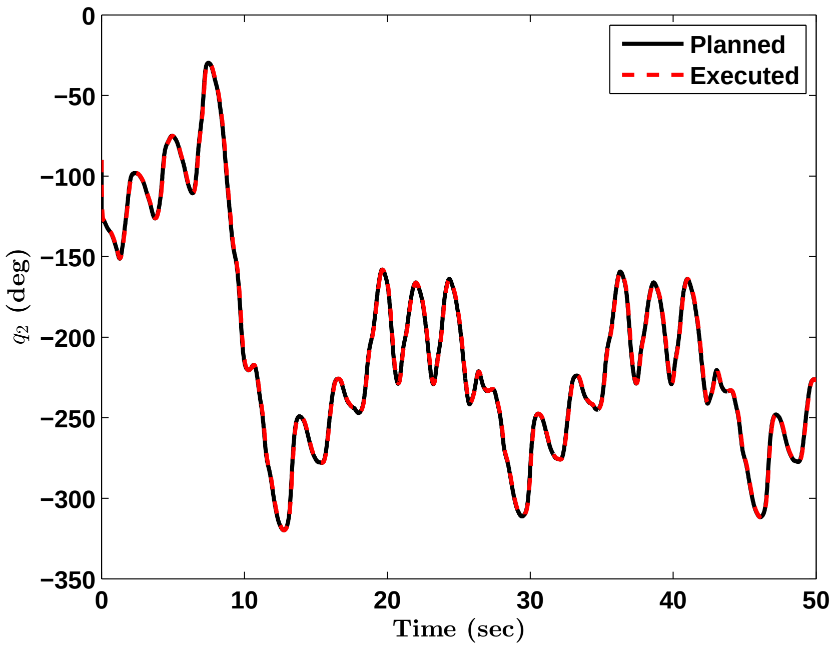

Figure 7.4. Joint two trajectories, desired and actual - MZNN.

system has converged to the same pattern that satisfies the circular trajectory. This behavior can be attributed to the system's intelligence where it has learned that no change in the desired trajectories is required.

The third and final result that can be seen in Figs. 7.3, 7.4, and 7.5, is the neuro-adaptive controller's ability to track the desired joint trajectories throughout the simulation. This results in the performance seen in Fig. 7.2 for tracking a circular trajectory in operational space.

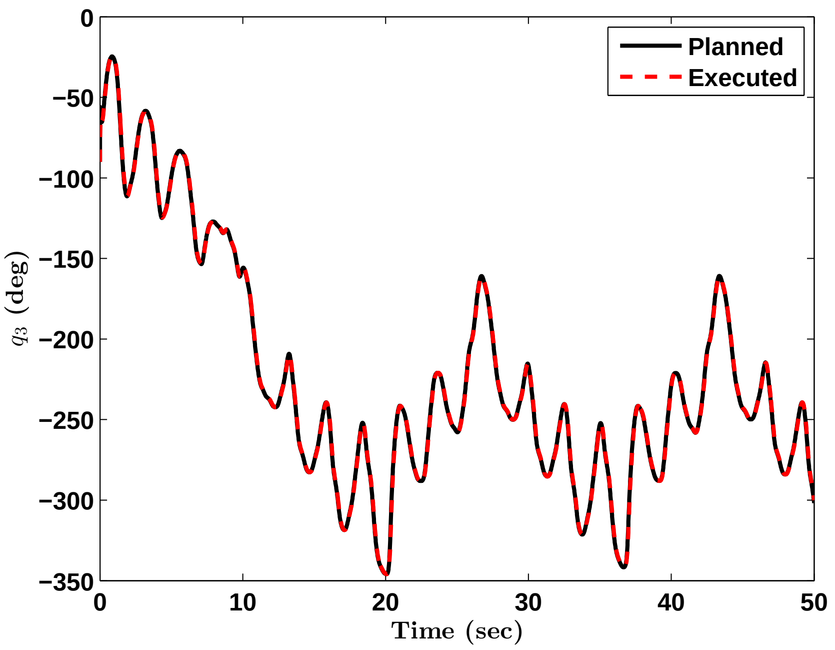

Figure 7.5. Joint three trajectories, desired and actual - MZNN.

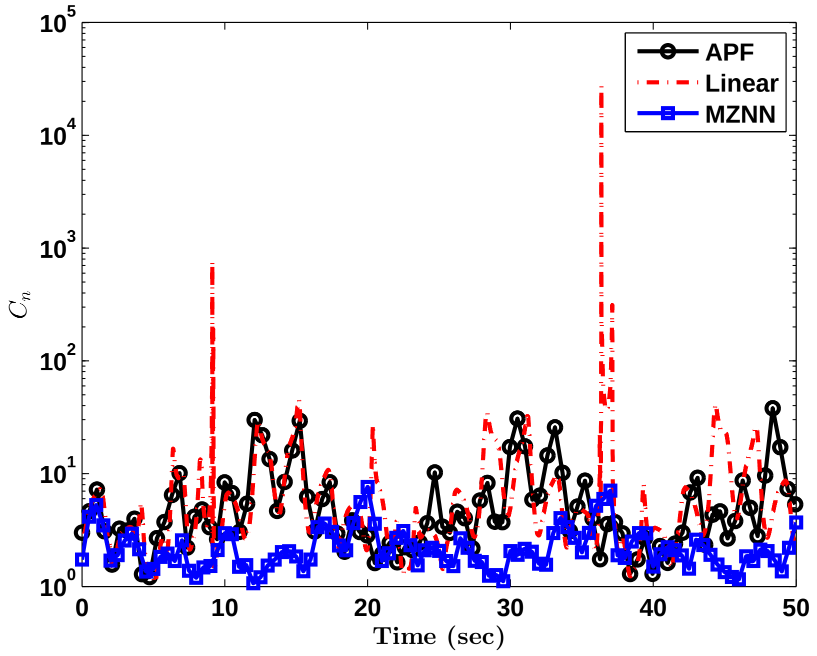

Figure 7.6. Jacobian matrix condition number.

CHAPTER 8

Humanoid Robot Planning and Control In Simulation

While the three-link manipulator arm serves as a good example to verify the methods developed in this dissertation, a more realistic simulation is considered to demonstrate the use of the aforementioned methods in a complex robotic application. This chapter deals with testing these methods on the Atlas humanoid robot, which is a complex electro-mechanical system with 28 joints.

For applications such as the one in this chapter, the APF method for singularity free joint trajectory generation developed in chapter 5 cannot be utilized. We use the singularity robust solution as opposed to the singularity free solution derived previously. In this chapter, the DNN, the M-ZNN, and the neuro-adaptive control system derived previously, are used for solving the planning and control problem for the Atlas humanoid robot.

8.1 Atlas humanoid robot

The DARPA Robotics Challenge (DRC) is a competition of robot systems and software to develop robots capable of assisting humans in responding to natural and man-made disasters. Participating teams collaborating and innovating on a very short time line developed the hardware, software, sensors, and interfaces that enable their robots to complete a series of challenges defined by DARPA [80].

Atlas is a bipedal humanoid robot developed by Boston Dynamics [81], with funding and oversight from DARPA. It is 1.8 m (6 ft) in length and includes 28 hydraulically-actuated degrees of freedom and a mass of approximately 150 kg (330 lb). A simulation model of Atlas along with the actual hardware are shown in Fig. 8.1. Details of the Atlas robot are given in appendix B.

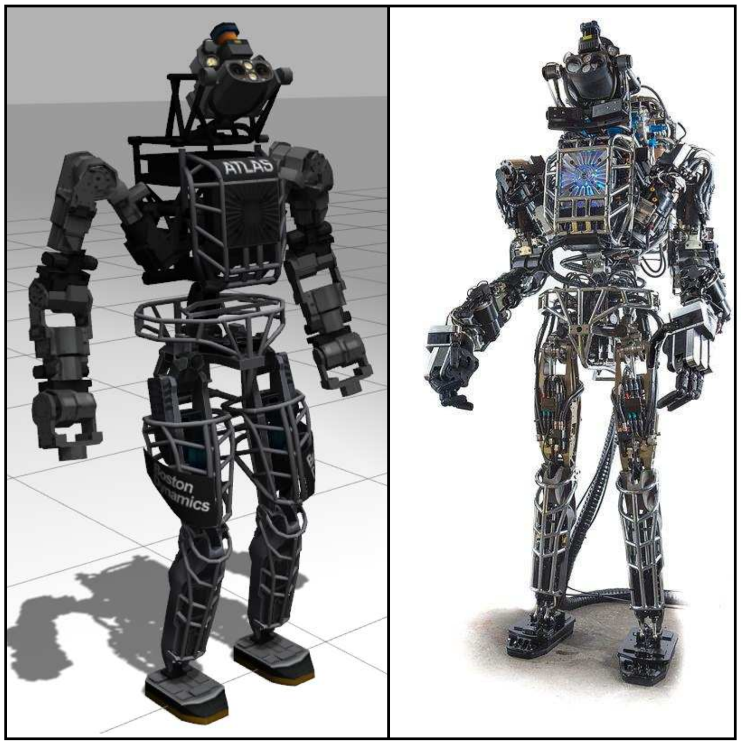

Figure 8.1. Atlas humanoid robot, simulation model and actual hardware.

8.2 Simulation

The open source three-dimensional robotics simulator, Gazebo [82], is used to simulate the kinematics and dynamics of the Atlas robot. Gazebo is a high-fidelity simulator developed and maintained by the Open Source Robotics Foundation (OSRF). It allows for highly customizable robot and environment models. The simulator is intended to allow for an easy transition from simulation to hardware implementation [82].

Gazebo operates in a server-client architecture; the physics engine and sensor data generator operate at the server level, a visualization and user interface client connects to that server and allows for user interaction. It allows for socket communication to pass messages and a variety of plugins to control any aspect of the simulation. The server runs in a loop to update the physics and sensor data while the client can send commands or receive (visualize) the results. The Gazebo architecture is shown in Fig. 8.2.

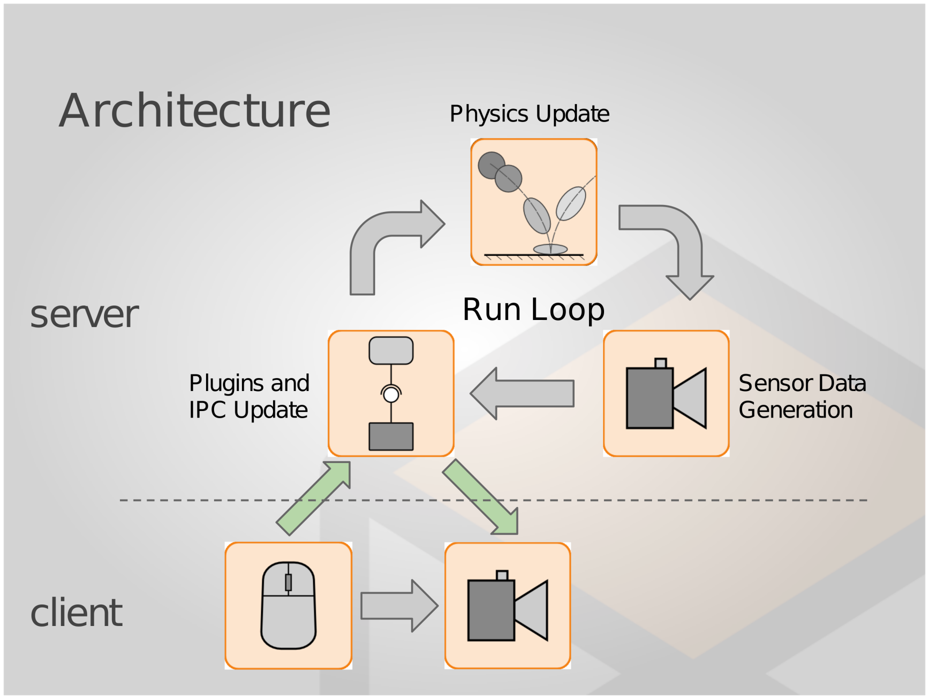

Figure 8.2. Gazebo server-client architecture [83].

Due to the open source and customizable nature of Gazebo, it allows for modeling the surrounding environment to fit any simulation scenario. For simple scenarios such as a manipulation task an environment can be a simple flat surface with objects on top. For indoor navigation a more complicated office space like environment can be used to test path planning and navigation algorithms. Outdoor environments can also be modeled easily since Gazebo is flexible and allows for using custom CAD models.

Robot models are also highly customizable in Gazebo, by defining a collection of links, joints, sensors, actuators, and plugins any robot model can be represented. From simple platforms such as a differential drive robot to highly realistic humanoid robots with full sensor suites are possible. The dynamics of the robot are simulated by solving the Newton-Euler equations with the assumption of frictionless joints and perfectly inelastic contact between objects. A variety of sensors can be defined in Gazebo such as cameras, laser range finders, force-torque sensors, joint encoders, and many more. To define a robot model in Gazebo a Universal Robot Description File (URDF) is required, a tutorial on how to do so is available on the Gazebo website [84].

During the first phase of the DRC, teams competed in a virtual simulation environment. The OSRF was selected to develop the the DRC simulator for the Atlas humanoid robot. The DRC simulator is built on Gazebo and simulates the kinematics and dynamics of the Atlas humanoid robot along with the entire sensor suite that is available on the actual robot. This simulation is used here in this research to verify the capabilities of the neuro-dynamic planning and control framework developed in earlier chapters. The environment of DRC simulator is shown in Fig. 8.3.

8.3 Software development in ROS

The Robot Operating System (ROS) is a flexible framework for developing robot software. It combines a set of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across all robotic platforms [85]. This is done through the following core components

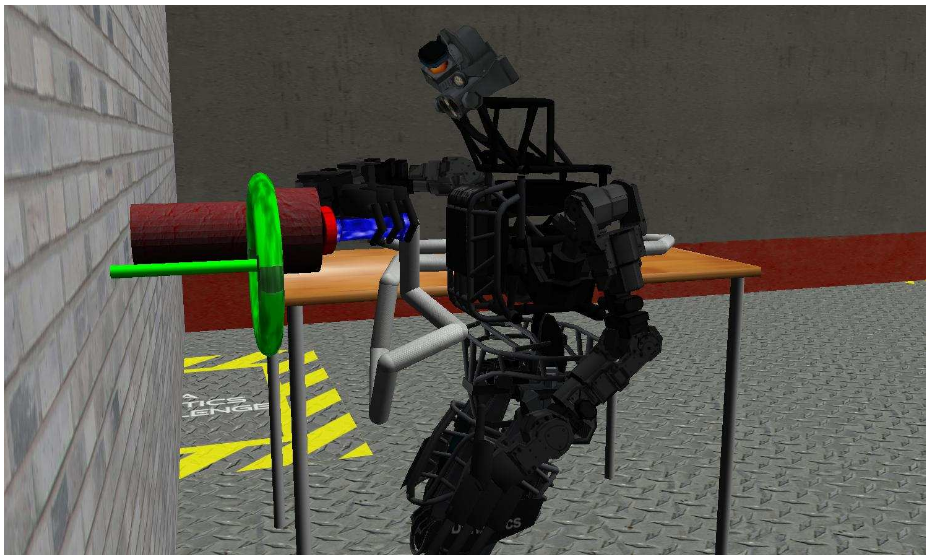

Figure 8.3. The DRC simulator environment.

1. A communications infrastructure
2. Robot-specific features
3. Tools to aid in operation

The communication infrastructure acts as a message passing interface for inter-process communication and is called the middleware. This middleware has a messaging system that manages the communication between distributed nodes; these nodes execute tasks required to operate the robot and either publish or subscribe to these messages over topics or services. The publishing/subscription of a message is asynchronous in nature and managed by a master node, Fig. 8.4 shows a simplified block diagram depiction of the communication infrastructure. These messages can be recorded for analysis or playback of a specific result.

