COMPARATIVE STUDY OF CONTROL STRATEGIES FOR UNDERACTUATED MANIPULATOR

A THESIS IN MECHATRONICS

Presented to the faculty of the American University of Sharjah
School of Engineering
in the partial fulfilment of
the requirements for the degree

MASTER OF SCIENCE

by
JOUDEH YASIN ABED
B.S. 2000

Sharjah, UAE
June 2006

We approve the thesis of Joudeh Yaisn Mohammad Abed

Date and Signature

Dr. Mohammad Amin Al Jarrah _____

Professor of Mechanical Engineering

Thesis advisor

Dr. Ibrahim Deiab _____

Assist. Professor of Mechanical Engineering

Graduate Committee

Dr. Riadh Al Khazaali _____

Professor of Electrical Engineering

Graduate Committee

Dr. Nabil M. Abdel Jabbar _____

Assoc. Professor of Chemical Engineering

Graduate Committee

Dr. Mohammad Amin Al Jarrah _____

Program Director

Dr. Yousef Al Asaaf _____

Interim Dean of Engineering School

Dr. Judith Killen _____

Director of Graduate Study Researches

© 2006

JOUDEH YASIN ABED

ALL RIGHTS RESERVED

COMPARATIVE STUDY OF CONTROL STRATEGIES FOR UNDERACTUATED MANIPULATOR

Joudeh Yasin Abed, Candidate for the Master of Science in Mechatronics Engineering

American University of Sharjah, 2006

ABSTRACT

Underactuated mechanical systems are systems with fewer control inputs (actuators) than degrees of freedom. They arise in many applications such as underwater robots, mobile robots and manipulators with passive joints. This thesis considers two-link manipulator with one active joint at shoulder and develops various control strategies for stabilizing the manipulator at the upright position. In nonlinear control, we implemented partial feedback linearization technique for comparison study. We developed sliding mode controller for swinging up the manipulator to the upright position and implemented mono layer sliding mode controller for balancing the manipulator at upright position. In intelligent control, we developed Neuro fuzzy controllers for swinging up and balancing the manipulator to the upright position. In hybrid control, we developed genetic sliding mode controllers for balancing the manipulator at upright position, and tuned the PD gains of feedback linearization technique using genetic algorithm. Results showed that feed back linearization and LQR methods are good for ideal cases when manipulator parameters are precisely known and noise and disturbances are neglected exists. These methods are unable to stabilize the manipulator at the upright position in the presence of certain levels of noise and external disturbance. By tuning the PD gains adaptively using genetic algorithm, feedback linearization tolerates better parameter uncertainties and noise. However, the computational time of Genetic Algorithm is relatively long which

prevents the real time implication for short time constant systems. Sliding mode swinging up controller and sliding mode balancing controller are successful techniques for balancing the manipulator at the upright position. They are robust and tolerate noise and external disturbance. The genetic sliding mode controller improves the robustness, reduces the chattering and reduces the hitting time of sliding surface. Neuro fuzzy swinging and balancing controllers are successful in balancing the manipulator at upright position; however, they were sensitive to noise in signals and external disturbances.

CONTENTS

FIGURES

TABLES

SYMBOLS

m_1 : Total mass of link one

m_2 : Total mass of link two.

$l1$: Length of link one

$l2$: Length of link two

lc_1 : Longitudinal center of mass of link one

lc_2 : Longitudinal center of mass of link two

I_1 : Mass moment of inertia of link one

I_2 : Mass moment of inertia of link two

θ_1 : Angle of joint one

θ_2 : Angle of joint two

$\dot{\theta}_1$: Angular velocity of joint one

$\dot{\theta}_2$: Angle of joint two

G : Gravity acceleration

K Total kinetic energy

V : Total potential energy

L : Lagrange function

τ_1 : Torque applied on joint one

τ_2 : Torque applied on joint two

V_l : Lyapunuv function

AKNOWLEDGMENTS

I would like to thank every one who has taught, assisted, guided and supported me on my quest to obtain the Master's degree.

First and foremost, I would like to thank my advisor Dr. Mohammad Al-Jarrah for his guidance, encouragement and unlimited supports

I would like to thank Mr. Hassan Abdul AL Hamid, the manager of my previous company (REMCO), for his significant support.

I would like to thank my friends Maher, Mahmoud, Mohammad, Moatasem, Ehab, Samer, Omar, Zakeria, and Rabea for their support.

Finally, I would like to deeply thank my mother, father and my family for their love and emotional support.

CHAPTER 1: INTRODUCTION

1.1. Significance

Underactuated mechanical systems are systems with fewer control inputs (actuators) than degree of freedom (DOF). Mechanical robotic systems might be purposely designed underactuated to obtain higher flexibility, reduce robot weight, reduce energy consumption and reduce cost. Mechanical systems may become underactuated because of failure of one or more actuators [39]. Pendubot, acrobat, planar underactuated manipulator, inverted pendulum, double inverted pendulum, and gymnast robot, cart pendulum, and inverted wedge are typical examples of such systems. Furthermore systems can be inherently underactuated as in the case of underwater robots and manipulators with flexible joints [34].

Since underactuated mechanical systems are so broad and have complicated variant dynamics, many research efforts have been made on control aspects and strategies of underactuated systems. However, underactuated systems are still subject to further study to develop other control strategies that ensure safety, robustness, adaptability and fault tolerance. To investigate the control problem of underactuated systems, a two link manipulator arm is considered in this thesis for its dynamics simplicity as shown in Figure 1.1. This underactuated mechanical system was subject of extensive research to develop various control algorithms to deal with the complication of the nonlinearity, instability, and controllability of the underactuated system. The two-link manipulator with one active joint at the shoulder "Pendubot" is selected to be as a test bed for underactuated systems to develop sliding mode controllers, Genetic sliding mode controller and Neuro Fuzzy controllers.

1.2. Previous Works

1.2.1 Underactuated Mechanical Systems in Horizontal Plane

Arai and Tachi [3] have presented a method of controlling the position of an underactuated manipulator equipped with motors and encoders on active joints, and holding brakes and encoders on passives joints. The passive joints are excited to the

desired position using dynamic decoupling and held on the desired position by engaging the holding brakes.

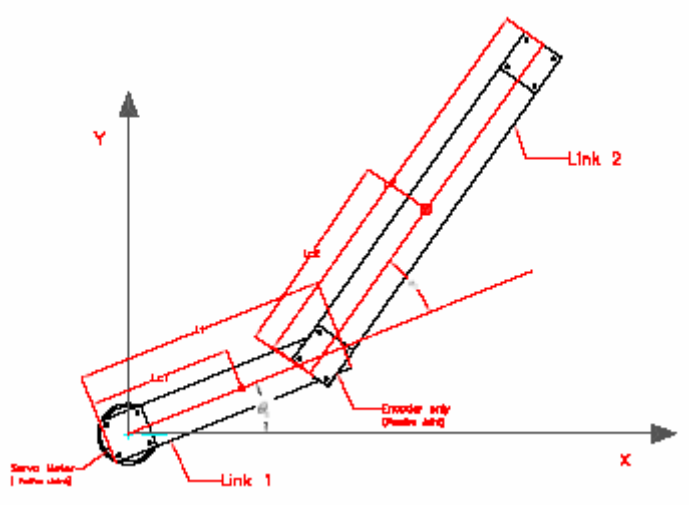


Figure 1.1 Two-link manipulator with one active joint at shoulder "Pendubot"

Also Arai and Tachi [3] have described the controllability conditions of such manipulators. In [2], Arai has proved the controllability of a 3-DOF underactuated manipulator (that is not equipped with brakes on passive joints) by a constructive method in which the trajectory from any initial state to any desired state are composed. In [6], Marcel Bergerman studied the controllability of planner manipulators and developed robust optimal sequential control methods for the underactuated manipulators using variable structure controllers with the aid of brakes on passive joints. He also developed a graphical method to plan a collision –free motion of the manipulator inside its constrained configuration space.