ROS provides a set of robot-specific libraries that allows for getting robots up and running in short amount of time. A set of standard robot messages are available in ROS, there are message definitions for geometric concepts like poses, transforms,

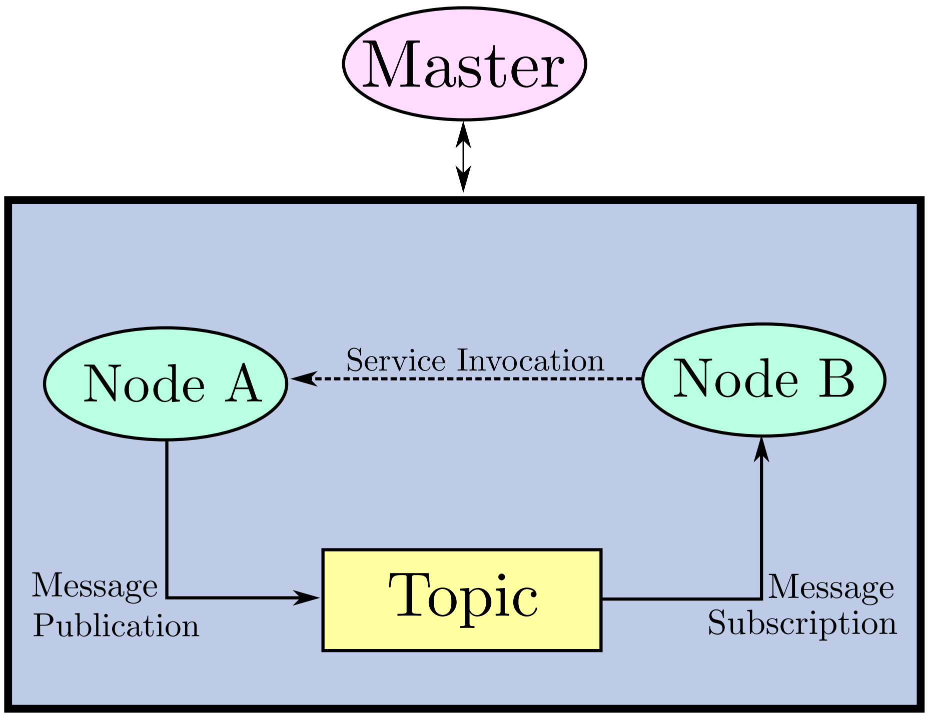

Figure 8.4. Block diagram depicting the ROS communication infrastructure.

and vectors; for sensors like cameras, IMUs and lasers; and for navigation data like odometry, paths, and maps, among many others.

A challenge in robotic applications is keeping track of different reference frames in multi-body robots, especially when sensor data is defined in one frame while used to execute a task in another. ROS provides a transform library that allows for keeping track of the robot geometry during operation. A wide variety of other robot-specific tools are available in ROS such as estimation, localization, and navigaton tools along with robot diagnostic tools. There are also a variety of tools that support introspecting, debugging, plotting, and visualizing the state of the system being developed.

An appealing feature in ROS is its ability to seamlessly integrate with other open source projects, most importantly Gazebo. The three-dimensional robot simulator can be run as a node inside ROS, and the ROS communication infrastructure is used to acquire sensor data from the robot simulation in Gazebo and send commands based on the user's request. This capability allows for robot software development and testing in a virtual simulation environment. And due to the interaction between ROS and Gazebo, the migration from simulation to hardware implementation is made simple provided that the hardware is built to provide the same information Gazebo does. Which is usually the case due to the realistic nature of simulations in Gazebo. A block diagram of two nodes running in ROS is shown in Fig. 8.5, Gazebo is running as a node that publishes robot sensor data while another node uses that data to generate control action that is sent to Gazebo to control the robot.

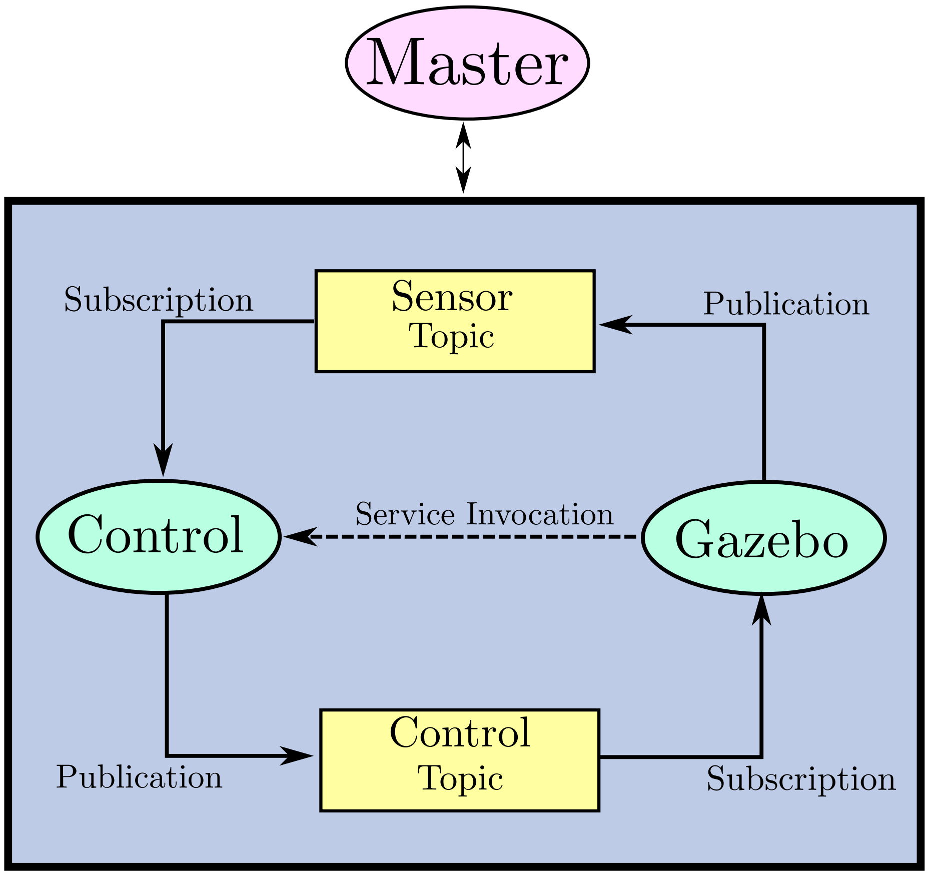

Figure 8.5. Block diagram depicting the use of ROS with Gazebo as a node.

8.4 Implementation on Atlas robot in simulation

To implement the neuro-dynamic framework for robotic planning and control on the Atlas humanoid robot in simulation, ROS and Gazebo are utilized. The DRC simulator is used for simulating the Atlas kinematics and dynamics along with the available sensors. Of interest in the implementation are the joint encoders that provide a measurement of the joint positions and velocities. In house software is developed using the programming language C++ to create the three libraries needed for implementation. The libraries are:

1. The Atlas kinematics library: deals with solving the forward kinematics for any point on the robot given the joint configuration. Also calculates the required Jacobian matrix.
2. The dynamic neural network (DNN) library: houses the design of a DNN used to generate desired operational space trajectories.
3. The Modified Zhang Neural Network (M-ZNN) library: uses a M-ZNN to calculate the damped-least-squares inverse of the Jacobian matrix and solve the IK problem numerically.

By using a set of ROS nodes the neuro-dynamic framework developed in earlier chapters is implemented on the Atlas robot in simulation. An operational space trajectory generator node that uses a DNN is used to generate required trajectories, while a joint space trajectory generator node solves the IK problem in a singularity robust fashion using the M-ZNN to generate desired joint trajectories. These desired trajectories are tracked by a neuro-adaptive control node. This implementation is graphically depicted in Fig. 8.6. The figure uses the same shapes and colors for ROS communication infrastructure as in Figs. 8.4 and 8.5, it however shows published messages in green lines and subscribed messages in red.

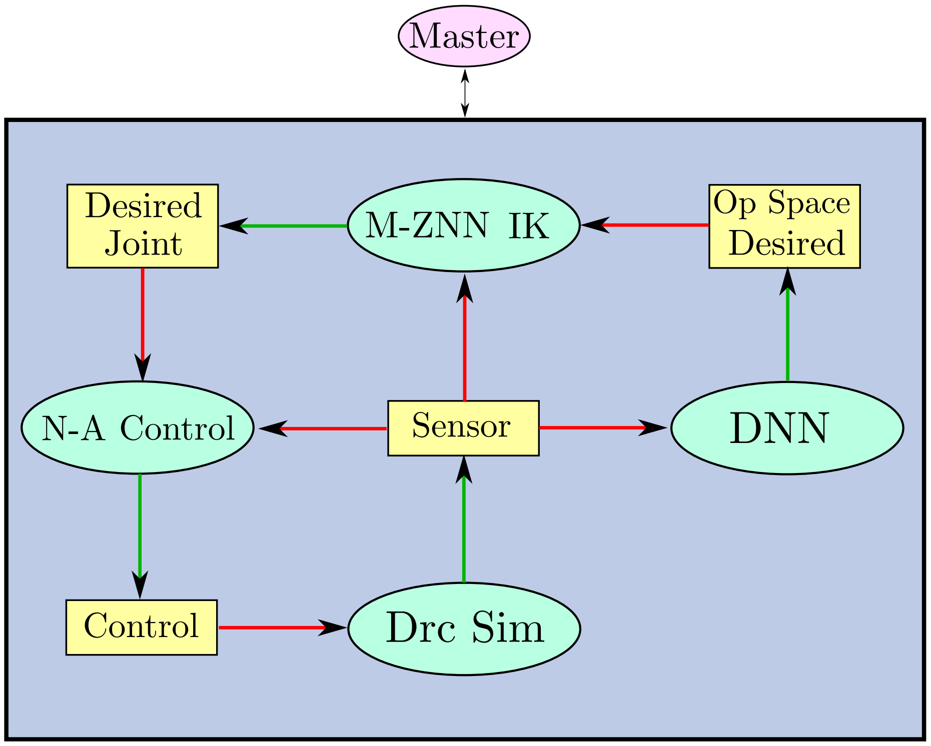

Figure 8.6. Block diagram depicting the implementation of neuro-dynamic methods in ROS - red lines are subscribed messages and green lines are published messages.

8.4.1 Practical considerations

To implement the neuro-dynamic methods developed in this dissertation on the Atlas robot it is required to compute several quantities and solve systems of ordinary differential equations. The realistic nature of the DRC simulator within the ROS framework, in terms of how the robotic system is implemented, dictates carrying out those computations in a practical manner. Here the problem of operational space path planning, IK solution, and control for the Atlas robot is outlined and the required practical solutions are detailed.

To control the motion of point or link (end-effector) on the Atlas robot, for example the right hand or left foot, it is required to define the pose of that point in a base frame. The kinematic structure of the Atlas, as presented in appendix B, contains 28 frames each attached to a link. In the work presented here all planning and control problems that are solved involve having at least one foot on the ground. This includes a multitude of tasks for a humanoid such as manipulation and walking. For this reason, the base frame is selected to be the foot in contact with the ground.

Using the defined base frame, the pose of the point or link to be controlled is calculated using the appropriate coordinate transformations. This is done using the in house Atlas kinematics library mentioned earlier. Figure 8.7 shows the kinematic structure of Atlas with the left foot as the base frame. Once the pose is defined, the Jacobian matrix can be calculated. Using the actual pose and desired pose along with the Jacobian matrix the IK problem can be then solved as detailed in chapter 2.

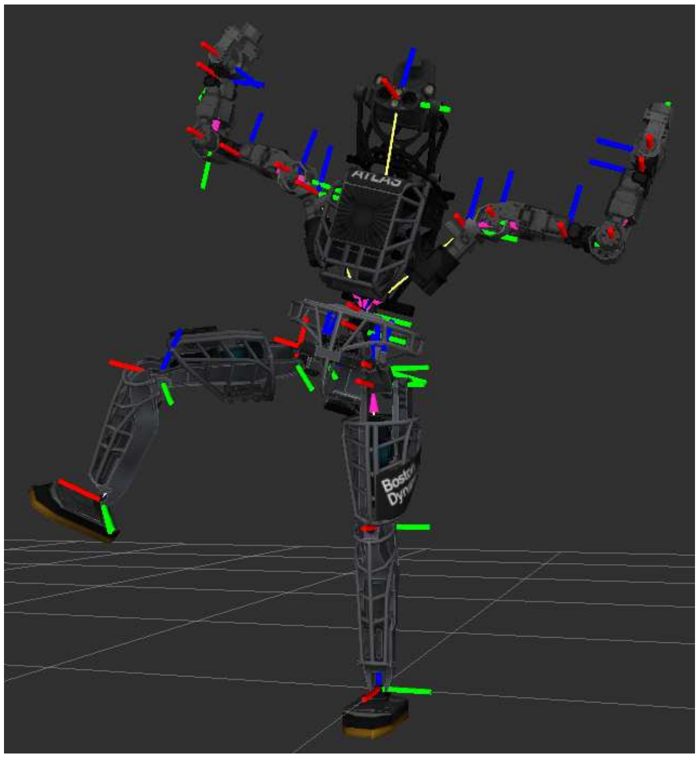

Figure 8.7. The Atlas robot kinematic structure with left foot as base link.

The points and links of interest in the applications presented here are the swing foot link, the upper torso link, and the center of mass (COM). The swing foot is the foot which is not selected to be the base link, it can be in contact with the ground or not depending on the application. For example if it is required to conduct a manipulation task both feet need to be on the ground to have better support. In this case the desired swing foot pose would be selected to keep it on the ground. For walking however, that requirement changes every time a step is to be taken. In Fig. 8.7 the swing foot is the right foot.

The pose of the upper torso is of interest since usually it is required to execute tasks while in an upright position, therefore the orientation of the torso is important. And finally the COM; to conduct any task using a humanoid it is important to maintain balance so that it does not fall. This can be done by controlling the location of the COM. As an example, for walking to be statically stable in the sense of ZMP [41], the COM is required to remain within the support polygon. This polygon is defined by convex hull of contact points with the ground and its size changes based on the number of feet on the ground and their position relative to each other. In the case when one foot is not in contact with the ground, the support polygon shrinks to the area of the sole of the foot in contact with the ground (base link).

Based on the points and links mentioned above, the operational space pose vector for the Atlas can be defined as $\mathbf{x}_e \in \mathbb{R}^{12} = \begin{bmatrix} \mathbf{x}_{com}^T & \mathbf{x}_{torso}^T & \mathbf{x}_{sf}^T \end{bmatrix}^T$, were

$$\mathbf{x}_{com} = [x_{com} \quad y_{com} \quad z_{com}]^T$$

$$\mathbf{x}_{torso} = [\phi_{torso} \quad \theta_{torso} \quad \psi_{torso}]^T$$

$$\mathbf{x}_{sf} = [x_{sf} \quad y_{sf} \quad z_{sf} \quad \phi_{sf} \quad \theta_{sf} \quad \psi_{sf}]^T$$

with x_i, y_i, z_i as the x, y, and z-positions of the i-th link or point. And ϕ_i, θ_i, ψ_i as the roll, pitch, and yaw angles of the i-th link. The subscript (sf) refers to swing foot.

It is important to note that this operational space pose vector is not unique, it is demonstrated here in detail since these links and points are considered important for controlling the motion of a humanoid while remaining stable. The vector can be changed by adding the pose of other links and points, one case would be adding either (or both) of the robot hands for tasks such as manipulation. The pose of the left hand for example can be augmented to the operational space pose vector in situations where controlling the motion of that link is required. This makes the planning and control solution more versatile by allowing different modes of operation, it however also introduces some practical issues when it comes to implementation; mainly calculating the Jacobian matrix. This is discussed next.

Numerical calculation of the Jacobian

The Jacobian matrix for the Atlas robot can be calculated as follows when the base frame and operational space pose vector are defined

$$\mathbf{J}(\mathbf{q}) = \frac{\partial \mathbf{x}_e(\mathbf{q})}{\partial \mathbf{q}} \tag{8.1}$$

this Jacobian $\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{\mathrm{M} \times \mathrm{N}}$ is such that (M) is the number of elements in $\mathbf{x}_e$, and (N) is equal to the number of joints which is 28. For the previously defined $\mathbf{x}_e$ the Jacobian matrix is given in Eq. 8.2.

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x_{com}}{\partial q_1} & \frac{\partial x_{com}}{\partial q_2} & \frac{\partial x_{com}}{\partial q_3} & \cdots & \cdots & \cdots & \frac{\partial x_{com}}{\partial q_{28}} \\ \frac{\partial y_{com}}{\partial q_1} & \frac{\partial y_{com}}{\partial q_2} & \frac{\partial y_{com}}{\partial q_3} & \cdots & & \cdots & \cdots \frac{\partial y_{com}}{\partial q_{28}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial \phi_{torso}}{\partial q_1} & \frac{\partial \phi_{torso}}{\partial q_2} & \frac{\partial \phi_{torso}}{\partial q_3} & \cdots & \cdots & \cdots & \frac{\partial \phi_{torso}}{\partial q_{28}} \\ \frac{\partial \theta_{torso}}{\partial q_1} & \frac{\partial \theta_{torso}}{\partial q_2} & \frac{\partial \theta_{torso}}{\partial q_3} & \cdots & \cdots & \cdots & \frac{\partial \theta_{torso}}{\partial q_{28}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial \theta_{sf}}{\partial q_1} & \frac{\partial \theta_{sf}}{\partial q_2} & \frac{\partial \theta_{sf}}{\partial q_3} & \cdots & \cdots & \cdots & \frac{\partial \theta_{sf}}{\partial q_{28}} \\ \frac{\partial \psi_{sf}}{\partial q_1} & \frac{\partial \psi_{sf}}{\partial q_2} & \frac{\partial \psi_{sf}}{\partial q_3} & \cdots & \cdots & \cdots & \frac{\partial \psi_{sf}}{\partial q_{28}} \end{bmatrix} \tag{8.2}$$

Given the size of the Jacobian matrix and the fact that $\mathbf{x}_e$ can be redefined based on the required task, deriving an analytical form for the Jacobian is not practical. Therefore, a numerical approach is adopted to calculate the required partial derivatives which constitute elements of the Jacobian matrix. By using a forward difference approximation the partial derivatives can be approximated as follows

$$\frac{\partial x_i\left(q_1, ..., q_i, ..., q_N\right)}{\partial q_i} \approx \frac{x_i\left(q_1, ..., q_i + \delta, ..., q_N\right) - x_i\left(q_1, ..., q_i, ..., q_N\right)}{\delta} \tag{8.3}$$

The perturbation δ can be selected to minimize the approximation error for the partial derivatives. Using this finite difference approach the Jacobian matrix for the Atlas can be numerically calculated given any operational pose vector.

Numerical calculation of the Jacobian time derivative

To solve the IK problem, the inverse of the Jacobian matrix is required. By using the neuro-dynamics of the M-ZNN derived in chapter 6 the Jacobian DLS inverse can be calculated. These neuro-dynamics require knowledge of the Jacobian time derivative. For the same reasons given for having a practical approach to calculating the Jacobian, a practical solution to calculate the time derivative of the Jacobian is required. The Jacobian time derivative can be defined using the chain rule as follows

$$\frac{\mathrm{d}\mathbf{J}}{\mathrm{d}t} = \frac{\partial \mathbf{J}}{\partial \mathbf{q}} \frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t} \tag{8.4}$$

where $\frac{\partial \mathbf{J}}{\partial \mathbf{q}}$ is a third order tensor, and $\frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t}$ is the joint velocities. To compute this tensor again the forward difference scheme is used where

$$\frac{\partial \mathbf{J}\left(q_1, ..., q_i, ..., q_N\right)}{\partial q_i} \approx \frac{\mathbf{J}\left(q_1, ..., q_i + \delta, ..., q_N\right) - \mathbf{J}\left(q_1, ..., q_i, ..., q_N\right)}{\delta} \tag{8.5}$$

This approach allows for numerically computing the time derivative of any Jacobian based on the defined operational space pose vector for Atlas. While the exact Jacobian Matrices can be computed using a symbolic engine, it is a significantly tedious task given the large number of degrees of freedom of the robot. Furthermore, when conducting a walking task the base frame will be switched to the foot on the ground each time a step is taken, this means that at least two Jacobian matrices need to be computed; one for each base frame. Also, the fact that for different operating modes a different operational space vector may be chosen, the definition of the Jacobian will change accordingly. This will also require having analytical expressions for every task related Jacobian. On the other hand, the practical approach presented here by means of numerically computing the Jacobian and its time derivative is easy to implement and the execution times are significantly shorter. The approximate Jacobian and its time derivative were verified for the three-link manipulator example by comparing it to the analytically derived value.

Numerical integration of the M-ZNN neuro-dynamics

To compute the DLS inverse of the Jacobian matrix the M-ZNN is used. The neuro-dynamics equations for the M-ZNN in Eq. 6.36 need to be solved to do that computation. The M-ZNN neuro-dynamics are a system of nonlinear ordinary differential equations, to solve this system a fourth order Runge-Kutta numerical integration scheme is used. In the left hand side of Eq. 6.36 the vector derivative is premultiplied by a matrix $\left(\mathcal{M}^T(t) \otimes \mathbf{I}\right)$ which is to be inverted before applying numerical integration to solve the system of differential equations. The size of this matrix is specified based on the number of elements in the state vector, which is dictated by the size of the operational space pose vector.

For example, when the operational space vector is chosen as detailed earlier for a swing foot, torso, and COM the size of that matrix $\left(\mathcal{M}^T(t) \otimes \mathbf{I}\right) \in \mathbb{R}^{336\times 336}$. Inverting such a large matrix requires vast computational resources and may not be completed in real time, i.e fast enough to compute the inverse and provide desired joint values for the controller on the robotic system. To overcome this problem the mixed-product property of the Kronecker product is used [86]. This property states that if two matrices $\mathbf{M}$ and $\mathbf{N}$ are non singular then

$$(\mathbf{M} \otimes \mathbf{N})^{-1} = \mathbf{M}^{-1} \otimes \mathbf{N}^{-1} \tag{8.6}$$

This is valid for the case of $\left(\mathcal{M}^T(t) \otimes \mathbf{I}\right)$ because $\mathcal{M}^T(t)$ is guaranteed to be nonsingular since the M-ZNN uses a DLS formulation, and $\mathbf{I}$ is the identity matrix. Therefore, rather than inverting $\left(\mathcal{M}^T(t) \otimes \mathbf{I}\right)$ only the inverse of $\mathcal{M}^T(t)$ needs to be calculated since the inverse of an identity matrix is itself. The size of $\mathcal{M}^T(t)$ is an order of magnitude less than that of $\left(\mathcal{M}^T(t) \otimes \mathbf{I}\right)$, therefore the inverse operation can be computed at a small computational overhead. The software developed for this purposes uses the C++ template library *Eigen* [87] for all linear algebra operations. This library contains an inverse functionality which uses the method of cofactors to invert a matrix of size 4×4 or less, and an LU decomposition of a matrix with partial pivoting for larger matrices [88].

8.4.2 System design

DNN design for trajectory generation

Controlling the Atlas humanoid to execute a task requires generating trajectories in operational space. This is done using a DNN with one RNN and multiple FFNN, one for each point or link that is required to move in a desired motion. For example during a walking task it is required to generate a walking gait that has trajectories for the COM and the feet. One RNN is used to generate a time varying trajectory with a frequency related to the speed of walking and one FFNN is used to generate the path for the COM while another for the swing foot. The torso orientation is kept upright at all times during walking.

The use of one RNN is possible since when conducting a task such as walking the motion of all links happen at the same time scale. The swing foot motion, the COM motion, and all other links motion happen at the same time scale; therefore, using one RNN with a frequency that is on the same scale reduces computation while allowing for the generation of the required trajectories. This is done by training multiple FFNNs to modulate the trajectory to a desired walking gait.

M-ZNN design for Jacobian inversion and IK solution

The IK problem is solved by numerically solving the system of differential equations in Eq. 2.9. A fourth order Runge-Kutta numerical integration scheme is used to compute the solution. The Jacobian inverse is computed by numerically solving the the M-ZNN neuro-dynamics equations in Eq. 6.36, where the parameters γ and λ are selected to have the values 2.1 and 0.01 respectively.

Table 8.1. FFNN structure for neuro-adaptive control of the Atlas

FFNN name	Hidden layer function	# hidden layer neurons	Output layer function
Left leg	Sigmoid	8	Linear
Right leg	Sigmoid	8	Linear
Left arm	Sigmoid	10	Linear
Right arm	Sigmoid	10	Linear
Body	Sigmoid	10	Linear

Neuro-adaptive control design

The neuro-adaptive control scheme discussed in chapter 7 is used to track the desired joint values that are calculated by the IK solver for Atlas. To implement the neuro-adaptive controller five different two-layer FFNN with online learning are utilized; one for each leg and arm as well as one for the body and head. A summary of these FFNN structure is given in table 8.1.

8.5 Results

8.5.1 Controlling the Atlas center-of-mass location

The first simulation to verify the capabilities of the DNN, M-ZNN, and neuro-adaptive controller is one where the position of the COM of controlled. The operational space pose vector is chosen to be $\mathbf{x}_e \in \mathbb{R}^{12} = \begin{bmatrix}\mathbf{x}_{com}^T & \mathbf{x}_{torso}^T & \mathbf{x}_{sf}^T\end{bmatrix}^T$, defined in the base foot which is selected to be the left foot. The desired pose for the right foot (swing foot) is at a fixed location on the ground next to the left foot with a zero relative orientation, i.e. $\mathbf{x}_{sf} = [0 \quad 0.27 \quad 0 \quad 0 \quad 0 \quad 0]^T$. This allows to have more support for testing when moving the COM and is a valid requirement since no locomotion (walking) is required. The desired torso orientation is $\mathbf{x}_{torso} = [0 \quad 0 \quad 0]^T$ to keep the body upright.

The largest range of motion that can be achieved for the COM when the Atlas is standing is along the z-axis which is the vertical direction. Therefore, in the simulation used for controlling the COM the desired trajectory for the COM is that where the x-position is kept at the value 0.043 m, and the y-position is kept at 0.135 m which is half the distance between the feet in the y-direction. For the z-position, it is desired to move the COM along a sinusoidal trajectory with an amplitude of 0.1 m and a mean value of 0.7 m. This would generate a trajectory with a maximum value of 0.8 m and a minimum value of 0.6 m. The required period for the sine wave is 12 seconds.

Using a DNN with an RNN that has the frequency of $\pi/6$ and a two-layer FFNN with 3 hidden layer neurons, the desired trajectory for the COM z-position can be generated. This is the desired trajectory in operational space. By solving the IK problem using the neuro-dynamics of the M-ZNN to compute the Jacobian inverse, the desired joint trajectories can be generated. And finally the neuro-adaptive controller can be used to track the desired joint trajectories. The simulation is run for 13 seconds in Gazebo and the results of which are given in Fig. 8.8.

The results given in Fig. 8.8 show a sequence of time stamped images from the simulation. In every sequence there are two images, the one on the left is that of the simulation and the one on the right is that of the ROS visualization tool RVIZ[1]. In the simulation image the Atlas robot configuration is seen, while in RVIZ a shadow of that configuration is recreated with a green sphere marker to indicate the actual location of the COM, while a blue cube markers indicates the desired position. The figure shows that the COM starts at the highest point (0.8 m) and moves downwards over a time of 6 seconds to the lowest position (0.6 m). It takes Atlas around 6 more seconds to bring the COM back up to the initial starting position. From these

[1]http://wiki.ros.org/rviz

results it is concluded that the planning and control system implemented on Atlas in simulation is capable of controlling the operational space position of the COM. The error in the tracking of the z-position is on the order of millimeters. The error in tracking the y position is less than the noise in the measurements therefore it is negligible. And the error in tracking the x-position is less than 0.015 m, i.e on the order of centimeters.

The results presented here verify the ability of neuro-dynamics framework in tracking a desired COM trajectory for the Atlas humanoid robot.

8.5.2 Walking

In order to verify the neuro-dynamic methods developed and outlined in this document a simulation experiment for executing a walking gait (trajectory) using the Atlas humanoid robot is conducted. The focus of the simulation experiment is not the design of the gait itself but rather executing it in a uncertainty and singularity robust manner. There is a wealth of literature available for generating walking gaits based in different measures such as stability, efficiency, and agility among others. The focus of the work presented here is that given a desired gait for a humanoid robot, the neuro-dynamic methods can be used to generate that gait and execute it in a reliable and robust manner.

Walking gait 1

By relying on the theory of ZMP and its stability criterion, as discussed in chapter 2, a statically stable gait can be defined as one where the COM of the humanoid remains within the confines of the support polygon. Therefore, it is safe to say that a conservative gait from a stability stand point is one where throughout the motion of the humanoid the COM is kept directly above the base foot when the system is in

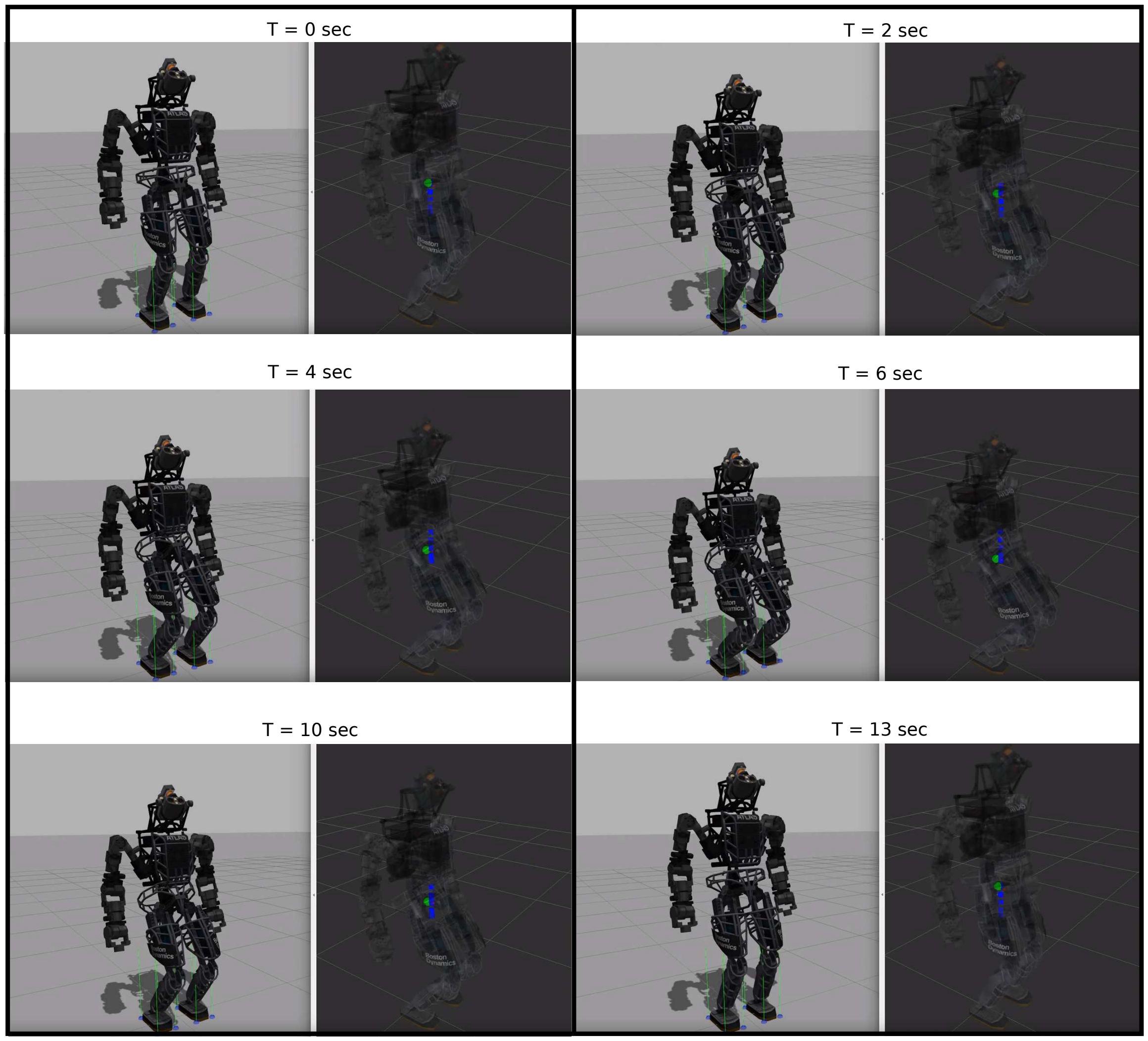

Figure 8.8. Tracking a sinusoidal trajectory for the Atlas COM.

single support (one foot on the ground), and the COM shift occurs when both feet are on the ground (double support). By cycling through such a gait, a stable mode of humanoid locomotion can be achieved.

The gait designed based on the logic presented above is shown in Figs. 8.9, 8.10, and 8.11. This gait is defined in a global frame where an observer can see the motion of the robot feet and COM in three dimensions. The x-position (forward direction) and z-position (upward direction) of the feet are shown in Figs 8.9 and

8.10 respectively; these positions are synchronized so that the foot is first lifted off the ground and then moved forward. One foot step is 35 cm in length and 10 cm in hight. The y-position is kept constant at a foot separation of 27 cm. This gait will allow the robot to move forward and take 6 steps every 140 seconds, which gives a period of the motion of approximately 23 seconds. This is a conservative gait in terms of stability therefore takes a relatively long time to execute.