1.2.2 Underactuated Mechanical Systems in Gravity Domain

Since the 1990s, many researchers have been studying control of underactuated mechanical systems in the gravity domain. Their works can be summarized as follows:

a. Non Linear control Methodologies

Spong and Praly [34] have presented the partial feedback linearization technique for swinging up underactuated robots about unstable equilibrium points. They have considered Acrobat, three-link gymnast robot and cart pole system. Daniel Jerome [7]

has designed two-link underactuated manipulator (Pendubot), and used partial feedback linearization technique for a swing up control and linear quadratic regulator (LQR) technique for balancing the manipulator about the upright equilibrium point. Isabelle Fantoni and Rogelio Lozano [12] have presented energy based approach for swinging up the Pendubot and used Lyapunov theory for convergence analysis. Also they have used full-state feedback linear controller for balancing the manipulator at the equilibrium position. In [10], K. L. Carroll presented a robust sliding mode controller for acrobat while in [36] W. Wang and D. Liu have presented a sliding mode controller for overhead crane

b. Linear control Methodologies

Underactuated mechanical systems are not fully linearizable, but they can only be linearized about specific points. Most researchers have used linear control theory for balancing the systems about equilibrium points. Daniel Jerome [7] used linear quadratic regulator (LQR) technique for balancing the Pendubot about the upright equilibrium point. Fantoni and Lozano [12] used full state feedback linear controller for balancing the acrobat about the equilibrium position. G.Giua and A.Usai have developed a gain-scheduling controller for overhead cranes [15].

c. Intelligent Controllers:

A real time Neuro-fuzzy controller is developed for balancing the Pendubot [29]. Brown and Passino [8] have developed fuzzy and adaptive fuzzy controller for balancing an acrobat (two-link planar manipulator with one active joint at the elbow) in the upright unstable equilibrium position. Also they have developed a fuzzy controller for swinging up an acrobat. X. Lai and Z. Gai [19] have developed an energy based fuzzy controller for swinging up an acrobat, while P. Marco, L. Raul [20] have presented a Neuro control scheme for controlling Pendubot.

d. Hybrid Control Strategies:

X. Lai and Z. Gai [19] have developed a fuzzy sliding mode controller for balancing the acrobat in the upright position. In [14], T.Fujinaka and Y. Kishda proposed a self tuning Neuro PID controller for stabilizing a double inverted pendulum. M. Moore and J. Musacchioa [21] have developed a Genetic adaptive

controller for balancing an inverted wedge .Table 1.1 summarizes the previous studies and works in underactuated systems.

Table 1.1: Summary of Previous Works

Field	Methodology	Authors	Contribution	Ref.	Year
Underactuated Systems in Horizantal Plane	Control of Passive Joints Using Holding Brakes	Arai & Tachi	Development of a method for controlling the passive Joints Using Holding Brakes	[3]	1991
	Constructive Method	Arai	Proof of Controllability for 3-DOF manipulator using constructive Method without using holding brakes	[2]	1996
	Variable structure controllers	M. Bergerman	Development of robust optimal sequential control methods with aid of brakes on passive joints	[6]	1996
	Graphical method	M. Bergerman	Development of graphical method to plan a collision –free motion of the manipulator inside its constrained configuration space	[6]	1996
Nonlinear Control of Underactuated Systems in Gravity Domain	Partial Feedback Linearization	Spong & Praly	Presentation of partial feedback linearization technique for swinging up underactuated robots about unstable equilibrium points and study of acrobat, three-link gymnast robot and cart pole.	[34]	1996
		Daniel Jerome	Design of Pendubot and implementation of feedback linearization technique	[7]	1996
	Energy Based Approach	I. Fantoni and R.Lozano	Presentation of energy based approach for swinging up the Pendubot and used Lyapunov theory for convergence analysis	[12]	2002
	Sliding mode Controller	K. L. Carroll	Presentation of robust sliding mode controller for acrobat	[10]	1998
		W.Wang and J.Zhaoand	Presentation of sliding mode controller for overhead crane	[36]	2004

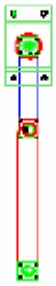
continue Table 1.1

Field	Methodology	Authors	Contribution	Ref.	Year
Linear Control of Underactuated Systems in Gravity Domain	Linear Quadratic Regulator (LQR)	Daniel Jerome	Implementation of linear quadratic regulator (LQR) for balancing the Pendubot.	[7]	1996
	Full State feedback	I. Fantoni and R.Lozano	Implementation of full state feedback linear controller for balancing the acrobat about the equilibrium position.	[12]	2002
	Gain-Scheduling	C. G.Giua, and A.Usai	Development of gain-scheduling controller for overhead cranes.	[15]	1998
Intelligent Controllers	Fuzzy Logic	Brown and Passino	Development of fuzzy and adaptive fuzzy controller for balancing an acrobat	[8]	1997
		Brown and Passino	Development of fuzzy controller for swinging up and balancing acrobat	[8]	1997
		X. Lai and Z.Gai	Development of energy based fuzzy controller is developed for swinging up an acrobat	[19]	2000
	Neural Network	P. Marco,L. Raul	Presentation of Neuro control scheme for controlling Pendubot	[20]	2002
	Neuro Fuzzy	F. L.Rojo and E.Sanchez	Development of real time Neuro-fuzzy controller for balancing the Pendubot	[29]	2002
Hybrid Control Strategies	Fuzzy Sliding Mode	X. Lai and Z.Gai	Development of fuzzy sliding mode controller for balancing the acrobat.	[19]	2000
	Neuro PID Controller	T.Fujinaka and Y. Kishda	Development of self tuning Neuro PID controller for stabilizing a double inverted pendulum	[14]	2000
	Genetic adaptive controller	M. L. Moore and K. M. Passino	Development of Genetic adaptive controller for an inverted wedge for tuning the nonlinear state feedback controller	[21]	2002
	Adaptive Fuzzy Controller	M. L. Moore and K. M. Passino	Development of adaptive Fuzzy controller for an inverted wedge	[21]	2002

1.3. Problem Statement and Contributions of Thesis

Control of Pendubot at an equilibrium position is based on swinging up the Pendubot from its downward position to the attractive region of equilibrium point ($|\theta_1 - \pi/2| < 0.2 \quad rad, |\dot{\theta}_1| < 0.2 \quad rad/s, \quad , |\theta_2| < 0.3 \quad rad, |\dot{\theta}_2| < 0.2 \quad rad/s$), then switching the controller to balancing controller to balance the second link at the upright position. The swinging up controller considers only the states of the first link and it absolutely nonlinear, while balancing controllers considers the full state of mechanical system and could be linearized in that attractive region. Figure 1.2 shows the steps of stabilizing the two link manipulator at the upright position. Spong [34] presented feedback linearization technique for swinging up underactuated systems and LQR technique for balancing them. However, both Feedback Linearization Technique and LQR Techniques are based on precise knowledge of system parameters. Since, mechanical systems always have parameter uncertainties and external disturbances, the actual output of these controllers will deviate from the actual one. Therefore, the goal of this work is to develop controllers that are robust, accurate, stable and insensitive to parameter uncertainties. Consequently, we have used Variable Structure theory (VSC) and developed a sliding mode controller for swinging up the Pendubot, and another sliding mode controller of two sliding surfaces for balancing the Pendubot at upright position. To optimize the reaching time of sliding surfaces and reduce chattering, we have used Genetic Algorithm and developed Genetic Sliding controllers of sliding mode controllers. In intelligent control, we have implemented Neuro fuzzy controllers for swinging up and balancing the Pendubot. Moreover, we have tuned the PD gains of feedback linearization technique adaptively using genetic algorithm.

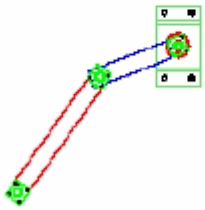
The main contribution in this thesis can be summarized by the development of sliding mode balancing and swinging up controllers for two link manipulator (Pendubot), development of genetic sliding mode controllers, and adoption of Neuro fuzzy controllers for Pendubot and tuning PD gains of feedback linearization technique adaptively.



(1) Initial Equilibrium Position

$\theta_1 = -90^\circ$

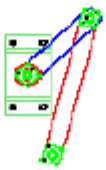
$\theta_2 = 0^\circ$



(2) Pumping Energy

$\theta_1 = -154.8^\circ$

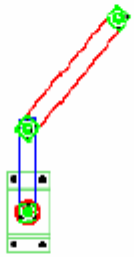
$\theta_2 = 33.4^\circ$



(2) Swinging Up
(Swing up controller)

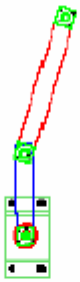
$\theta_1 = -43^\circ$

$\theta_2 = -148^\circ$



(3) Swinging Up
(Swinging Up Controller)

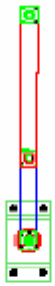
$\theta_1 = -90.3^\circ$



(5) Balancing Controller
(Balancing Controller)

$\theta_1 = -90.4^\circ$

$\theta_2 = -16.9^\circ$



(6) Stabilization at up right position
(Balancing Controller)

$\theta_1 = -90^\circ$

$\theta_2 = 0^\circ$

Figure 1.2 Steps for stabilizing the two link manipulator at up right position

1.4. Organization of Thesis

In this research, we present in Chapter two a brief background of the proposed control methodologies and strategies. In Chapter three we present the kinematics and dynamic of the manipulator. Chapter four verifies the controllability of the manipulator. In chapter five, we present the proposed control strategies for swinging up the manipulator to the upright position. In chapter six, we present the proposed control strategies for balancing the manipulator at the upright position. Chapter seven presents the simulation and experimental results of each control strategy. In Chapter eight we summarize and conclude the results of each control strategy.