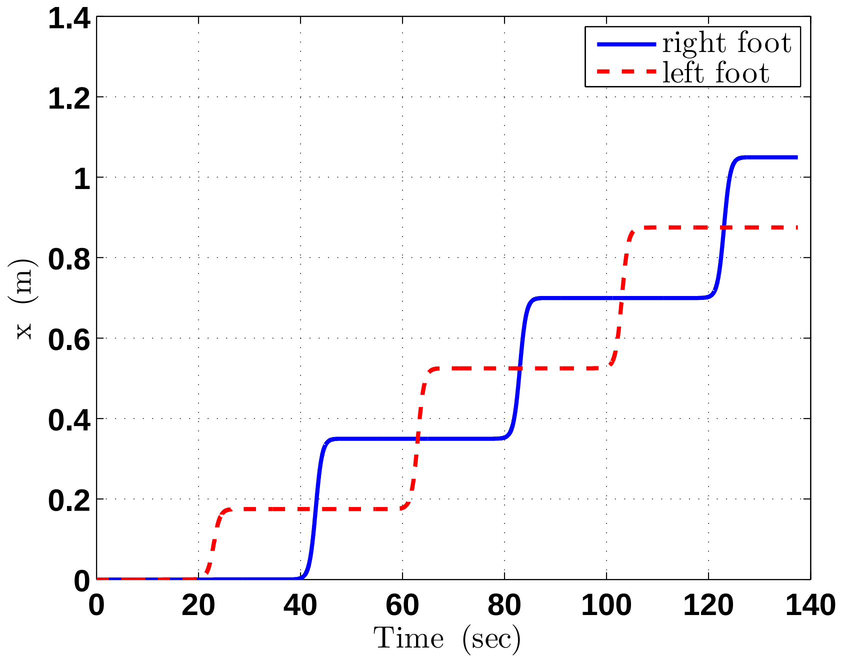

Figure 8.9. ZMP stable walking gait for Atlas robot - feet x-position.

The COM position in x and y-directions is given in Fig. 8.11, these trajectories are synchronized with those of the feet so that the COM shift occurs during the double support phase. The COM is shifted forward a distance equal to that of the step size, and sideways a distance equal to half the feet separation distance. This ensures the COM is always in the support polygon, i.e the gait is always stable.

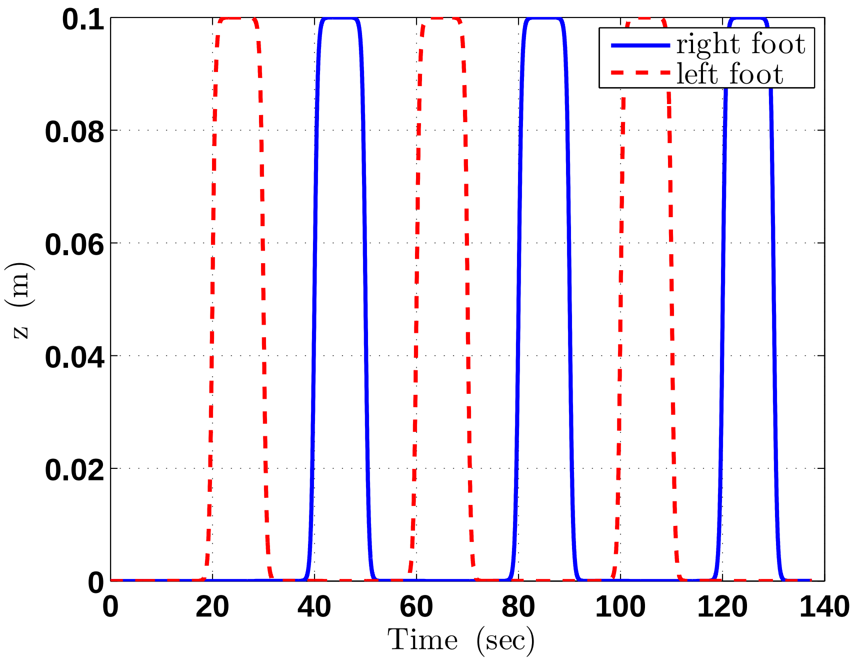

Figure 8.10. ZMP stable walking gait for Atlas robot - feet z-position.

To generate this gait a DNN that uses a RNN and three FFNN is used. The RNN is designed as outlined in chapter 4 and drives three FFNN that generate the left foot, right foot, and COM trajectories of the gait given above. For the feet trajectories two identical FFNN are used, each having three layers with 15 neurons in the first hidden layer and 10 in the second one. Both layers use a sigmoid activation function. For the COM trajectories a three-layer FFNN is used with 10 neurons in the first hidden layer and 15 in the second. The FFNN for the COM trajectories also uses sigmoid activation functions.

Walking gait 2

To further test the capabilities of the neuro-dynamic framework for robotic planning and control, a slightly different gait than the one discussed earlier is used.

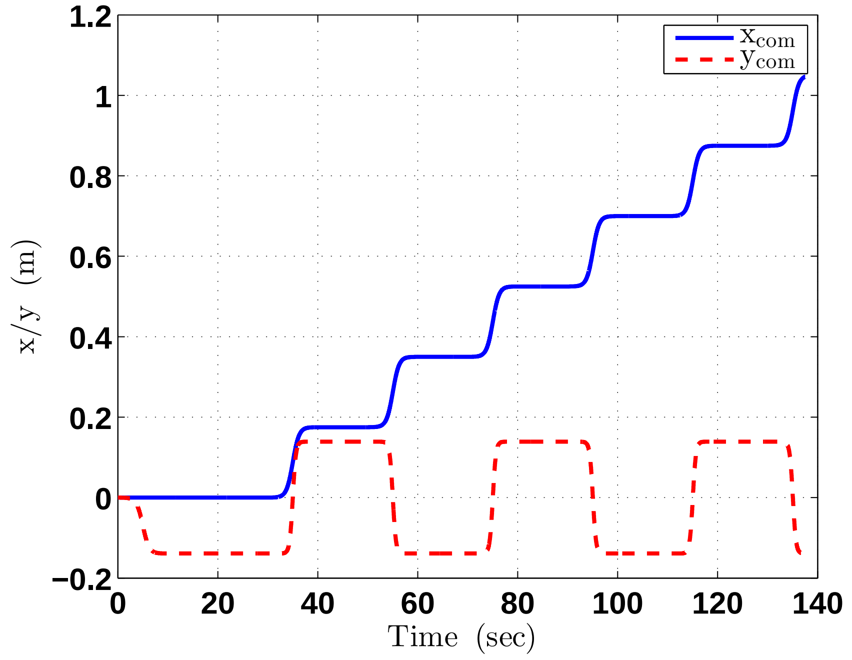

Figure 8.11. ZMP stable walking gait for Atlas robot - COM position.

This second walking gait uses the same exact feet and COM trajectories given in Figs. 8.9, 8.10, and 8.11. It also has the exact same foot separation distance and step size. The difference in this gait is that it requires swinging the robot's arms while moving; this is done by using the same swing foot trajectories in the x-direction given in Fig. 8.9. By tracking these trajectories with the humanoid hands a more natural looking gait can be executed. In this gait however, the swing hand is the one opposite to the current swing foot, i.e. if the current swing foot is the left one then the swing hand is the right hand.

Gait execution

The walking gaits for Atlas were designed from the point of view of an observer that would see the motion of Atlas as it walked by. The implementation on Atlas in simulation as explained earlier in the chapter does not model such an observer. This is based on the notion that the operational space pose vector and the Jacobian are defined in a base frame which is part of the robot structure. Therefore, appropriate coordinate transformations have to be used to account for this.

Since walking involves moving both feet (not at the same time), the base frame (foot) needs to be switched every time a step is taken. This is based on the idea that the base link has to be relatively fixed compared to the other links, and therefore the support foot is used. This support foot is switched every time a step is taken and the gait describe earlier is transformed to the base foot using a coordinate transformation. This is done using an in house kinematics library.

Walking simulation results - gait 1

Using the ROS/Gazebo architecture the Atlas robot is simulated and the DNN discussed above is used to generate a walking gait. The operational space pose vector is chosen to be $\mathbf{x}_e \in \mathbb{R}^{12} = \begin{bmatrix} \mathbf{x}_{com}^T & \mathbf{x}_{torso}^T & \mathbf{x}_{sf}^T \end{bmatrix}^T$. The Jacobian inverse is computed using the M-ZNN to solve the IK problem, and the desired joint trajectories are tracked using the neuro-adaptive controller. The results of that simulation are discussed below.

The COM trajectories are given in Fig. 8.12, the first conclusion is that the executed trajectories match the planned trajectories and this shows the neuro-dynamic framework's ability to execute the walking gait. It should be noted that the dashed green vertical lines in the figure represent the instants when the base frame switch occurs, and it can be clearly seen from the figure that the trajectories (both planned and executed) exhibit jumps at these instants. This is due to the fact that the trajectories are transformed from one base frame to another, and the jumps are only an artifact of that transformation.

Examining the top plot in Fig. 8.12 shows that the COM x-position is moved forward after every base link switch, and it is moved a distance of approximately 17 cm. This distance is half the step size or the x-distance between both feet when in double support. It is also clear that this motion is periodic with a period of approximately 23 seconds, matching that of the designed gait. So it can be concluded that during the walking gait execution the COM x-position is changed during the double support phase and moved to the new base foot.

A similar conclusion can be made regarding the y-position of the COM from the middle plot in Fig. 8.12; clearly in the beginning of the gait the COM is shifted to the base foot center, zero position in y, from a distance of approximately 13 cm which is equal to half the foot separation distance in the designed gait. Then at the first base foot switch a jump in the y-position equal to the foot separation can be seen, which is explained by the fact that the next base foot is now at that distance from the previous one where the COM is located. After the first switch the y-position of the COM is shifted to the zero position (next base foot). It can also be seen that this shift happens in the same time frame that the shift in the x-position occurs, i.e. during double support phase.

As for the z-position trajectory of the COM, they are shown in the bottom plot of Fig. 8.12. Initially the hight of the COM is lowered, this is done to gain more stability. Afterwards the COM hight is kept constant at 72 cm above the ground. At the switching instants there is a slight jump in the trajectory, this can be explained by examining the swing foot z-trajectories in the bottom plot of Fig. 8.13; it can be seen that the switch occurs at an instant were the swing foot is slightly above the ground (approximately 2 cm), therefore when the base frame is switched, the new swing foot (previous base foot) is actually below the new base foot since it is still on the ground. This is concluded from the negative z-position of the swing foot

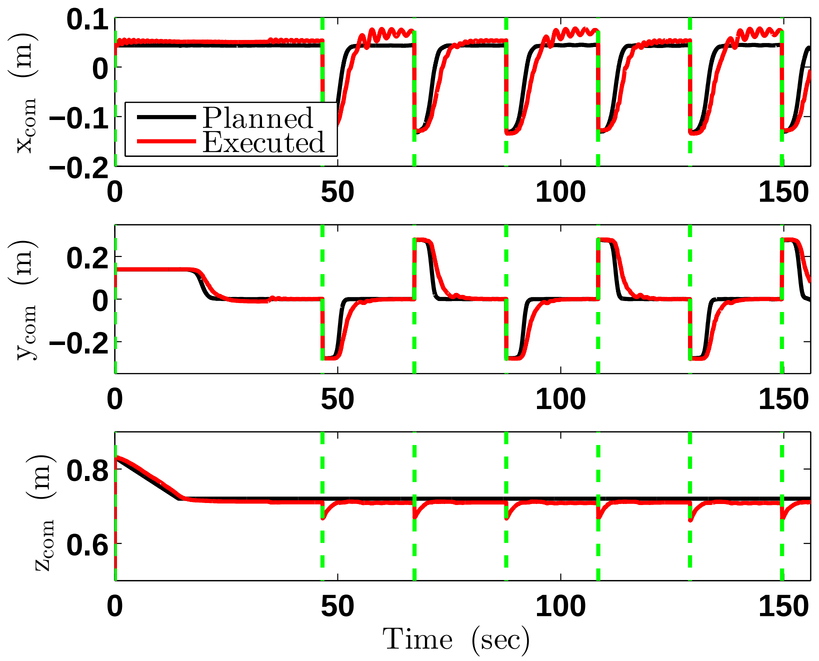

Figure 8.12. COM trajectories for Atlas robot - gait 1.

after every switch. However, since this is a switch that is handled by a coordinate transformation it does not affect the actual gait and that can be concluded from the fact that the swing foot location goes to a zero value after the switch in a smooth fashion. That implies that both feet are on the ground (double support). Using the nature of this switching, the jumps in the COM z-position observed in the bottom plot of Fig. 8.12 are explained; the hight of the COM is relative to the base foot, so when the switch occurs while the new base foot is a few centimeters off the ground, a small jump in the COM z-position is observed.

As a result of this switching, the shift in COM x and y positions are started before the swing foot is on the ground. Despite that the gait remains stable since the swing foot actually reaches the ground before the COM positions clear the plan-form area of the base foot. This also gives a more natural and synchronized walking gait.

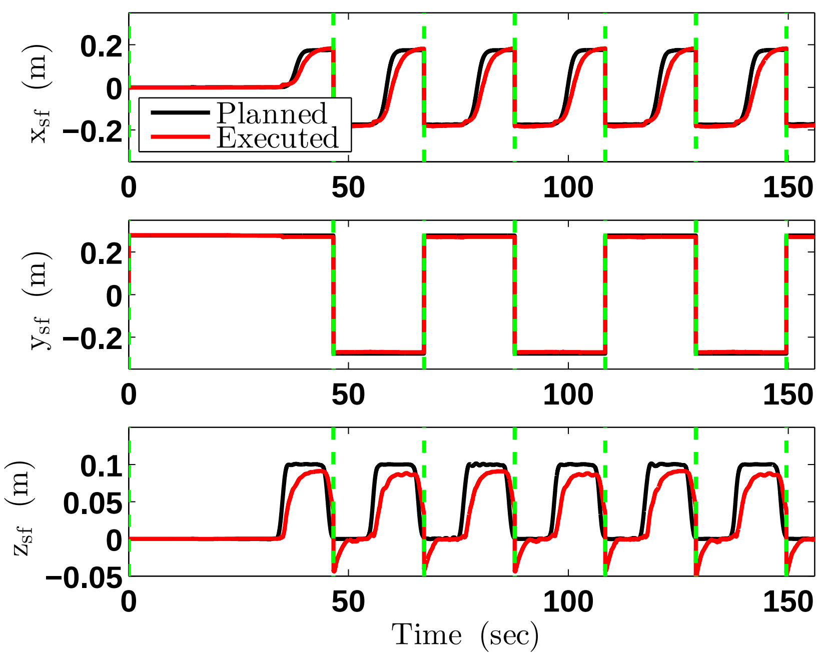

Figure 8.13. Swing foot position for Atlas robot - gait 1.

In the top plot of Fig. 8.13 the x-position of the swing foot can be clearly seen as periodic, starting with moving the swing foot forward 17 cm during the first step and 35 cm every step afterwards. This because the swing foot moves relative to the base foot, and at the start of the first step both feet are at the same x-position therefore, by moving the first swing foot by a value of half the step size it is guaranteed that all the next steps have the correct step size. The y-position of the swing foot remains constant during the walking since it is only a forward walking gait, again here the change in sign is due to the base frame switching and the fact that the motion is relative to the base frame.

The orientation of both the swing foot and the torso are shown in Figs. 8.14 and 8.15 respectively. Both orientations are required to be kept at a zero pitch, roll, and yaw relative to the base foot. This allows for walking while keeping the body upright and the swing foot leaving the floor and landing flat. It can be seen from

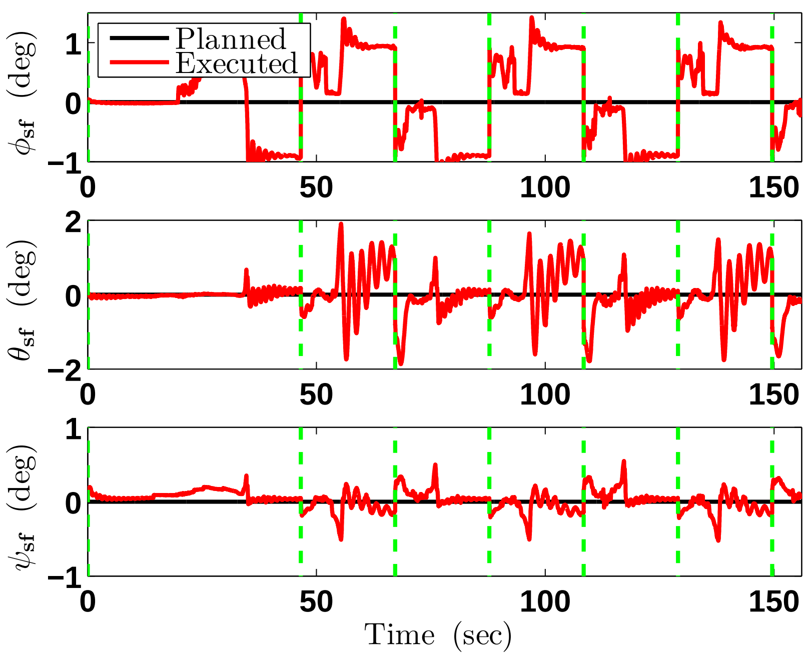

Figure 8.14. Swing foot orientation for Atlas robot - gait 1.

the results in both figures that the orientation error is less than 2 degrees for both. A time stamped sequence of images for the walking simulation test is shown in Fig. 8.16.

Walking simulation results - gait 2

The same exact simulation conditions used to execute gait 1 are used to execute gait 2. In this case as mentioned earlier the hands of Atlas are required to track a specific trajectory in the x-direction. Therefore, the operational space pose vector is chosen to be $\mathbf{x}_e \in \mathbb{R}^{14} = \begin{bmatrix} \mathbf{x}_{com}^T & \mathbf{x}_{torso}^T & \mathbf{x}_{sf}^T & x_{rh} & x_{lh} \end{bmatrix}^T$, where x_{rh} is the x-position of the right hand relative to the base foot, and x_{lh} is the x-position of the left hand relative to the base foot

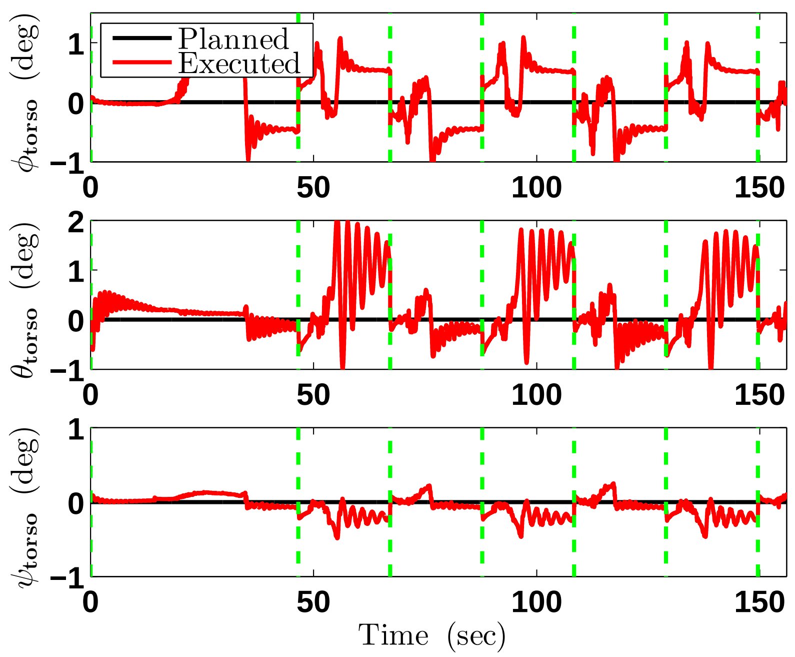

Figure 8.15. Torso orientation for Atlas robot - gait 1.

The results from the execution of gait 2 in simulation are given in Figs. 8.17 - 8.21; in all figures the dashed vertical green lines show the instants at which the base foot switch takes place. Here again it is seen that the executed trajectories of the COM, hands, feet, and torso are track the desired trajectories when executing this gait. The periodic nature of the gait is also seen in these results. The intial base foot in this gait is the right foot.

From the results in Fig. 8.17 the COM trajectories show the initial shift in the y-position (middle plot) to the center of the base (right) foot. This shift occurs after the COM z-position (bottom plot) is lowered to 72 cm, which is the desired constant hight from the base foot throughout the simulation. While tracking that constant hight for the COM the system also trackes the periodic shifts in the COM x-positon, as shown in the top plot, as well as the periodic shifts in the COM y-postion. It is seen that the COM is shifted forward and to the opposite foot before each base foot

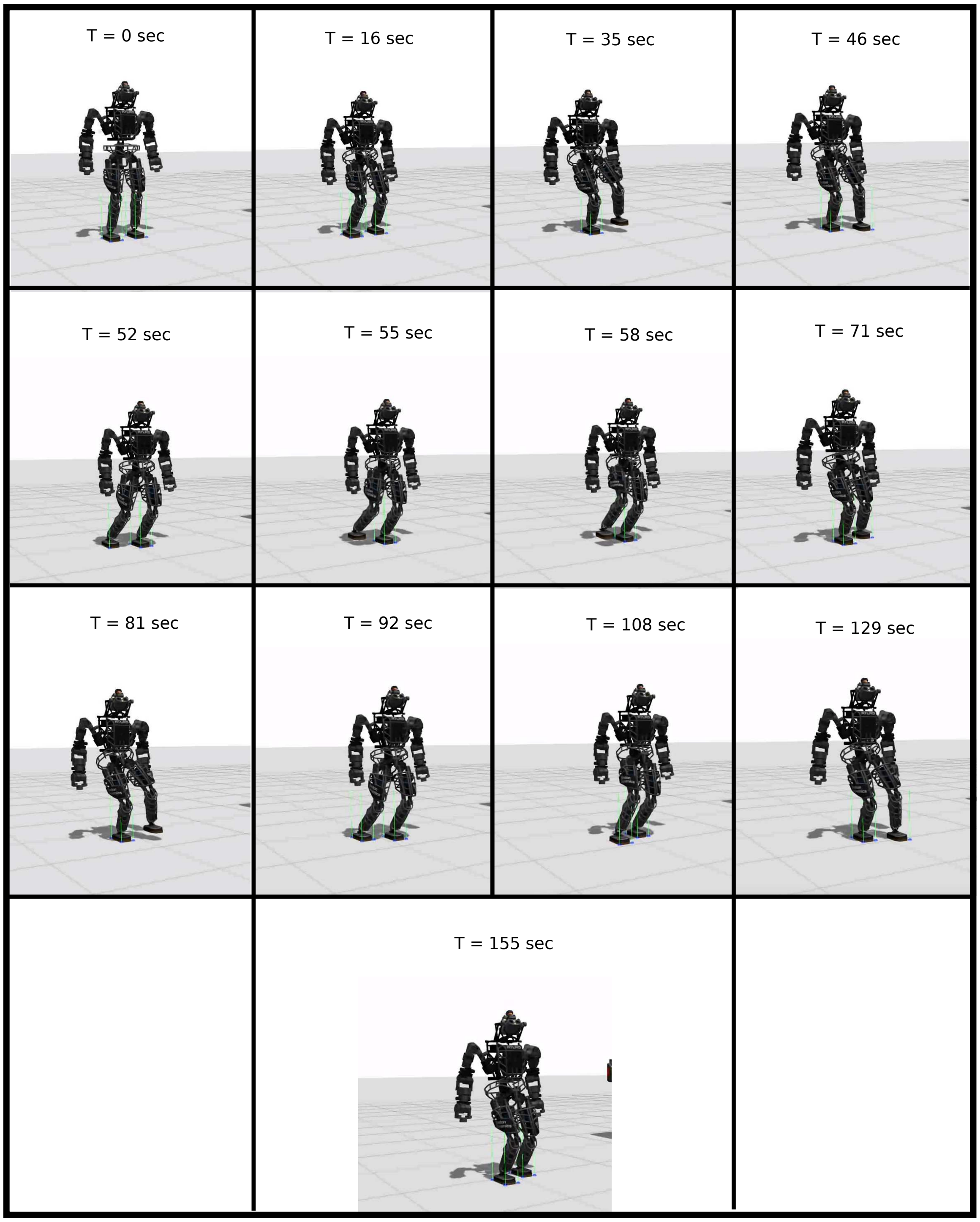

Figure 8.16. Atlas walking simulation image sequence - gait 1.

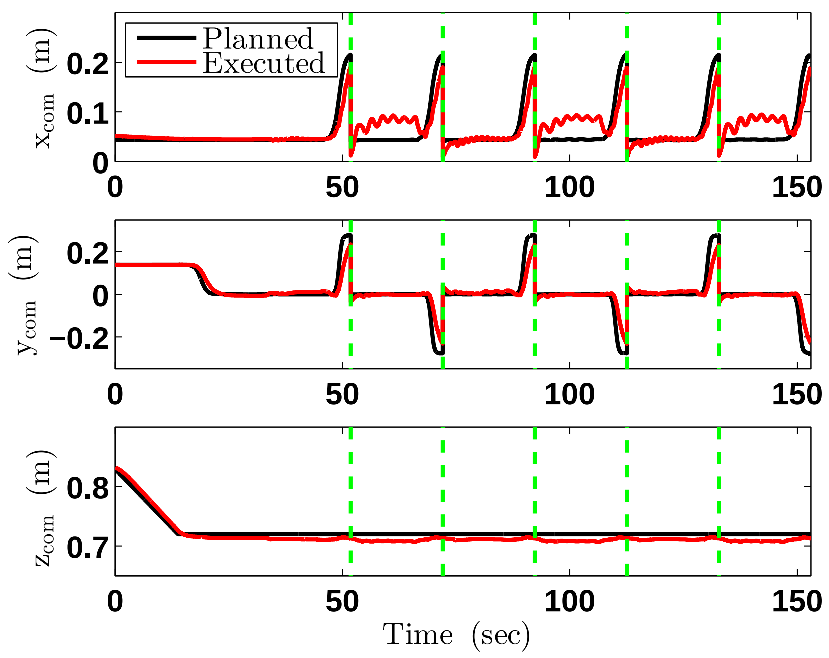

Figure 8.17. COM trajectories for Atlas robot - gait 2.

switch occurs. This is due to the fact that the swing foot is moved forward and kept at a constant foor seperation distance in the y-direction from the base foot.

Since the initial base foot is the right one, the first step is taken with the left foot and therefore, the first swing hand is the right one. The x-position for both hands is given in Fig. 8.18, the results from that figure are discussed here in tandem with the results from Fig. 8.19; which contains the swing foot position. From the top plot of Fig. 8.18 one can see that the right hand is swung first at approximately 40 seconds, this is the same time when the swing foot (left foot at the time) is also being moved forward in the x-direction as shown in the top plot of Fig. 8.19. At this time the swing foot is already off the ground as is shown in the bottom plot of Fig. 8.19. After the right hand is swung forward it is kept at that position relative to the new base foot (left foot now), this switch happens after 50 seconds. Between approximately 50 and 70 seconds (when the next switch occurs), the swing foot (now

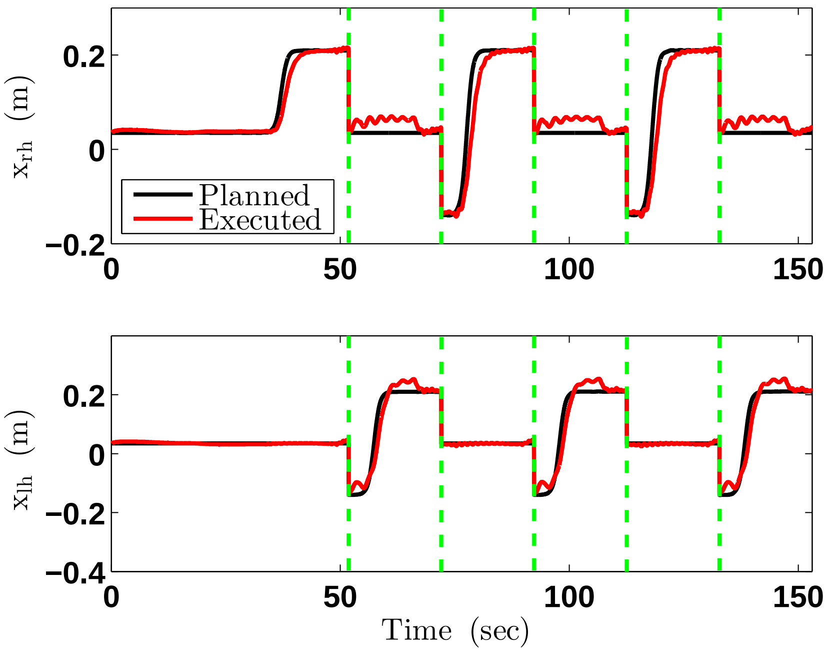

Figure 8.18. Swing hand trajectories for Atlas robot - gait 2.

right foot) is lifted in the air an moved forward as can be seen in the bottom and top plots of Fig. 8.19 respectively. During that period the right hand is kept at the zero x-position while the left hand is swung forward a distance equal to that of the step size (35 cm), this can be seen in the bottom plot of Fig. 8.18 which matches the forward motion of the swing foot (right foot) in the top plot of Fig. 8.19. The periodic behavior of the swing hand and foot motion can be clearly seen from both Figs. 8.18 and 8.19, where during every step the swing foot is lifted off the ground and moved forward relative to the base foot while the opposite hand is swung forward the same distance.

The orientation of the swing foot and torso during the execution of gait 2 are shown in Figs. 8.20 and 8.21. These figures show that the roll, pitch, and yaw of both the swing foot and torso are kept as close as possible to zero during the entire

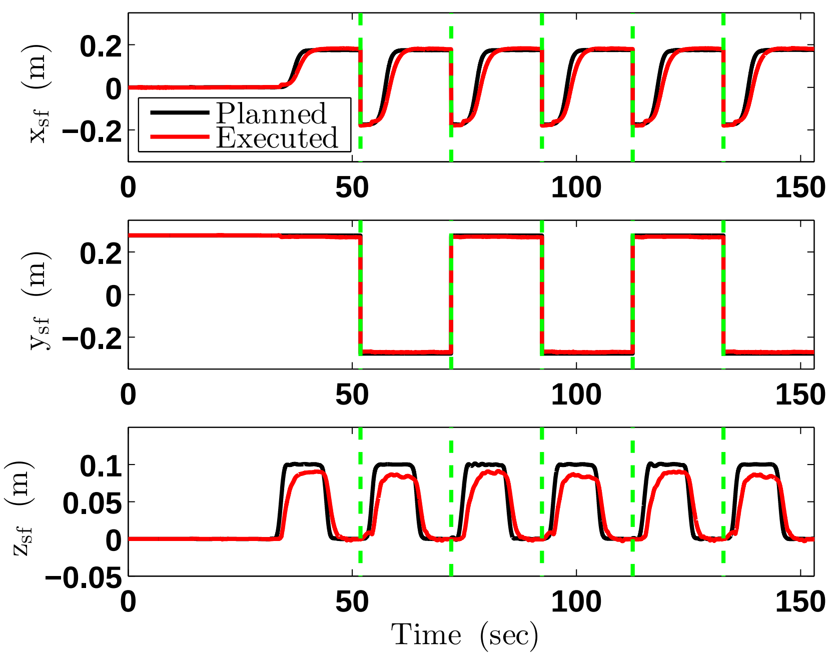

Figure 8.19. Swing foot position for Atlas robot - gait 2.

simulation. The reason for doing so is similar to what was stated earlier for the results from executing gait 1; the torso is required to be upright when walking and the swing foot is kept flat so that when it is lowered towards the ground no undesired or early impacts occur. It is however noted here that the maximum error in both trajectories is less than 5 degrees which is larger than that in gait 1 (2 degrees).

Despite the fact that this deviation is small, on the order of 3 degrees more than gait 1, it can be explained with the two following reasons; first, in gait 1 the arms were not swung during walking as much as there are in gait 2. The swinging in gait 1 was small in magnitude since it was only a product of solving the IK problem while constraining the COM and swing foot. In gait 2 however, the hands are moved forward a substantial distance at every step. This swinging of arms forward induces angular momentum that is mostly confined in the pitching plane. Due to the law of conservation of angular momentum, the swinging motion of the arms affects the

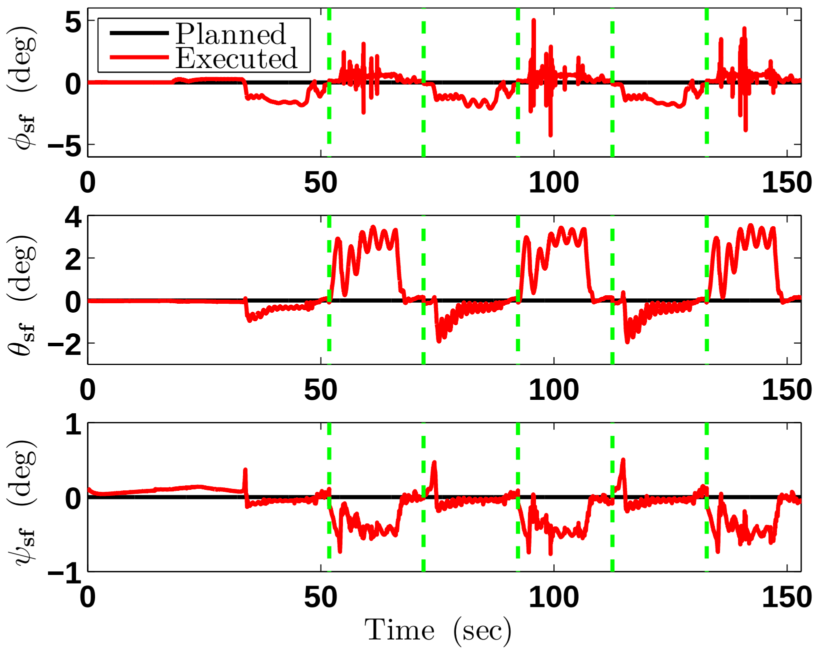

Figure 8.20. Swing foot orientation for Atlas robot - gait 2.

dynamics of the entire humanoid. This is most clearly seen in the pitch of the torso given in the middle plot of Fig. 8.21 where it can be observed that the maximum deviation from the desired value is approximately 4 degrees. When this deviation is compared to that of the torso pitch in gait 1, as given in the middle plot of Fig. 8.15, one can see that it is almost double that quantity. Since the same controller that was used when executing gait 1 is used to execute gait 2, the disturbance induced by swinging the arms in gait 2 is more pronounced on the orientation of the torso. From this result it can be concluded that despite the fact that swinging the arms induced a disturbance on the humanoid, the neuro-adaptive controller is able to keep the error due to that disturbance bounded.