CHAPTER 2: CONTROL METHODOLOGIES

2.1. Feedback Linearization

The central idea behind feedback linearization is to algebraically transform a nonlinear system dynamics into a (fully or partially) linear one so that linear control techniques can be applied. As opposed to conventional linearization (i.e., Jacobian linearization) feedback linearization is achieved by exact state transformations and feedback, rather than by linear approximation of the dynamics [31].

In its simplest form, feedback linearization amounts for canceling nonlinearities in a nonlinear system so that the closed-loop dynamics is in a linear form. The method is simply applicable to systems in controllability canonical form (CCF). Even if the equations are not in CCF, they can be transformed to that form by using algebraic transformations. Another option in such a case would be to use partial linearization of the original dynamics instead of full linearization. There are two main ways of doing feedback linearization. The first one is *input-state linearization* where the full state equations are linearized. The second one is *input-output linearization* where the input-output map from the control input u to the output y is linearized, even if the state equations are only partially linearized [31].

2.1.1 Input-state linearization

For a nonlinear system of the form

$$\dot{x} = f(x,u) \tag{2.1}$$

Input-state linearization is done in two steps. First, a state transformation $z = z(x)$ and an input transformation $u = u(x, v)$ is found so that the nonlinear system dynamics is transformed into an equivalent dynamics in the form

$$\dot{z} = Az + Bv \tag{2.2}$$

Where v is designed using standard linear control techniques, such as pole placement [31].

2.1.2 Input-Output Linearization

Linearizing the state equations as in input-state linearization does not necessarily linearize the output equation. The idea in input-output linearization is to obtain a simple and direct relationship between the system output y and input u. If one needs to differentiate the output of a nonlinear system r times to obtain an explicit relationship between u and y, the system is said to have a relative degree r. This is consistent with the notion of relative degree in linear systems (excess of poles over zeros). If the relative degree is less than the degree of state equations, i.e., $r < n$, then a part of the system dynamics called the "internal dynamics" has been rendered unobservable in the input-output linearization; and the system is not input-state linearizable but input-output linearizable. When $r = n$, there is no internal dynamics; and the system is both input-state and input-output linearizable. In order for the closed-loop system to be stable, the internal dynamics must be stable.

It is possible for one choice of output to yield a stable internal dynamics while another choice would lead an unstable one. Therefore, if possible, one should choose the output y ("designer output") such that the internal dynamics is stable, without restricting y to be a physically meaningful quantity [31].

For linear systems the stability of the internal dynamics is simply determined by the location of zeros. This can be extended to nonlinear systems, by defining the so-called zero-dynamics for a nonlinear system. *The zero-dynamics is defined to be the internal dynamics of the system when the system output is kept at zero by the input.* Asymptotic stability of the zero-dynamics is enough to guarantee local asymptotic stability of the internal dynamics. However, no results on the global stability or even large range stability can be drawn for internal dynamics of nonlinear systems. In fact, only local stability is guaranteed for the internal dynamics even if the zero-dynamics is globally exponentially stable [31].

Although conceptually simple, feedback linearization has some important limitations. First of all, it cannot be used for all nonlinear systems. If the zero-dynamics are unstable, feedback linearization cannot be used. . Secondly, the sensitivity to modeling errors is significant [31].

2.2. Sliding Mode Controller

Sliding mode control methodology is based on a notational simplification, which amounts to replacing an n_{th} order tracking problem by a 1_{st} order stabilization problem [11]. First, a lower dimensional manifold is found which can be stabilized and made positively invariant by an appropriate control input. Then a second control is chosen so those system trajectories, which are not initialized on the manifold, can be forced to the manifold within finite time. This is achieved by applying a control input that switches discontinuously between an upper and lower bound. The switching logic is determined as a function of the distance from the invariant manifold. Sliding control has shown to achieve robust performance by effectively accounting for parameter uncertainties and un-modeled dynamics. The drawback is that it results in discontinuous high gain controllers with ensuing chattering of control [31]. To remedy this, various refinements for smooth switching of the control input have also been proposed such as definition of thin boundary layer neighboring the sliding surface. For the sake of simplicity, the formulation for single input systems will be explained here.

Consider the single input dynamic system

$$y^{(n)} = f(x) + b(x)u \tag{2.3}$$

Where y is the (scalar) output of interest, x is the state vector and u is the control input. The (nonlinear) function $f(x)$ is not exactly known, but the extent of the imprecision on $f(x)$ is upper-bounded by a known continuous function of x. Similarly, the control gain $b(x)$ is not exactly known, but is of known sign and is bounded by known, continuous functions of x. Let $\tilde{x}$ be the tracking error vector in the state:

$$\tilde{x} = x - x_d \tag{2.4}$$

Where x_d are the desired states.

And let a time-varying surface $S(t)$ be defined in the state space R_n by the scalar equation $S(x,t) = 0$

$$S(x,t) = \left(\frac{d}{dt} + \lambda \right)^{n-1} \tilde{x} \tag{2.5}$$

λ is a strictly positive constant

. For example, if $n = 3$,

$$S = \ddot{\tilde{x}} + 2\lambda\dot{\tilde{x}} + \lambda^2\tilde{x} \tag{2.6}$$

That is, S is simply a weighted sum of the acceleration, velocity and position error. With the initial condition $x(0) = x_d(0)$, the problem of tracking $x = x_d$ is equivalent to that of remaining on the surface $S(t)$ for all $t>0$. In fact, with the initial condition $x(0) = x_d(0)$, $S = 0$ represents a linear differential equation whose unique solution is $\tilde{x} = 0$. Thus, the problem of tracking the *n*-dimensional vector x_d (i.e., the original n_{th} order tracking problem in x) is in effect replaced by a first order stabilization problem in S. This stabilization task can be achieved by choosing the control law u in (2.3) such that outside of $S(t)$ [31]

$$\frac{1}{2}\frac{d}{dt}S^2 \le -\eta|S| \tag{2.7}$$

Where η is a positive constant.

Equation 2.7 is known as the *sliding condition*, which states that the squared "distance" to the surface, as measured by S^2, decreases along all system trajectories; thus constraining the trajectories to move towards the surface $S(t)$. Since $S(t)$ is an invariant set, once the trajectories reach it, they remain there for all times. The controller design procedure consists of two steps. First, a feedback control law u is selected so as to verify sliding condition (2.7). However, in order to account for modeling imprecision and disturbances, the control law must be discontinuous across $S(t)$. This discontinuity leads to chattering (see Figure 2.1) which is undesirable in practice because it involves high control activity and further may excite high-frequency dynamics neglected in the course of modeling. Thus, in a second step, the discontinuous control law u is suitably smoothed to achieve an optimal trade-off between control bandwidth and tracking precision. This can be achieved by smoothing out the control discontinuity in a thin *boundary layer see* Figure 2.2, neighboring the switching surface [31].

$$B(t) = \{x, S(x,t) \le \phi\ \}, \quad \phi > 0. \tag{2.8}$$

While the fist step accounts for parametric uncertainty, the second step achieves robustness to high frequency un-modeled dynamics. Boundary layer thicknesses ϕ

can be made time varying, and can be monitored so as to well exploit the control “bandwidth” variable [31].