The second reason for the larger deviation in the swing foot and torso orientation trajectories from the desired trajectories is the extra constraints added to the IK problem. In gait 1, only the COM position, swing foot position and orientation,

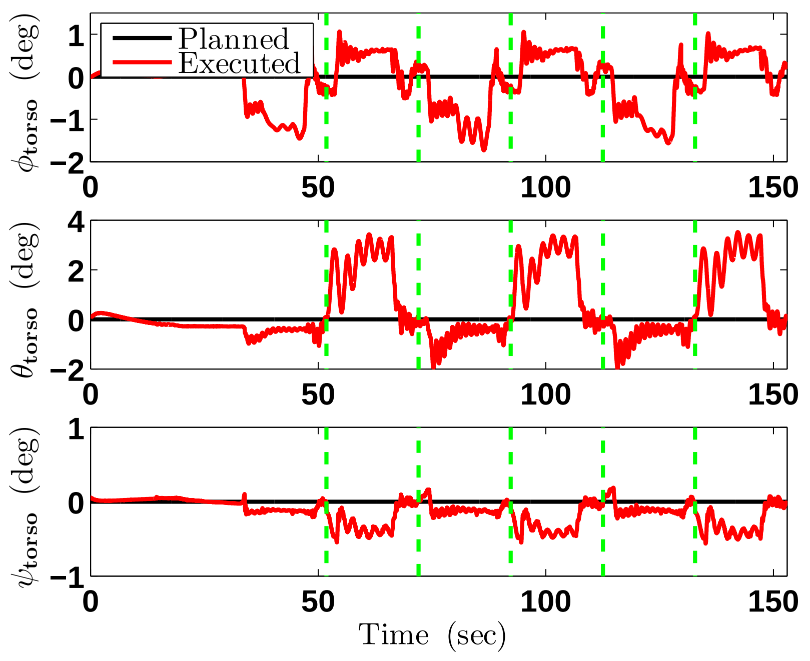

Figure 8.21. Torso orientation for Atlas robot - gait 2.

and torso orientation are given desired values when solving the IK problem. These desired values are considered constraints in the IK problem. In gait 2 however, extra constraints are added by including desired x-positions for both hands throughout the gait. These extra constraints change the solution to the IK problem, and since the problem is solved numerically there is always a tolerance to the given solution. Adding extra constraints to the problem will reduce the numerical solver's ability to generate joint values that satisfy all the constraints with zero tolerance. Therefore, the solution tends to generate desired joint values that when tracked will give a slightly different solution. In this case that can be seen clearly in the swing foot and torso orientation trajectories during the simulation as in Figs 8.20 and 8.21. A time stamped sequence of images for the walking simulation of gait 2 showing the first three steps is shown in Fig. 8.22.

From these results it can be concluded that the neuro-dynamic framework for robotic planning and control can execute complex trajectories on sophisticated robotic systems. The implementation of this framework in the ROS/Gazebo framework is modular, that implies that if robotic hardware for the Atlas or any other robot replaces the Gazebo node in ROS the neuro-dynamic planning and control framework can be implemented on that robot.

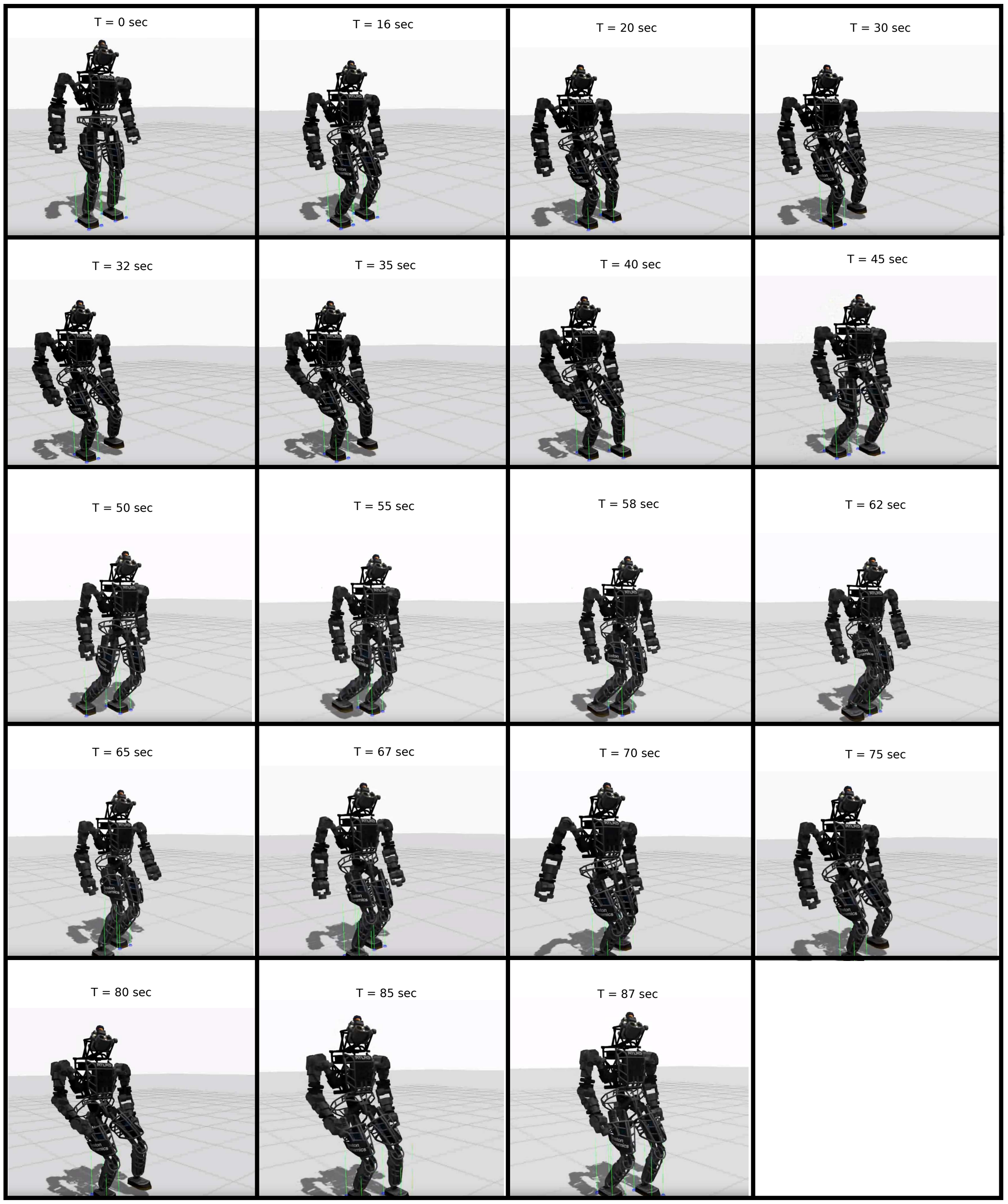

Figure 8.22. Atlas walking simulation image sequence - gait 2.

CHAPTER 9

Concluding Remarks and Proposed Future Work

The robotic planning and control problem was solved in this dissertation by providing solutions to the operational space trajectory generation problem, inverse kinematics problem, and control problem. This outline of the solutions encompasses all kinematic chains and therefore allows for a robotic planning and control framework. To solve the operational space trajectory generation problem the DNN was discussed; by breaking the problem into two parts, one temporal and another spatial, a DNN consisting of a RNN and a FFNN can be used to generate a trajectory. The adaptive planning form of the DNN was also developed by using an intelligent feedback element that utilizes a FFNN. The feedback uses the outputs of the DNN to control the RNN trajectory making the trajectory generation DNN adaptive.

To solve the IK problem the Jacobian inverse needs to be computed. Due to singular configurations, computing this inverse is not possible for all joint configurations. To overcome this problem two methods were devised; the first used APF to avoid singular configurations along the joint trajectory and thereby keeping it singularity free. This however required a priori knowledge of the singular configurations, which might prove to be an impossible task for large degrees of freedom systems. For this reason, a second approach was sought.

By utilizing the concept of neuro-dynamics from the ZNN model, a new model for the M-ZNN was derived to compute the Jacobian inverse in a singularity robust manner. This insured that the computation of the Jacobian inverse is always possible along the joint trajectory. It also provided a time-varying solution to the Jacobian inversion problem given the dynamic nature of the M-ZNN, which is more in line with the time-varying nature of the robotic system than static solutions that are currently employed.

Robotic systems are growing in complexity to execute the more sophisticated tasks they are designed for. This makes deriving a high fidelity, detailed models for the system a rather difficult task. Typically control systems are designed by using models of the robot dynamics, however, if for any reason the operation of the robot is not within the envelope of the model there is no guarantee of controller performance. This is referred to as modeling uncertainty, and the controller designed is augmented with an adaptive or robust term to account for it. In the work presented in this document, online learning FFNN were used to design a neuro-adaptive control system. This system used measurements of the robot's states and estimated the robot's dynamics in real-time. This allowed for an adaptive estimate of the nonlinear dynamics of the robot, and therefore, a controller less affected by modeling uncertainties that is able to track desired joint trajectories.

9.1 Research Conclusions

The results from chapter 5 clearly showed that conventional planning and control methods were highly affected by singular configurations, and caused high joint velocities to occur. Such velocities may not be possible on an actual system, and will damage it during operation. In the same chapter, the APF approach to avoiding singularities was shown to be highly effective in avoiding singular configurations while tracking the same trajectories. From these results it is concluded that the use of APF for planning singularity-free joint trajectories is an effective approach.

By combining the DNN, M-ZNN, and neuro-adaptive controller the same problem from chapter 5 was solved in simulation. Results in chapter 7 showed that the desired operational space trajectory can be tracked without suffering the effect of kinematic singularities. The conclusion to be made from this result is that this neuro-dynamic system is an effective framework for solving robotic planning and control problems in a time varying, singularity-robust, and model uncertainty robust fashion.

The same neuro-dynamic framework was implemented in a more realistic simulation environment in chapter 8. The results from that chapter showed that this neuro-dynamic framework can be used to execute a walking gait using the Atlas humanoid robot. From these results it was concluded that the methods developed in this document not only have substantial theoretical contributions but also practical ones. It can also be concluded that since in simulation the methods can be applied to a complicated high DOF system in real-time, using a realistic robotic framework such as ROS, then implementing such a system on actual hardware is highly plausible and possible. A summary of the conclusions is given below

1. The APF method for singularity-free joint trajectories is an effective approach as verified in simulation.
2. The neuro-dynamic methods developed in this research are an effective framework for solving robotic planning and control problems in a time varying, singularity-robust, and model uncertainty robust fashion.
3. The neuro-dynamic methods are a substantial theoretical and practical contribution to the solution of the robotic planning and control problem.
4. Implementing this neuro-dynamic framework on robotic hardware is possible as tests on realistic robotic simulations have shown.

9.2 Future Work

The robotic planning and control problem requires solving a multitude of smaller but critical problems, therefore, there is room for future development based on the research presented in this document. Below is a list of possible works that can enhance and validate the methods developed in this research.

1. Developing an exploratory method to search the joint space of a robot during its operation and map out the singular configurations. This can be done only in the vicinity of the robot operating point and a map of singular configurations can be retained. Such a method would relax the a priori knowledge of singular configurations the APF method developed in chapter 5 requires.
2. Use the DNN with adaptive feedback for real-time, online, adaptive gait generation for a humanoid. This can be used to plan stabilizing gaits for a humanoid based on its operating state. It can also be used for generating adaptive gaits for push or fall recovery of humanoid robots.
3. Implement the developed neuro-dynamic planning and control framework on robotic hardware. This can be done using ROS and the in house software libraries developed to test the methods on the Atlas in simulation.

APPENDIX A

THREE-LINK MANIPULATOR

This appendix describes the structure of a three-link robotic manipulator. The manipulator consists of three planar links of lengths a_1, a_2, and a_3. The links are connected to each other at through revolute joints with angles q_1, q_2, and q_3 as illustrated Fig. A.1. The vector $\mathbf{q} \in \mathbb{R}^3$ can be defined as $\mathbf{q} = [q_1 \quad q_2 \quad q_3]^T$, and the vector $\mathbf{x}_e$ can be defined in $\mathbb{R}^3$ as $\mathbf{x}_e = [x_e \quad y_e \quad \theta_e]^T$ or in $\mathbb{R}^2$ as $\mathbf{x}_e = [x_e \quad y_e]^T$, the latter making it a redundant manipulator. x_e and y_e are the coordinates of the end-effector defined in the (x_0, y_0) frame shown in Fig. A.1 and θ_e is the end-effector orientation.

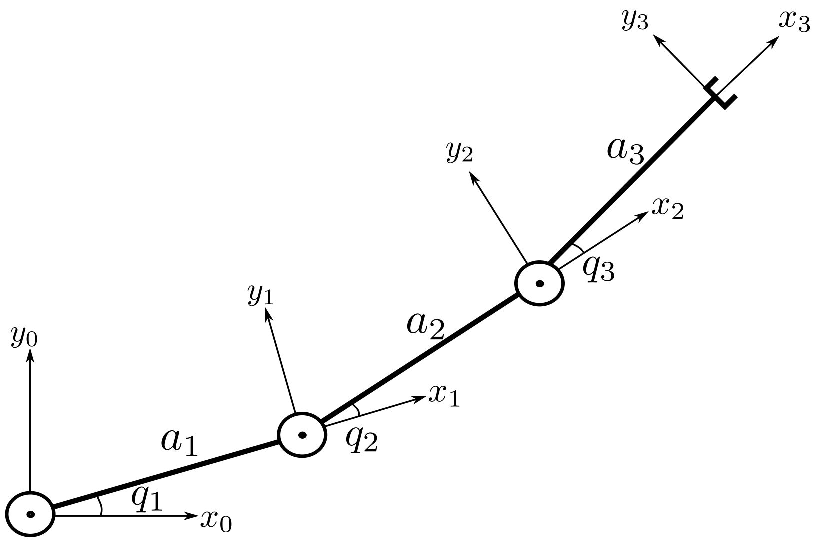

Figure A.1. Diagram illustrating a three-link planar manipulator.

The forward kinematics, i.e the position of the end-effector in the the (x_0, y_0) frame as a function of the joint angles, for the manipulator can be calculated as follows

$$\mathbf{x}_e = \begin{bmatrix} a_1 \cos(q_1) + a_2 \cos(q_1 + q_2) + a_3 \cos(q_1 + q_2 + q_3) \\ a_2 \sin(q_1) + a_2 \sin(q_1 + q_2) + a_3 \sin(q_1 + q_2 + q_3) \\ q_1 + q_2 + q_3 \end{bmatrix}$$

or in the redundant manipulator case as

$$\mathbf{x}_e = \begin{bmatrix} a_1 \cos(q_1) + a_2 \cos(q_1 + q_2) + a_3 \cos(q_1 + q_2 + q_3) \\ a_2 \sin(q_1) + a_2 \sin(q_1 + q_2) + a_3 \sin(q_1 + q_2 + q_3) \end{bmatrix}$$

Based on the given forward kinematics and the differential kinematics from Eq. 3.2, the Jacobian matrix can be defined.