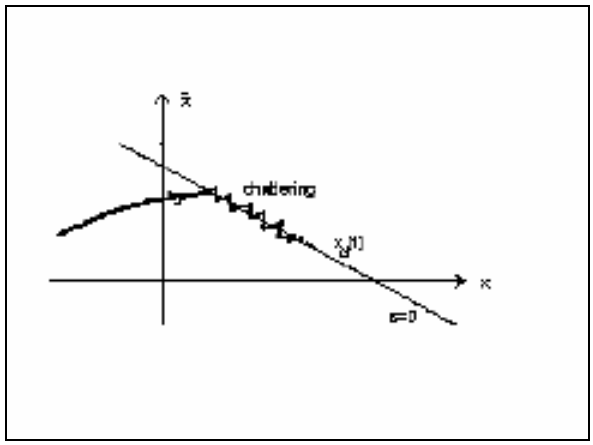


Figure 2.1 Chattering around the sliding surface [31]

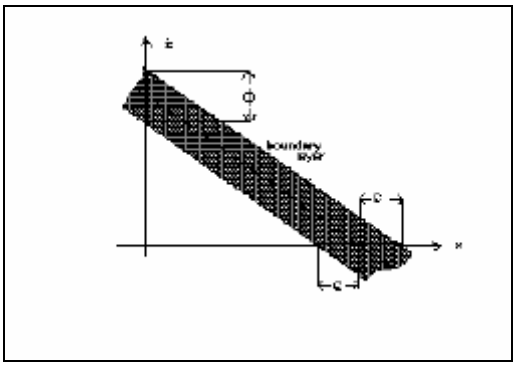


Figure 2.2 Boundary layer of the sliding surface [31]

2.3. Genetic Algorithm

A genetic algorithm (GA) is an optimization search method that manipulates a string of numbers in a manner similar to how chromosomes are changed in biological evolution. An initial population made up of strings of numbers is chosen at random. Each string of numbers is called a "chromosome" or an “individual," and each number slot is called a "gene”. A set of chromosomes forms a population. Each chromosome represents a given number of traits which are the actual parameters that are being varied to optimize the "fitness function". The fitness function is a performance index that is to be maximized [26].

A key part of the success of the GA is the encoding of the strings. Obviously, we need to choose parameters that tend to have reasonable information about the problem. Common encoding of multiple real-values continuous variables (as opposed to binary or digital variables) consists of placing them in integer form and concatenating them, keeping track of decimal places and variable separation points for decoding. Thus, two real numbers 12.345 and 9.8765 could be represented in a 10-digit long string 1234598765. Decoding can occur by first splitting the string at the appropriate point (in this case, between the fifth and sixth digit) and keeping track of the appropriate decimal places. In this case, the actual values are represented by multiplying the integer given by the first portion of the string by 10^{-3} and the integer in the second portion of the string by 10^{-4}. This is base-10 encoding of strings. A base-2 encoding is also popular, where strings are represented in the binary form as a series of 0's and 1's. Regardless of the base of the encoding, the procedure is the same. The strings are decoded for fitness valuation and remain in encoded form for genetic manipulation, producing successive generation until a termination criterion is reached. A termination criterion is used to specify when the GA should end (e.g., the maximum number of generations or until the fitness stops increasing). Figure 2.3 demonstrates creation of random population [26].

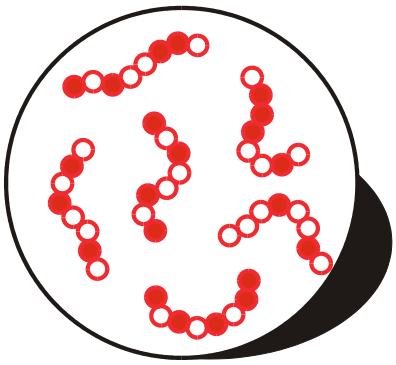


Figure 2.3 Creation of random population of genetic algorithm [27]

The members of a population are manipulated cyclically through three primary "genetic operators" called selection, genetic operation (also referred to as crossover and mutation), and replacement, to produce a new generation (a new population) that tends to have higher overall fitness evaluation [26]. By creating successive generations which continue to evolve, the GA will tend to search for a global optimal solution. The key to the search is the fitness evaluation, see Figure 2.4, parents for the

next generation are selected based on the fitness value of the strings. That is, strings that have a higher fitness value are more likely to be selected as parents, and, thus, are more likely to survive to the next generation [21]. This first stage is the selection stage. Although there are many techniques for the selection of parents, a commonly used method is the Roulette Wheel Selection, which bases the number of offspring on the average fitness of the population. Strings with greater than the average fitness are allocated more than one offspring. The actual method of selection is relatively inconsequential. The key is being that strings with greater fitness tend to produce more offspring [26].

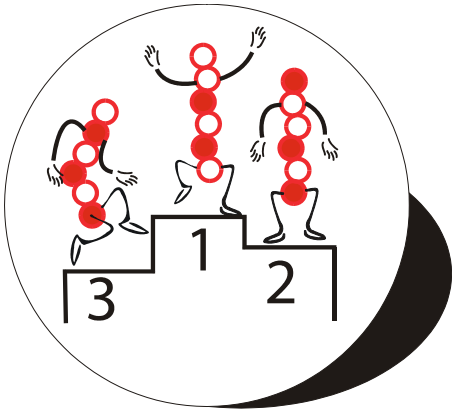


Figure 2.4 Evaluation the fitness of each chromosome (string) [27]

For the evolutionary process, the selected members of a population are randomly paired together to "mate" and share genetic information, see. The GA accomplishes this sharing by the swapping of portions of strings between parents. The simplest model is the single-point crossover, where the selection of a random position between the lengths of chromosomes indicates which portions are interchanged between parents. Multiple crossovers points can also be selected, where strings swap several portions of their strings. The resulting interchange produces two new strings, see Figure 2.6. The actual interchange is often based on a crossover probability (cp). The sharing of genetic material occurs only if a randomly generated normalized number is greater than (cp); otherwise, the strings are not affected [26].

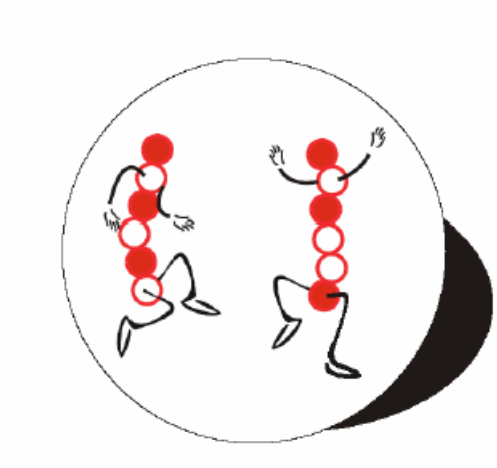


Figure 2.5 Selection of parents and 'Mating' [27]

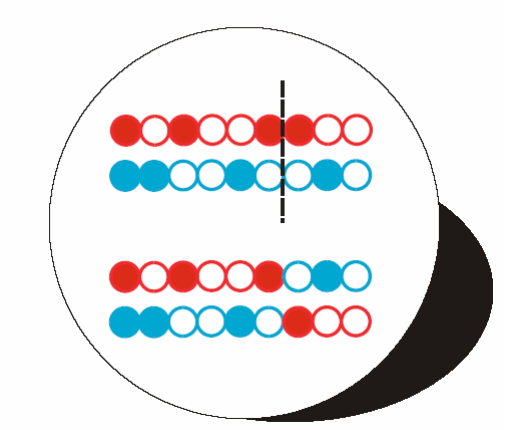


Figure 2.6 Cross over process [27]

Once this crossover stage is complete, the new strings are subject to mutation based on the mutation probability (*mp*). This genetic operation randomly selects a string and position (between1 and lengths of chromosome) and changes the value at that position to a random number (see Figure 2.7). Since the variations due to the mutation operation occur randomly, this tends to keep the Genetic Algorithm from focusing on a local minimum [26].

.

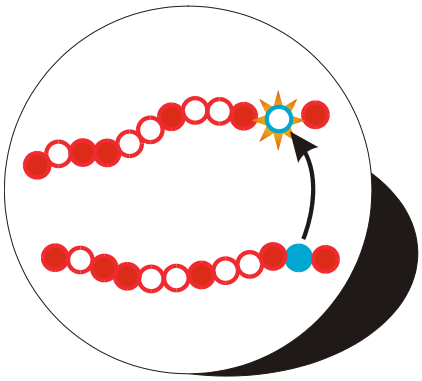


Figure 2.7 Mutation process [27]

The final step in GA is the replacement operation which determines how the new subpopulation produced from the genetic operations, will be introduced as a new generation. Two primary methods exist, complete generation replacement, where each population produces an equal number of strings, which then completely replaces the parent population, and a partial replacement where only a small portion of new strings are developed and introduced into the population. In addition, the former procedure may have a tendency to throw away a best fit solution since the entire generation is replaced. For this reason, often the complete generation replacement method is combined with an " elitist" strategy, where a one or more of the most fit members of a previous population are passed, untouched, to the next generation. This tends to ensure that there always is a string within the population that is a good solution to the problem. Figure 2.8 demonstrates the integration of all Genetic Algorithm processes [26].

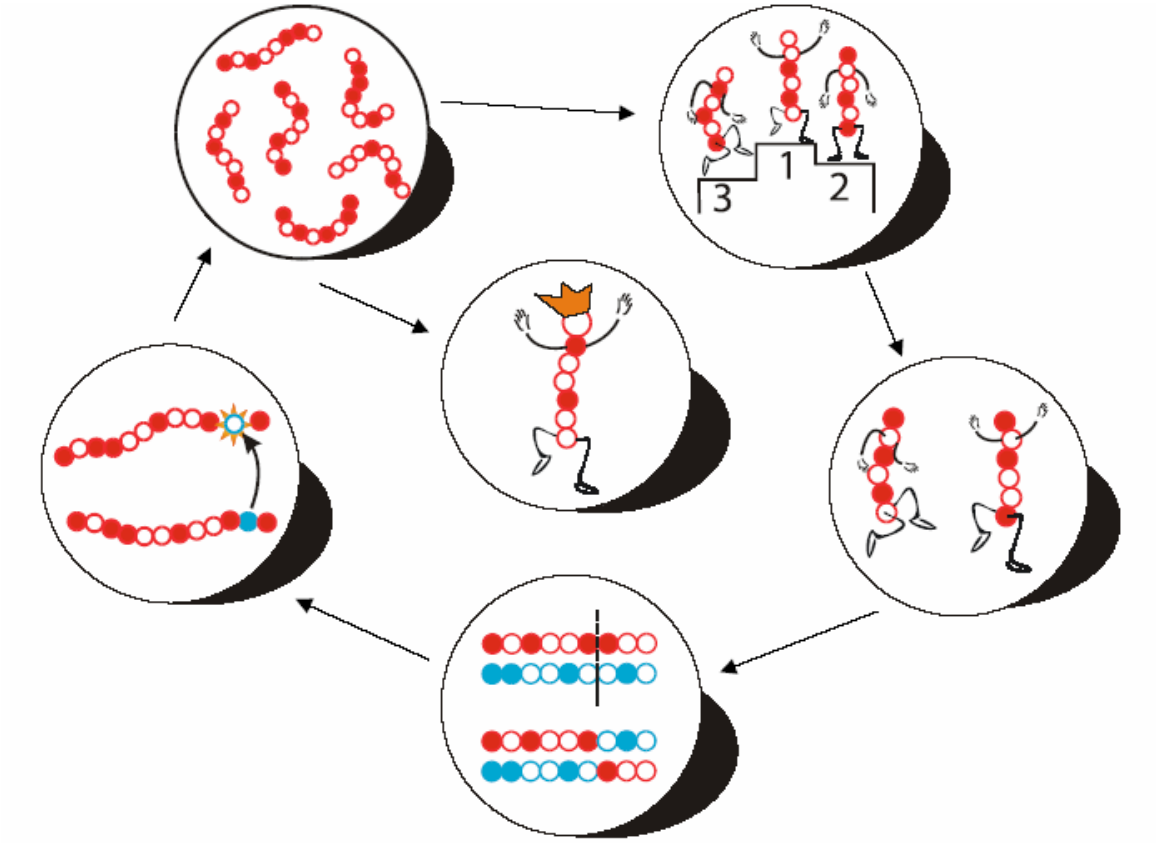


Figure 2.8 Processes of genetic algorithm [27]

2.4. Neuro Fuzzy Controllers

A neural network can model a dynamic plant by means of a nonlinear regression in the discrete time domain. If we have data, or can learn data from a simulation (training) then we can build Artificial Neural networks (ANNs) by specifying the architecture and learning algorithm. The result is a network, with adjusted weights, which approximates the plant. Conversely, if we have a knowledge expressed in linguistic rules, we can build a fussy interface system (FIS) by specifying the fuzzy rules; fuzzy membership functions. However one cannot learn the rules and memberships themselves separately from data [26].

While the learning capability is an advantage from the viewpoint of fuzzy interface system (FIS), the formation of linguistic rule base will be advantageous from the viewpoint of Artificial Neural Network. The two algorithms are combined in Neuro-Fuzzy system in order to integrate their advantages. In the simplest way, Neuro Fuzzy systems utilize Artificial Neural Network (ANN) learning algorithm to determine the

FIS membership functions from the training data. Once the FIS parameters are determined, ANN goes to the background. The rules are usually determined by fuzzy clustering algorithms.

Neuro Fuzzy systems have many design approaches to implement, Adaptive Neuro Fuzzy Inference System (ANFIS), is one of these approaches. In this thesis we have adopted (ANFIS) to build up the Neuro fuzzy system for our applications .The structure of ANFIS is shown in Figure 2.9.

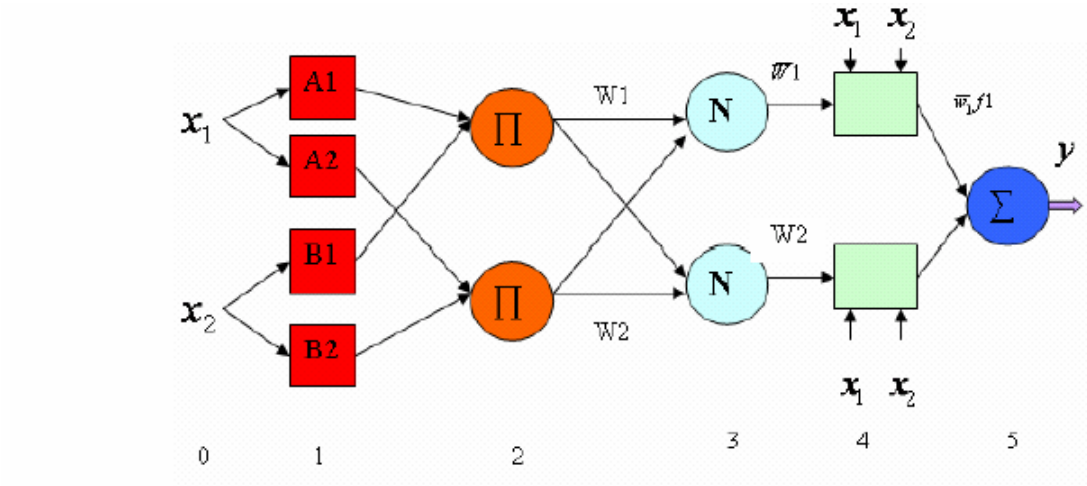


Figure 2.9 Structures of ANFIS

ANFIS [26] implements a Takagi Sugeno FIS and has five structural layers as shown in Figure 2.9. The first hidden layer is for fuzzification of the input variables, as shown in Figure 2.10. The second layer calculates the firing strength of each rule. The third hidden layer normalizes the rule strengths followed by the fourth hidden layer where the consequent parameters of the rule are determined. Output layer computes the overall input as the summation of all incoming signals.