The Jacobian matrix for the three link manipulator is of the form $\mathbf{J} \in \mathbb{R}^{3\times 3}$ when the vector $\mathbf{x}_e \in \mathbb{R}^3$, and of the form $\mathbf{J} \in \mathbb{R}^{2\times 3}$ if $\mathbf{x}_e \in \mathbb{R}^2$. Below are the definitions of the elements $\mathbf{J}_{ij}$.

$$J_{11} = -a1\ \sin(q_1) - a2\ \sin(q_1 + q_2) - a3\ \sin(q_1 + q_2 + q_3)$$

$$J_{12} = -a2\ \sin(q_1 + q_2) - a3\ \sin(q_1 + q_2 + q_3)$$

$$J_{13} = -a3\ \sin(q_1 + q_2 + q_3)$$

$$J_{21} = a1\ \cos(q_1) + a2\ \cos(q_1 + q_2) + a3\ \cos(q_1 + q_2 + q_3)$$

$$J_{22} = a2\ \cos(q_1 + q_2) + a3\ \cos(q_1 + q_2 + q_3)$$

$$J_{23} = a3\ \cos(q_1 + q_2 + q_3)$$

$$J_{31} = 1$$

$$J_{32} = 1$$

$$J_{33} = 1$$

The determinant of the Jacobian matrix for the three-link manipulator can be shown to be

$$|\mathbf{J}| = a_2 a_1 \left(-\cos\left(q_1 + q_2\right)\sin\left(q_1\right)\right) + a_2 a_1 \left(\sin\left(q_1 + q_2\right)\cos\left(q_1\right)\right) \tag{A.1}$$

When examining the Jacobian for this manipulator it is found that the values of q_1 and q_3 have no bearing in the singularity configurations. It can also be seen that $q_2 = n\pi$ with $(n = 0, 1, 2, ...)$ is a singularity. Therefore, that the singularity configurations correspond to those where $q_2 = n\pi$ with n is any integer and are not dependent on the values of q_1 and q_3.

APPENDIX B

ATLAS HUMANOID ROBOT

The information presented in this appendix is found in the source code for the DRC simulator, which is publicly available [1]. Of interest is the Atlas URDF file (refer to section 8.2) which can be found in the source code as well [2].

B.1 Atlas Kinematic Structure

Atlas is a humanoid robot with 28 hydraulically actuated joints. The kinematic structure of atlas is shown in Fig. B.1; six joints for each arm, six joints for each leg, and four joints for the body and head. The Atlas can be treated as a kinematic chain with a base link, the convention used here is selecting either of the feet as a base frame and defining the links and joints starting from that frame. This is a valid approach since all applications presented here are based on having at least one foot on the ground, that foot can be selected to be the base link. In this appendix the left foot will be used as a base link and will be given an index of 0. All the link numbers assigned in Fig. B.1 are based on that convention. The axes used to represent a link's frame are color coded as follows: red is the x-axis, green is the y-axis, blue is the z-axis. Table B.6 contains the mass of each link.

The left leg structure is shown in Fig. B.2, it consists of six links and six angles as summarized in table B.1. The left arm is shown in Fig. B.3, it consists of six links and six joints which are summarized in table B.3. The right leg and arm have the same configuration as their left counterparts and the links and joints associated with them are summarized in table B.2 and B.4 respectively. The body of the Atlas

[1] https://bitbucket.org/osrf/drcsim/src/583228ebd72677c36a393b3260f0c5f0bc7b0f87?at=default

[2] https://bitbucket.org/osrf/drcsim/src/084739a7530dd022180fee281b1d5ccffc6bbeb6/ros/atlas_description/urdf/atlas_simple_shapes.urdf?at=default&fileviewer=file-view-default

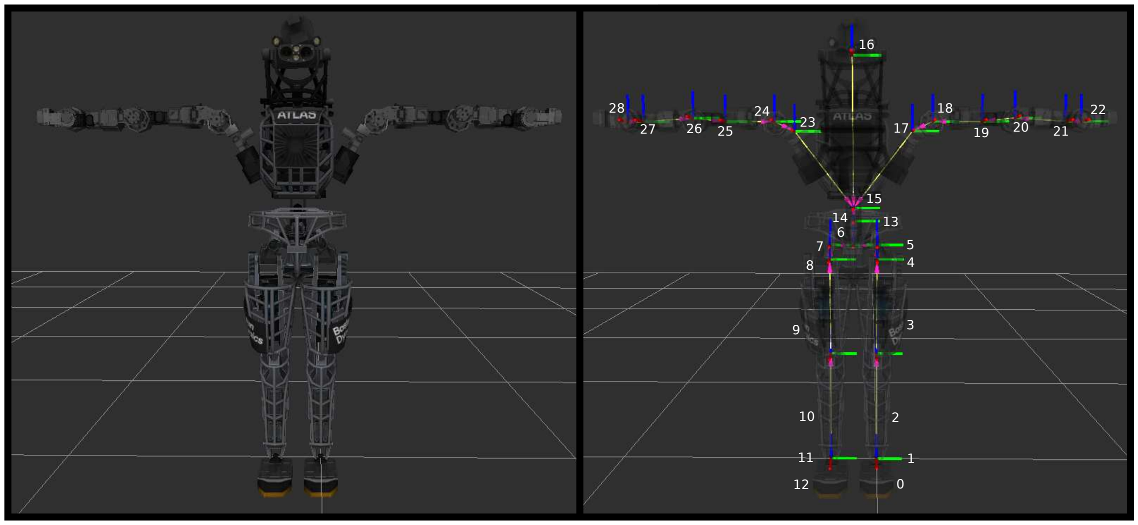

Figure B.1. Atlas humanoid robot kinematic structure, joints at zero position.

Table B.1. Atlas humanoid robot left leg - links and joints

Link #	Link name	Angle	Axis of rotation
0	Left foot	θ_1	x
1	Left talus	θ_2	y
2	Left lower leg	θ_3	y
3	Left upper leg	θ_4	y
4	Left lower glute	θ_5	x
5	Left upper glute	θ_6	z

consists of five links and four joints as shown in Fig. B.4 and summarized in table B.5.

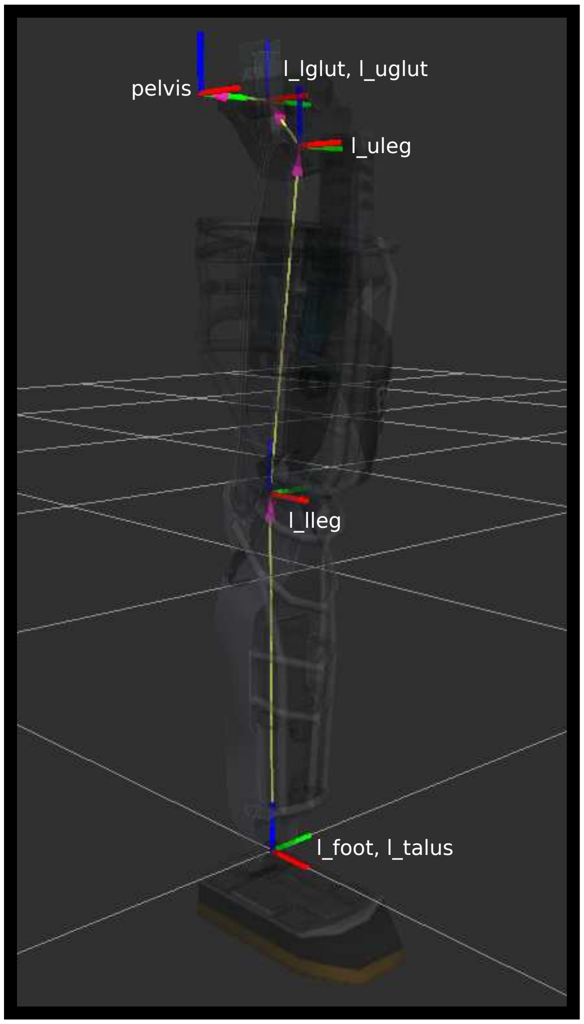

Figure B.2. Atlas humanoid robot left leg kinematic structure.

Table B.2. Atlas humanoid robot right leg - links and joints

Link #	Link name	Angle	Axis of rotation
12	Right foot	θ_{12}	x
11	Right talus	θ_{11}	y
10	Right lower leg	θ_{10}	y
9	Right upper leg	θ_9	y
8	Right lower glute	θ_8	x
7	Right upper glute	θ_7	z

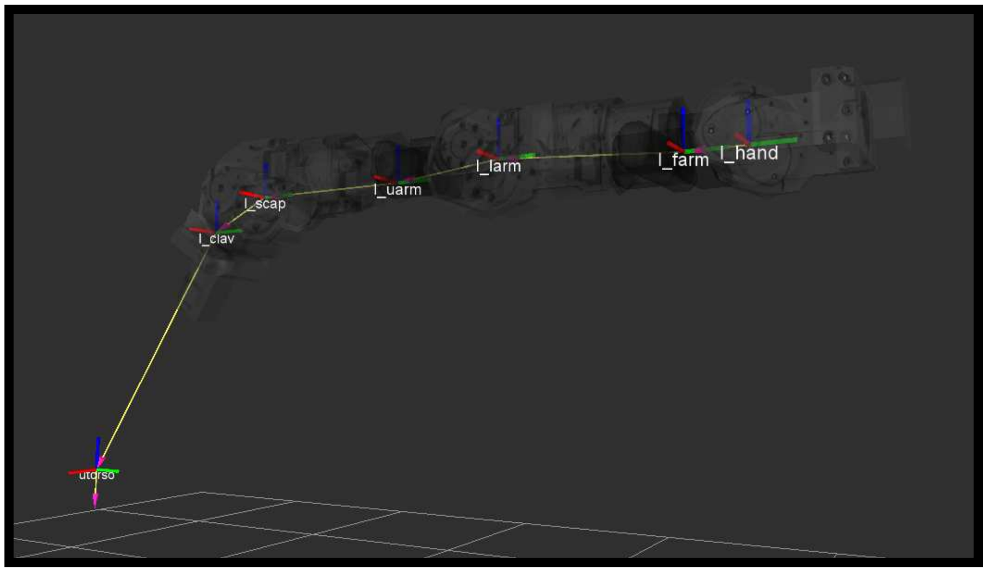

Figure B.3. Atlas humanoid robot left arm kinematic structure.

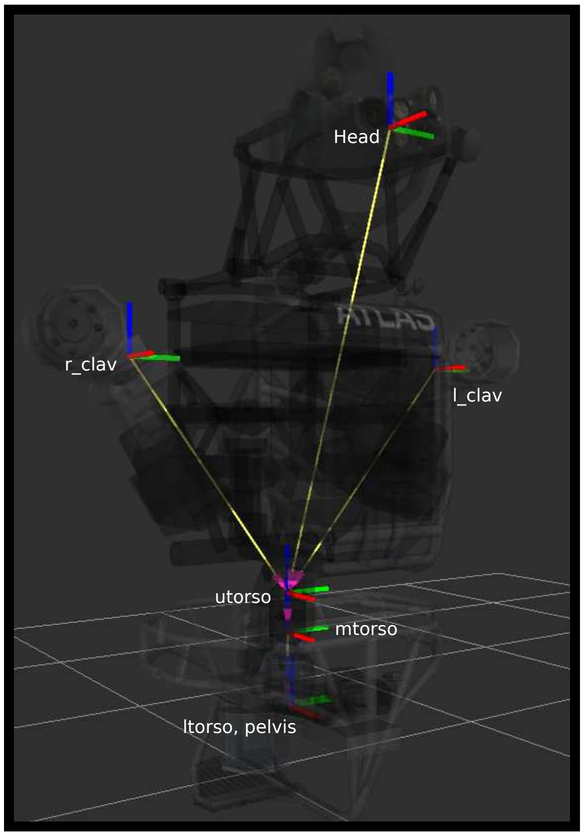

Figure B.4. Atlas humanoid robot body kinematic structure.

Table B.3. Atlas humanoid robot left arm - links and joints

Link #	Link name	Angle	Axis of rotation
23	Left clavicle	θ_{23}	y
24	Left scapula	θ_{24}	x
25	Left upper arm	θ_{25}	y
26	Left lower arm	θ_{26}	x
27	Left forearm	θ_{27}	y
28	Left hand	θ_{28}	x

Table B.4. Atlas humanoid robot right arm - links and joints

Link #	Link name	Angle	Axis of rotation
17	Right clavicle	θ_{23}	y
18	Right scapula	θ_{24}	x
19	Right upper arm	θ_{25}	y
20	Right lower arm	θ_{26}	x
21	Right forearm	θ_{27}	y
22	Right hand	θ_{28}	x

Table B.5. Atlas humanoid robot body - links and joints

Link #	Link name	Angle	Axis of rotation
6	Pelvis	θ_{13}	z
13	Lower torso	θ_{14}	y
14	Middle torso	θ_{15}	x
15	Upper torso	θ_{16}	y
16	Head	-	-

Table B.6. Atlas humanoid robot link mass in kg

Link #	Link mass	Link #	Link mass	Link #	Link mass	Link #	Link mass
0	1.634	7	0.5166	14	0.55	21	0.981
1	0.1	8	0.69	15	18.484	22	2.263
2	4.367	9	7.34	16	1.41984	23	2.369
3	7.34	10	4.367	17	2.369	24	2.707
4	0.69	11	0.1	18	2.707	25	1.881
5	0.5166	12	1.634	19	1.881	26	2.148
6	14.25	13	1.92	20	2.148	27	0.981
28	2.263	-	-	-	-	-	-

REFERENCES

[1] L. Sciavicco and L. Villani, *Robotics: modelling, planning and control.* Springer, 2009.

[2] R. Boudreau and R. P. Podhorodeski, "Singularity analysis of a kinematically simple class of 7-jointed revolute manipulators," *Transactions of the Canadian Society for Mechanical Engineering*, vol. 34, no. 1, pp. 105–117, 2010.

[3] R. Mayorga and A. Wong, "A singularities avoidance method for the trajectory planning of redundant and nonredundant robot manipulators," in *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, vol. 4. IEEE, 1987, pp. 1707–1712.

[4] M. V. Kiréanski and T. M. Petrovié, "Combined analytical-pseudoinverse inverse kinematic solution for simple redundant manipulators and singularity avoidance," *The International journal of robotics research*, vol. 12, no. 2, pp. 188–196, 1993.

[5] T.-W. Park and H.-S. Yang, "A study on singularity avoidance and robust control of redundant robot," in *Intelligent Control and Automation, 2002. Proceedings of the 4th World Congress on*, vol. 3. IEEE, 2002, pp. 1687–1691.

[6] R. V. Mayorga and P. Sanongboon, "Inverse kinematics and geometrically bounded singularities prevention of redundant manipulators: An artificial neural network approach," *Robotics and Autonomous Systems*, vol. 53, no. 3, pp. 164–176, 2005.

[7] S. S. Parsa, H. M. Daniali, and R. Ghaderi, "Optimization of parallel manipulator trajectory for obstacle and singularity avoidances based on neural network," *The*

International Journal of Advanced Manufacturing Technology, vol. 51, no. 5-8, pp. 811–816, 2010.

[8] C. Qiu, Q. Cao, and S. Miao, “An on-line task modification method for singularity avoidance of robot manipulators,” *Robotica*, vol. 27, no. 04, pp. 539–546, 2009.

[9] S. Yahya, M. Moghavvemi, and H. A. Mohamed, “Geometrical approach of planar hyper-redundant manipulators: Inverse kinematics, path planning and workspace,” *Simulation Modelling Practice and Theory*, vol. 19, no. 1, pp. 406–422, 2011.

[10] C. C. Cheah and X. Li, “Singularity-robust task-space tracking control of robot,” in *Robotics and Automation (ICRA), 2011 IEEE International Conference on.* IEEE, 2011, pp. 5819–5824.

[11] T. Rybus, T. Barcinski, J. Lisowski, K. Seweryn, J. Nicolau-Kuklinski, J. Grygorczuk, M. Krzewski, K. Skup, T. Szewczyk, and R. Wawrzaszek, “Experimental demonstration of singularity avoidance with trajectories based on the bézier curves for free-floating manipulator,” in *Robot Motion and Control (RoMoCo), 2013 9th Workshop on.* IEEE, 2013, pp. 141–146.

[12] B. Dasgupta and T. Mruthyunjaya, “Singularity-free path planning for the stewart platform manipulator,” *Mechanism and Machine Theory*, vol. 33, no. 6, pp. 711–725, 1998.

[13] S.-H. Cha, T. A. Lasky, and S. A. Velinsky, “Kinematically-redundant variations of the 3-r rr mechanism and local optimization-based singularity avoidance,” *Mechanics based design of structures and machines*, vol. 35, no. 1, pp. 15–38, 2007.