The detailed mathematical description of ANFIS, Learning algorithm, clustering algorithms are described in the following subsections:

2.4.1 Mathematical Description of ANFIS

Consider a first-order Tasaki-Sugeno-Kang (TSK) [26] fuzzy inference system that consists of two rules

Rule 1: If X is A_1 and Y is B_1 then $f_1 = p_1 x + q_1 y + r_1$

Rule 2: If X is A_2 and Y is B_2 then $f_2 = p_2 x + q_2 y + r_2$

Where the parameters (p_i , q_i , r_i) are called the consequence parameters

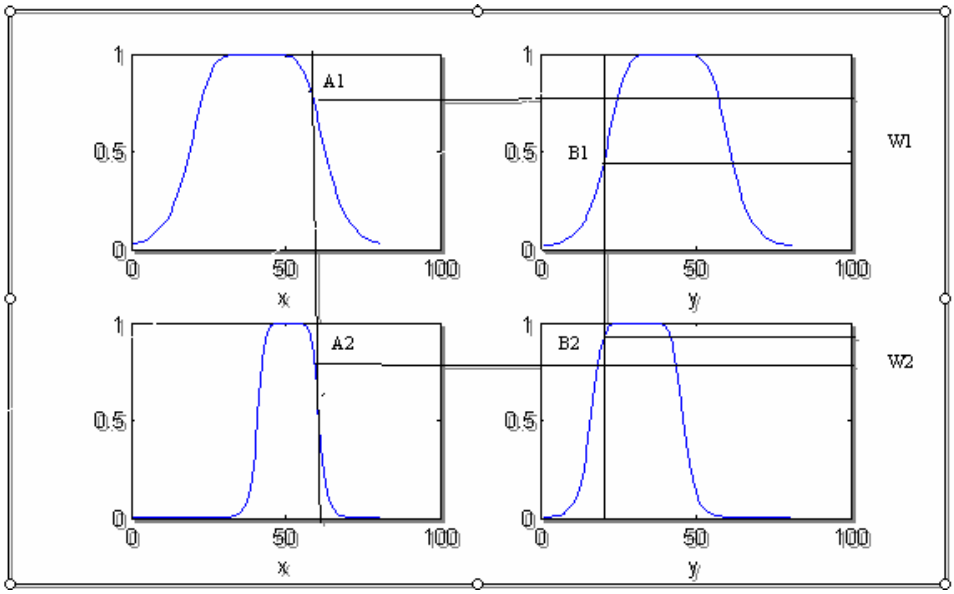


Figure 2.10 Fuzzification of the input variables

The subsequence functions of each layer in ANFIS are described as follows:

Layer1: Each node in this layer generates the membership grade of a linguistic label (large, small…). Figure 2.10 demonstrate this process. The function of i^{th}-node is described by:

$$O_i = \mu_{Ai} = \frac{1}{1 + \left| \frac{x - c_i}{a_i} \right|^{2b}} \tag{2.9}$$

Where x is the input to node A_i and the parameters (a, b, c) in this layer are called premise parameters. These parameters are adjusted during the learning algorithm [26]. The effect of changing these parameters are shown in Figure 2.11

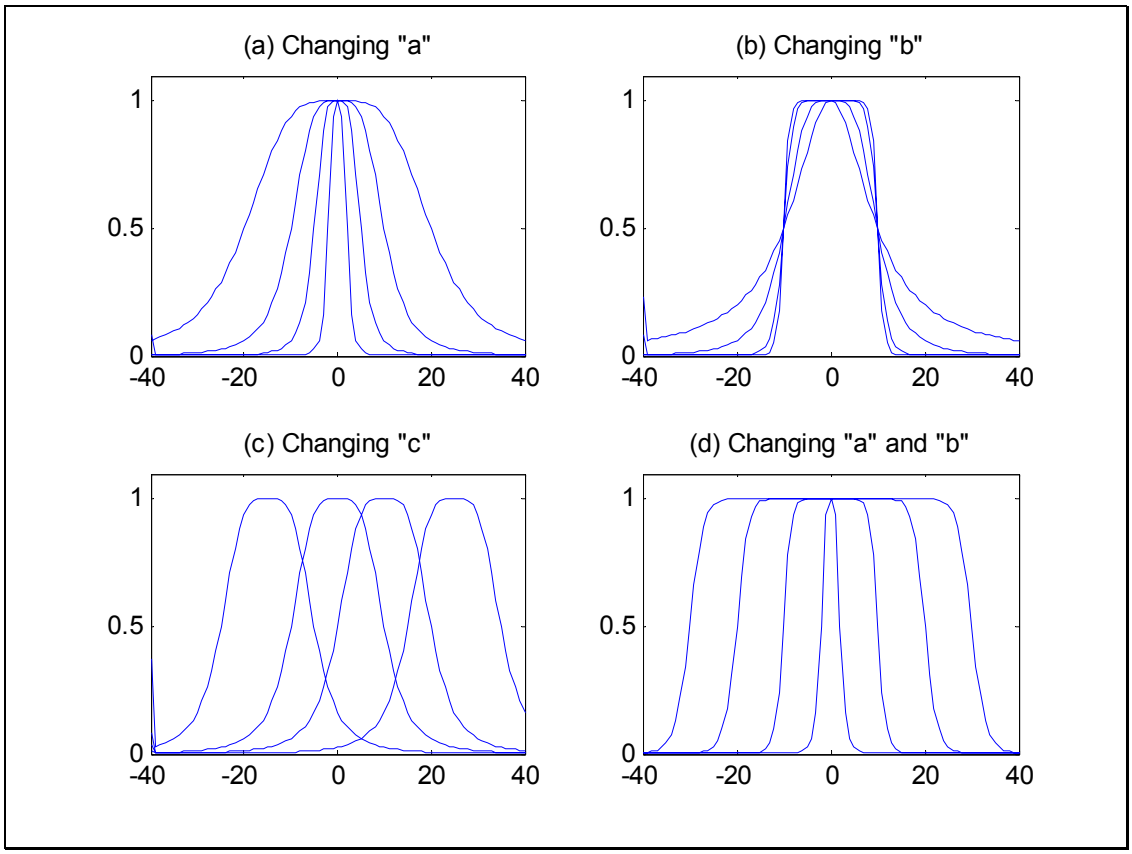


Figure 2.11 Effect of changing premise parameters a, b, c

Layer2: Each node in this layer calculates the firing strength of each rule via multiplication [26].

$$O_i^2 = w_i = \mu_{A_i}(x)\mu_{B_i}(y) \tag{2.10}$$

Layer3: The i^{th}-node in this layer calculates the ratio of the firing i^{th} rule's firing strength to the sum of all rules firing strength [26] (Normalizing the firing strength of each rule).

$$O_i^3 = \overline{w}_i = \frac{w_i}{\Sigma w} \tag{2.11}$$

Layer4: The i^{th} node in this layer calculates the following function [26]

$$O_i^1 = \overline{w}_i f_i = w_i(p_i x + q_i y + r_i) \tag{2.12}$$

Where the parameters $\{p_i, q_i, r_i\}$ are called the consequence parameters.

Layer5: The single node of this layer computes the overall output.

$$O_i = overalloutput = \Sigma \overline{w}_i f_i \tag{2.13}$$

2.4.2 Learning Algorithm

ANFIS uses back propagation learning to determine premise parameters (to learn the parameters related to membership functions) and least mean square estimation to determine the consequent parameters. Each step in the learning procedure has got two parts: In the first part the input patterns are propagated, and the optimal consequent parameters are estimated by an iterative least mean square procedure, while the premise parameters are assumed to be fixed for the current cycle through the training set. In the second part, the patterns are propagated again, and in this epoch, back propagation is also used to modify the premise parameters, while the consequent parameters remain fixed [26].

2.4.3 Clustering

The purpose of clustering is to distil natural groupings of data from a large data set, producing a concise representation of a system's behaviour. The clustering of I/O data produces a set of cluster centres, and each cluster centre acts as a prototypical data point that describes a characteristic mode of the system, and can be considered as the nucleus of a fuzzy if-then rule. In that way partitioning of the inputs and determination of the initial minimal rule base for ANFIS can be performed [26].

CHAPTER 3: KINEMATICS AND DYNAMICS OF UNDERACTUATED MANIPULATOR

A two link manipulator is considered as shown in Figure 1.1. The system represents an underactuated manipulator with one active joint at the shoulder. The detailed description of manipulator design and experimental setup are presented in Appendix III. In this Chapter, we present the forward kinematics of the manipulator first, followed by inverse kinematics, then we discuss the dynamics of underactuated manipulator is derived and presented.

3.1. Forward Kinematics

The forward kinematics problem is to determine the position and orientation of end effecter, given the values of joint variables of manipulator $(\theta_1, \theta_2 \ldots etc)$ [32]. Using Denavit-Hartenberc representation, the transformation matrix for two-link planar manipulator from world frame to the end effecter is presented in equation 3.1.

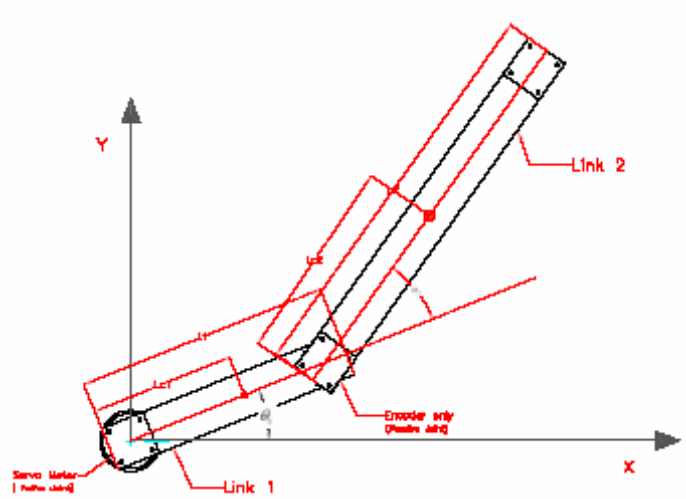


Figure 3.1 Two -link manipulator with one active joint at shoulder "Pendubot"

$$T_0^2 = \begin{bmatrix} \cos(\theta_1+\theta_2) & \sin(\theta_1+\theta_2) & 0 & l_1\cos(\theta_1)+l_2\cos(\theta_1+\theta_2) \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 & l_1\ \sin(\theta 1)+l_2\sin(\theta_1+\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.1)$$

From the fourth column of T_0^2 and as shown in Figure 3.2, the position of end effecter in Cartesian coordinates is determined as follows:

$$x = l_1\cos(\theta_1)+l_2\cos(\theta_1+\theta_2) \quad (3.2)$$

$$y = l_1\sin(\theta_1)+l_2\sin(\theta_1+\theta_2) \quad (3.3)$$

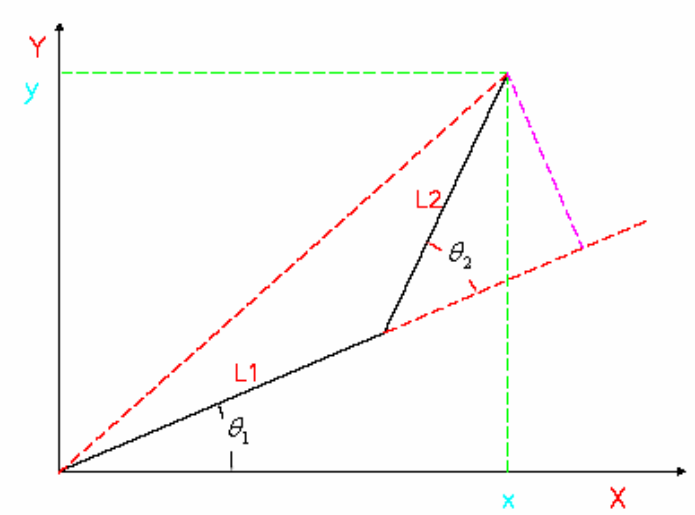


Figure 3.2 Schematic diagrams for two-link planar manipulator

3.2. Inverse Kinematics

On the opposite of forward kinematics, Inverse kinematics is concerned with finding the joint variable $(\theta_1, \theta_2 \ldots etc)$ of robot or manipulator in terms of position and orientation of end effecter. Using the geometric approach [32], the inverse kinematics for two-link manipulator is found as follows:

Consider Figure 3.2. Using the Law of Cosine we see that:

$$l_1^2 + l_2^2 + 2l_1l_2\cos\theta_2 = x^2 + y^2 \quad (3.4)$$

$$\cos\theta_2 = \frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1 l_2} = D \tag{3.5}$$

Then

$$\sin\theta_2 = \pm\sqrt{1 - D^2} \tag{3.6}$$

And

$$\theta_2 = \frac{\tan^{-1}\sqrt{1 - D^2}}{D} \tag{3.6}$$

The angle θ_1 is evaluated by:

$$\theta_1 = \tan^{-1}(y/x) - \tan^{-1}\left(\frac{l_2 \sin(\theta_2)}{l_1 + l_2 \cos\theta_2}\right) \tag{3.7}$$

3.3. Manipulator Dynamics

From Manipulator kinematics, we proceed with derivation of manipulator dynamics as follows:

Let, v_{c1} v_{c2} be the linear velocity for the center of link one and link two respectively .These linear velocities are related to joint velocities $\dot{\theta} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$, by the following equation:

$$v_{c1} = J_{vc1} \cdot \dot{\theta} \tag{3.8}$$

$$v_{c2} = J_{vc2} \cdot \dot{\theta} \tag{3.9}$$

Where J_{vc1} , J_{vc2} are the Jacobian matrices defined by:

$$J_{vc1} = \begin{bmatrix} -l_{c1}\sin(\theta_1) & 1 \\ l_{c1}\cos(\theta_1) & 0 \\ 0 & 0 \end{bmatrix} \tag{3.10}$$

$$J_{vc2} = \begin{bmatrix} -l_1 \sin(\theta_1) - l_{c2} \sin(\theta_1 + \theta 2) & -l_{c2} \sin(\theta_1 + \theta_2) \\ -l_1 \cos(\theta_1) - l_{c2} \cos(\theta_1 + \theta 2) & -l_{c2} \cos(\theta_1 + \theta_2) \\ 0 & 0 \end{bmatrix} \quad (3.11)$$

The total Kinetic energy of the manipulator equals to:

$$K = \tfrac{1}{2} \dot{\theta} \left\{ m_1 J_{vc1}^T J_{vc1} + m_2 J_{vc2}^T J_{vc2} \right\} \dot{\theta} \quad (3.12)$$

The potential energy of link one equals

$$V_1 = m_1 g l_{c1} \sin \theta_1 \quad (3.13)$$

And potential energy of link two equals

$$V_2 = m_2 g (l_1 \sin \theta_1 + l_{c2} \sin(\theta_1 + \theta_2)) \quad (3.14)$$

The total potential energy of manipulator equals

$$V = V_1 + V_2 = m_1 l_{c1} g \sin \theta_1 + m_2 g (l_1 \sin \theta_1 + l_{c2} \sin(\theta_1 + \theta_2)) \quad (3.15)$$

Then the Langrange function equals [32]

$$L = K - V \quad (3.16)$$

The equation of motion for link one equals [32]

$$\tau_1 = \frac{d}{dt} \frac{\partial L}{\partial \dot{\theta}_1} - \frac{\partial L}{\partial \theta_1} \quad (3.17)$$

and for link two

$$\tau_2 = \frac{d}{dt} \frac{\partial L}{\partial \dot{\theta}_2} - \frac{\partial L}{\partial \theta_2} \quad (3.18)$$

Where τ_1 and τ_2 are the torques applied to joint one and joint two respectively.

By carrying out the above differentiation, using the geometric identities and definition of link moment of inertia, making the necessary modification and noticing joint two is

passive joint, i.e. $\tau_2 = 0$, the dynamics of the underactuated manipulator can be described by the following equations:

$$d_{11}\ddot{\theta}_1 + d_{12}\ddot{\theta}_2 + h_1 + \varphi_1 = \tau_1 \tag{3.19}$$

$$d_{12}\ddot{\theta}_1 + d_{22}\ddot{\theta}_2 + h_2 + \varphi 2 = 0. \tag{3.20}$$

Where:

$$d_{11.} = m_1 l_{c1}^2 + m_2\left(l_1^2 + l_{c2}^2 + 2l_1 l_{c2}\cos(\theta_1)\right) + I_1 + I_2 \tag{3.21}$$

$$d_{12.} = d_{21} = m_2\left(l_{c2}^2 + l_1 l_{c2}\cos(\theta_2)\right) + I_2 \tag{3.22}$$

$$d_{22.} = m_2 l_{c2}^2 + I_2 \tag{3.23}$$

$$h_1 = -m_2 l_{c2}\sin(\theta_2)\dot{\theta}_2^2 - 2m_2 l_1 l_{c2}\sin(\theta_2)\dot{\theta}_2\dot{\theta}_1 \tag{3.24}$$

$$h_2 = -m_2 l_1 l_{c2}\sin(\theta_2)\dot{\theta}_1^2 \tag{3.25}$$

$$\varphi_1 = (m_1 l_{c1} + m_2 l_1)g\cos(\theta_1) + m_2 l_{c2} g\cos(\theta_1 + \theta_2) \tag{3.26}$$

$$\varphi_2 = m_2 l_{c2} g\cos(\theta_1 + \theta_2) \tag{3.27}$$

The parameters h_1 and $h2$ represent Coriolis constants, while the parameters φ_1 , φ_2 contain the gravity terms and they are zeros when the manipulator is in horizontal plane.