[14] O. Bohigas, M. E. Henderson, L. Ros, M. Manubens, and J. M. Porta, "Planning singularity-free paths on closed-chain manipulators," *Robotics, IEEE Transactions on*, vol. 29, no. 4, pp. 888–898, 2013.

[15] C. P. Connette, C. Parlitz, M. Hagele, and A. Verl, "Singularity avoidance for over-actuated, pseudo-omnidirectional, wheeled mobile robots," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on.* IEEE, 2009, pp. 4124–4130.

[16] A. Dietrich, T. Wimbock, A. Albu-Schaffer, and G. Hirzinger, "Singularity avoidance for nonholonomic, omnidirectional wheeled mobile platforms with variable footprint," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on.* IEEE, 2011, pp. 6136–6142.

[17] F. Lewis, S. Jagannathan, and A. Yesildirak, *Neural network control of robot manipulators and non-linear systems.* CRC Press, 1998.

[18] C.-L. Hwang and J.-Y. Huang, "Neural-network-based 3-d localization and inverse kinematics for target grasping of a humanoid robot by an active stereo vision system," in *Neural Networks (IJCNN), The 2012 International Joint Conference on.* IEEE, 2012, pp. 1–8.

[19] J. A. Shah, S. Rattan, and B. C. Nakra, "Kinematic analysis of 3-dof planer robot using artificial neural network," *IAES International Journal of Robotics and Automation (IJRA)*, vol. 1, no. 3, pp. 145–151, 2012.

[20] H. Toshani and M. Farrokhi, "Real-time inverse kinematics of redundant manipulators using neural networks and quadratic programming: A lyapunov-based approach," *Robotics and Autonomous Systems*, vol. 62, no. 6, pp. 766–781, 2014.

[21] T. Yuan and Y. Feng, "A new algorithm for solving inverse kinematics of robot based on bp and rbf neural network," in *Instrumentation and Measurement,*

Computer, Communication and Control (IMCCC), 2014 Fourth International Conference on. IEEE, 2014, pp. 418–421.

[22] A.-V. Duka, “Neural network based inverse kinematics solution for trajectory tracking of a robotic arm,” *Procedia Technology*, vol. 12, pp. 20–27, 2014.

[23] R. R. Kumar and P. Chand, “Inverse kinematics solution for trajectory tracking using artificial neural networks for scorbot er-4u,” in *Automation, Robotics and Applications (ICARA), 2015 6th International Conference on.* IEEE, 2015, pp. 364–369.

[24] R. Köker, T. Çakar, and Y. Sari, “A neural-network committee machine approach to the inverse kinematics problem solution of robotic manipulators,” *Engineering with Computers*, vol. 30, no. 4, pp. 641–649, 2014.

[25] R. Köker, “A neuro-simulated annealing approach to the inverse kinematics solution of redundant robotic manipulators,” *Engineering with Computers*, vol. 29, no. 4, pp. 507–515, 2013.

[26] P. Zhang, Z. Yan, and J. Wang, “Obstacle and singularity avoidance for kinematically redundant manipulators based on neurodynamic optimization,” in *Intelligent Control and Information Processing (ICICIP), 2014 Fifth International Conference on.* IEEE, 2014, pp. 460–465.

[27] Z. Miao, Y. Wang, and Y. Yang, “Robust tracking control of uncertain dynamic nonholonomic systems using recurrent neural networks,” *Neurocomputing*, vol. 142, pp. 216–227, 2014.

[28] S. I. Han and J. M. Lee, “Fuzzy echo state neural networks and funnel dynamic surface control for prescribed performance of a nonlinear dynamic system,” *Industrial Electronics, IEEE Transactions on*, vol. 61, no. 2, pp. 1099–1112, 2014.

[29] L. A. Vázquez, F. Jurado, and A. Y. Alanís, "Decentralized identification and control in real-time of a robot manipulator via recurrent wavelet first-order neural network," *Mathematical Problems in Engineering*, vol. 501, p. 451049, 2015.

[30] Y. Oniz and O. Kaynak, "Control of a direct drive robot using fuzzy spiking neural networks with variable structure systems-based learning algorithm," *Neurocomputing*, vol. 149, pp. 690–699, 2015.

[31] X. Zhong, X. Zhong, and X. Peng, "Robots visual servo control with features constraint employing kalman-neural-network filtering scheme," *Neurocomputing*, vol. 151, pp. 268–277, 2015.

[32] Y. Zhang, D. Jiang, and J. Wang, "A recurrent neural network for solving sylvester equation with time-varying coefficients," *Neural Networks, IEEE Transactions on*, vol. 13, no. 5, pp. 1053–1063, 2002.

[33] Y. Zhang, Z. Fan, and Z. Li, "Zhang neural network for online solution of time-varying sylvester equation," in *Advances in Computation and Intelligence.* Springer, 2007, pp. 276–285.

[34] Y. Zhang and S. S. Ge, "Design and analysis of a general recurrent neural network model for time-varying matrix inversion," *Neural Networks, IEEE Transactions on*, vol. 16, no. 6, pp. 1477–1490, 2005.

[35] Y.-N. Zhang and H.-F. Peng, "Zhang neural network for linear time-varying equation solving and its robotic application," in *Machine Learning and Cybernetics, 2007 International Conference on*, vol. 6. IEEE, 2007, pp. 3543–3548.

[36] Y. Zhang and Z. Li, "Zhang neural network for online solution of time-varying convex quadratic program subject to time-varying linear-equality constraints," *Physics Letters A*, vol. 373, no. 18, pp. 1639–1643, 2009.

[37] L. Xiao and Y. Zhang, “Solving time-varying inverse kinematics problem of wheeled mobile manipulators using zhang neural network with exponential convergence,” *Nonlinear Dynamics*, vol. 76, no. 2, pp. 1543–1559, 2014.

[38] Y. Zhang, Y. Yang, N. Tan, and B. Cai, “Zhang neural network solving for time-varying full-rank matrix moore–penrose inverse,” *Computing*, vol. 92, no. 2, pp. 97–121, 2011.

[39] M. Vukobratovic and D. Juricic, “Contribution to the synthesis of biped gait,” *Biomedical Engineering, IEEE Transactions on*, no. 1, pp. 1–6, 1969.

[40] D. Juričić and M. Vukobratović, “Mathematical modeling of biped walking systems,” *ASME Publ. 72-WA/BHF*, vol. 13, 1972.

[41] M. Vukobratović and B. Borovac, “Zero-moment pointthirty five years of its life,” *International Journal of Humanoid Robotics*, vol. 1, no. 01, pp. 157–173, 2004.

[42] Q. Huang, S. Kajita, N. Koyachi, K. Kaneko, K. Yokoi, H. Arai, K. Komoriya, and K. Tanie, “A high stability, smooth walking pattern for a biped robot,” in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 1. IEEE, 1999, pp. 65–71.

[43] Q. Huang, K. Yokoi, S. Kajita, K. Kaneko, H. Arai, N. Koyachi, and K. Tanie, “Planning walking patterns for a biped robot,” *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 3, pp. 280–289, 2001.

[44] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Yokoi, and H. Hirukawa, “A realtime pattern generator for biped walking,” in *Robotics and Automation, 2002. Proceedings. ICRA’02. IEEE International Conference on*, vol. 1. IEEE, 2002, pp. 31–37.

[45] R. Raja, A. Dutta, and K. Venkatesh, “New potential field method for rough terrain path planning using genetic algorithm for a 6-wheel rover,” *Robotics and Autonomous Systems*, vol. 72, pp. 295–306, 2015.

[46] E. S. Conkur and R. Gurbuz, "Path planning algorithm for snake-like robots," *Information Technology And Control*, vol. 37, no. 2, 2015.

[47] M. Hehn and R. D'Andrea, "Real-time trajectory generation for quadrocopters," 2015.

[48] B. Lee and H. J. Kim, "Trajectory generation for an automated excavator," in *Control, Automation and Systems (ICCAS), 2014 14th International Conference on.* IEEE, 2014, pp. 716–719.

[49] Y. Wang, Y. Zhao, S. Bortoff, K. Ueda, *et al.*, "A real-time energy-optimal trajectory generation method for a servomotor system," *Industrial Electronics, IEEE Transactions on*, vol. 62, no. 2, pp. 1175–1188, 2015.

[50] M. Behroo and A. Banazadeh, "Near-optimal trajectory generation, using a compound b-spline interpolation and minimum distance criterion with dynamical feasibility correction," *Robotics and Autonomous Systems*, 2015.

[51] E. Kahale, P. Castillo, and Y. Bestaoui, "Minimum time reference trajectory generation for an autonomous quadrotor," in *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on.* IEEE, 2014, pp. 126–133.

[52] Z. Hao, K. Fujimoto, A. Hasegawa, K. Asaba, Y. Hayakawa, and Q. Zhang, "Trajectory generation and control of a biped walking robot based on the double generating functions method," in *Multisensor Fusion and Information Integration for Intelligent Systems (MFI), 2014 International Conference on.* IEEE, 2014, pp. 1–6.

[53] C. Chen, Y. He, C. Bu, J. Han, and X. Zhang, "Quartic bézier curve based trajectory generation for autonomous vehicles with curvature and velocity constraints," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on.* IEEE, 2014, pp. 6108–6113.

[54] J. T. Jang, S. T. Moon, S. Han, H. C. Gong, G.-H. Choi, I. H. Hwang, and J. Lyou, "Trajectory generation with piecewise constant acceleration and tracking control of a quadcopter," in *Industrial Technology (ICIT), 2015 IEEE International Conference on.* IEEE, 2015, pp. 530–535.

[55] C. Liu, Q. Chen, and D. Wang, "Cpg-inspired workspace trajectory generation and adaptive locomotion control for quadruped robots," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 41, no. 3, pp. 867–880, 2011.

[56] A. Abe, "Trajectory planning for flexible cartesian robot manipulator by using artificial neural network: numerical simulation and experimental verification," *Robotica*, vol. 29, no. 05, pp. 797–804, 2011.

[57] H. Duan and L. Huang, "Imperialist competitive algorithm optimized artificial neural networks for ucav global path planning," *Neurocomputing*, vol. 125, pp. 166–171, 2014.

[58] U. A. Syed, F. Kunwar, and M. Iqbal, "Guided autowave pulse coupled neural network (gapcnn) based real time path planning and an obstacle avoidance scheme for mobile robots," *Robotics and autonomous systems*, vol. 62, no. 4, pp. 474–486, 2014.

[59] C. Luo, J. Gao, Y. L. Murphey, and G. E. Jan, "A computationally efficient neural dynamics approach to trajectory planning of an intelligent vehicle," in *Neural Networks (IJCNN), 2014 International Joint Conference on.* IEEE, 2014, pp. 934–939.

[60] C. Luo, S. X. Yang, M. Krishnan, and M. Paulik, "An effective vector-driven biologically-motivated neural network algorithm to real-time autonomous robot navigation," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on.* IEEE, 2014, pp. 4094–4099.

[61] J. Ni, X. Li, X. Fan, and J. Shen, "A dynamic risk level based bioinspired neural network approach for robot path planning," in *World Automation Congress (WAC), 2014*. IEEE, 2014, pp. 829–833.

[62] I. Gavrilut, L. Tepelea, and A. Gacsadi, "Path planning based on intermediate targets using cellular neural networks," in *Engineering of Modern Electric Systems (EMES), 2015 13th International Conference on*. IEEE, 2015, pp. 1–4.

[63] W. Xueli, G. Yapei, and Z. Jianhua, "A novel algorithm for shortest path problem based on pulse coupled neural network," in *Control and Decision Conference (CCDC), 2015 27th Chinese*. IEEE, 2015, pp. 2468–2473.

[64] P. Zegers and M. K. Sundareshan, "Trajectory generation and modulation using dynamic neural networks," *Neural Networks, IEEE Transactions on*, vol. 14, no. 3, pp. 520–533, 2003.

[65] I. Z. TSypkin, *Relay control systems*. CUP Archive, 1984.

[66] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[67] R. Volpe and P. Khosla, "Artificial potentials with elliptical isopotential contours for obstacle avoidance," in *Decision and Control, 1987. 26th IEEE Conference on*, vol. 26. IEEE, 1987, pp. 180–185.

[68] J.-O. Kim and P. K. Khosla, "Real-time obstacle avoidance using harmonic potential functions," *Robotics and Automation, IEEE Transactions on*, vol. 8, no. 3, pp. 338–349, 1992.

[69] A. Poty, P. Melchior, and A. Oustaloup, "Dynamic path planning by fractional potential," in *Computational Cybernetics, 2004. ICCC 2004. Second IEEE International Conference on*. IEEE, 2004, pp. 365–371.

[70] A. Badawy and C. McInnes, "Robot motion planning using hyperboloid potential functions," in *World Congress on Engineering 2007*, 2007.

[71] R. Volpe and P. Khosla, “Manipulator control with superquadric artificial potential functions: Theory and experiments,” *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 20, no. 6, pp. 1423–1436, 1990.

[72] F. L. Lewis, C. T. Abdallah, and D. M. Dawson, *Control of robot manipulators.* Macmillan New York, 1993, vol. 236.

[73] D. S. Bernstein, *Matrix mathematics: theory, facts, and formulas.* Princeton University Press, 2009.

[74] C. W. Wampler, “Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods,” *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 16, no. 1, pp. 93–101, 1986.

[75] R. M. Murray, Z. Li, S. S. Sastry, and S. S. Sastry, *A mathematical introduction to robotic manipulation.* CRC press, 1994.

[76] F. L. Lewis, K. Liu, and A. Yesildirek, “Neural net robot controller with guaranteed tracking performance,” *Neural Networks, IEEE Transactions on*, vol. 6, no. 3, pp. 703–715, 1995.

[77] F. L. Lewis, A. Yesildirek, and K. Liu, “Multilayer neural-net robot controller with guaranteed tracking performance,” *Neural Networks, IEEE Transactions on*, vol. 7, no. 2, pp. 388–399, 1996.

[78] Y. H. Kim and F. L. Lewis, “Neural network output feedback control of robot manipulators,” *Robotics and Automation, IEEE Transactions on*, vol. 15, no. 2, pp. 301–309, 1999.

[79] G. M. Atmeh, I. Ranatunga, D. O. Popa, K. Subbarao, F. Lewis, and P. Rowe, “Implementation of an adaptive, model free, learning controller on the atlas robot,” in *American Control Conference (ACC), 2014.* IEEE, 2014, pp. 2887–2892.

[80] DARPA. (2015) Darpa robotics challenge. [Online]. Available: http://www.theroboticschallenge.org/

[81] BDI. (2015) Boston dynamics webpage. [Online]. Available: http://www.bostondynamics.com/

[82] OSRF. (2014) Gazebo sim website. [Online]. Available: http://gazebosim.org/

[83] J. Hsu and N. Koenig. (2012) Gazebo, presentation at roscon 2012. [Online]. Available: http://www.osrfoundation.org/wordpress2/wp-content/uploads/2015/04/Gazebo-ROSCon-presentation.pdf

[84] OSRF. (2014) Build a robot tutorial - gazebo. [Online]. Available: http://gazebosim.org/tutorials?cat=build_robot

[85] OSRF. (2015) Robot operating system. [Online]. Available: http://www.ros.org/

[86] H. Roger and R. J. Charles, "Topics in matrix analysis," 1994.

[87] Eigen-Developers. (2015) Eigen library website. [Online]. Available: http://eigen.tuxfamily.org/index.php?title=Main_Page

[88] Eigen-Developers. (2015) Eigen library documentation. [Online]. Available: http://eigen.tuxfamily.org/dox/classEigen_1_1MatrixBase.html#aa2834da4c855fa35fed8c4030f79f9da

BIOGRAPHICAL STATEMENT

Ghassan M Atmeh is an engineer and technology enthusiast. He received his BSc in mechanical engineering in 2010 from the Jordan University of Science and Technology (JUST). Afterwards, he received his MSc (2012) and PhD (2015) in aerospace engineering at the University of Texas at Arlington (UTA). During his graduate studies at UTA Ghassan worked as a research assistant to develop neuro-adaptive control technology for the UTA Research Institute (UTARI) to compete in the DARPA robotics challenge. He also worked as a teaching assistant for multiple courses at the mechanical and aerospace engineering (MAE) department at UTA. In 2013 Ghassan started teaching as an adjunct professor at the MAE department of UTA while working on his doctoral dissertation. Ghassan mainly works in the areas of dynamical systems, sensor data processing, and control theory as applied to robotics and unmanned systems. He has worked on problems in unmanned aerial and ground vehicle navigation and control, artificial neural network design, development of intelligent systems, design of space robotics, robotic arm fabrication, humanoid robot locomotion, and sensor fusion.