To present equation 3.19 and 3.20 in a simpler form, let's define the following parameters [12]:

$$\alpha_1 = m_1 l_{lc1}^2 + m_2 l_1^2 + I_1 \text{ , } \alpha_2 = m_2 l_{lc2}^2 + I_2 \text{ , } \alpha_3 = m_2 l_1 l_{lc2} \text{ , } \alpha_4 = m_2 l_{c1} + m_2 l_1 \text{ , } \alpha_5 = m_2 l_{c2}$$

Then, equations 3.19 and 3.20 reduce to:

$$(\alpha_1+\alpha_2+2\alpha_3\cos(\theta_2))\ddot{\theta}_1+(\alpha_2+\alpha_3\cos(\theta_2))\ddot{\theta}_2-\alpha 3\sin(\theta_2)\dot{\theta}_2^2-2\alpha_3\sin(\theta_2)\dot{\theta}_1\dot{\theta}_2+\alpha_4 g\cos(\theta_1)+\alpha_5 g\cos(\theta_1+\theta_2)=\tau_1 \tag{3.28}$$

$$\alpha_2\ddot{\theta}_2+(\alpha_2+\alpha_3\cos(\theta_2))\ddot{\theta}_1-\alpha 3\sin(\theta_2)\dot{\theta}_1^2+\alpha_5 g\cos(\theta_1+\theta_2)=0_1 \tag{3.29}$$

In matrices form, one can express the equations of motion of the two link underactuated manipulator as follows:

Let the matrix D equals [12]

$$D(\theta)\ddot{\theta}+C(\theta,\dot{\theta})\dot{\theta}+G=\tau \tag{3.30}$$

Where

$$D=\begin{bmatrix}(\alpha_1+\alpha_2+2\alpha_3\cos(\theta_2)) & (\alpha_2+\alpha_3\cos(\theta_2))\\ (\alpha_2+\alpha_3\cos(\theta_2)) & \alpha_2\end{bmatrix}$$

$$C=\begin{bmatrix}-\alpha 3\sin(\theta_2)\dot{\theta}_2 & -\alpha 3\sin(\theta_2)\dot{\theta}_2-\alpha 3\sin(\theta_2)\dot{\theta}_1\\ -\alpha 3\sin(\theta_2)\dot{\theta}_1 & 0\end{bmatrix}$$

$$G=\begin{bmatrix}\alpha_4 g\cos(\theta_1)+\alpha_5 g\cos(\theta_1+\theta_2)\\ \alpha_5 g\cos(\theta_1+\theta_2)\end{bmatrix}$$

and

$$\tau=\begin{bmatrix}\tau_1\\ 0\end{bmatrix}.$$

CHAPTER 4: CONTROLLABILITY OF UNDERACTUATED MANIPULATOR

4.1. Controllability in Gravity Domain

To verify the controllability of the manipulator in gravity domain at the upright unstable position $(\theta_1 = \pi/2, \theta_2 = 0)$, we will linearize the system at that position and check its controllability as follows [12]:

Let's rewrite equations 3.28 and 3.29 in the following manner:

$$\ddot{\theta}_1 = \frac{1}{\alpha_1\alpha_2 - \alpha_3^2\cos^2(\theta_2)}\begin{bmatrix}\alpha_2\alpha_3\sin(\theta_2)(\dot{\theta}_1+\dot{\theta}_2)^2 + \alpha_3^2\cos(\theta_2)\sin(\theta_2)\dot{\theta}_1^{\,2} - \\ \alpha_2\alpha_4 g\cos(\theta_1) + \alpha_3\alpha_5 g\cos(\theta_2)\cos(\theta_1+\theta_2) + \alpha_2\tau_1\end{bmatrix} \tag{4.1}$$

$$\ddot{\theta}_2 = \frac{1}{\alpha_1\alpha_2 - \alpha_3^2\cos^2(\theta_2)}\begin{bmatrix}-\alpha_3(\alpha_2+\alpha_3\cos(\theta_2)\sin(\theta_2)(\dot{\theta}_1+\dot{\theta}_2)^2 - \\ (\alpha_1+\alpha_3\cos(\theta_2))\sin(\theta_2)\dot{\theta}_1^{\,2} + \\ (\alpha_2+\alpha_3\cos(\theta_2))(\alpha 4g\cos(\theta_1)-\tau_1) - \\ (\alpha_1+\alpha_3\cos(\theta_2))\alpha_5 g\cos(\theta_1+\theta_2)\end{bmatrix} \tag{4.2}$$

Also let's define $x_1 = \theta_1, x_2 = \dot{\theta}_1, x3 = \theta_2, x_4 = \dot{\theta}_2$, then the system states are expressed as

$$x = [x_1 \quad x_2 \quad x_3 \quad x_4]$$

To linearize the state space equations at the upright position, we differentiate equations (4.1) and (4.2) with respect to the states and evaluate them at unbalance position $\left[\theta_1 = \pi/2 \quad ,\dot{\theta}_1 = 0 \quad \theta_2 = \theta \quad ,\dot{\theta}_2 = 0\right]$. This reveals the following state space equation:

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \dfrac{(\alpha_2\alpha_4 - \alpha_3\alpha_5)g}{\alpha_1\alpha_2 - \alpha_3^2} & 0 & \dfrac{(\alpha_2\alpha_4 - \alpha_3\alpha_5)g}{\alpha_1\alpha_2 - \alpha_3^2} & 0 \\ 0 & 0 & 0 & 1 \\ \dfrac{\alpha_5 g(\alpha 1 + \alpha 3) - (\alpha_2 + \alpha_3)\alpha_4 g}{\alpha_1\alpha_2 - \alpha_3^2} & 0 & \dfrac{(\alpha_1 + \alpha_3)\alpha_5 g}{\alpha_1\alpha_2 - \alpha_3^2} & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ \dfrac{\alpha_2}{\alpha_1\alpha_2 - \alpha_3^2} \\ 1 \\ \dfrac{-\alpha_2 - \alpha_3}{\alpha_1\alpha_2 - \alpha_3^2} \end{bmatrix} \tau_1 \quad (4.3)$$

$$\dot{x} = Ax + B\tau_1 \quad (4.4)$$

Thus we have

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \dfrac{(\alpha_2\alpha_4 - \alpha_3\alpha_5)g}{\alpha_1\alpha_2 - \alpha_3^2} & 0 & \dfrac{(\alpha_2\alpha_4 - \alpha_3\alpha_5)g}{\alpha_1\alpha_2 - \alpha_3^2} & 0 \\ 0 & 0 & 0 & 1 \\ \dfrac{\alpha_5 g(\alpha 1 + \alpha 3) - (\alpha_2 + \alpha_3)\alpha_4 g}{\alpha_1\alpha_2 - \alpha_3^2} & 0 & \dfrac{(\alpha_1 + \alpha_3)\alpha_5 g}{\alpha_1\alpha_2 - \alpha_3^2} & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \dfrac{\alpha_2}{\alpha_1\alpha_2 - \alpha_3^2} \\ 1 \\ \dfrac{-\alpha_2 - \alpha_3}{\alpha_1\alpha_2 - \alpha_3^2} \end{bmatrix}$$

$$AB = \begin{bmatrix} 0 \\ \dfrac{\alpha_2}{\alpha_1\alpha_2 - \alpha_3^2} \\ 0 \\ \dfrac{-\alpha_2 - \alpha_3}{\alpha_1\alpha_2 - \alpha_3^2} \end{bmatrix} \qquad A^2B = \begin{bmatrix} 0 \\ \dfrac{g(\alpha_4\alpha_2^2 + \alpha_5\alpha_3^2)}{\alpha_1\alpha_2 - \alpha_3^2} \\ 0 \\ \dfrac{-g(\alpha_4\alpha_2^2 + \alpha_2\alpha_4\alpha_3 + \alpha_5\alpha_1\alpha_3 + \alpha_5\alpha_3^2)}{\alpha_1\alpha_2 - \alpha_3^2} \end{bmatrix}$$

$$A^3B = \begin{bmatrix} \dfrac{g(\alpha_4\alpha_2^2 + \alpha_5\alpha_3^2)}{\alpha_1\alpha_2 - \alpha_3^2} \\ 0 \\ \dfrac{-g(\alpha_4\alpha_2^2 + \alpha_2\alpha_4\alpha_3 + \alpha_5\alpha_1\alpha_3 + \alpha_5\alpha_3^2)}{\alpha_1\alpha_2 - \alpha_3^2} \\ 0 \end{bmatrix}$$

The matrix [B AB AB^2 AB^3] is nonsingular (rank=4), and its determinant equals:

$$\det\left(\begin{bmatrix} B & AB & AB^2 & AB^2 \end{bmatrix}\right) = \frac{g\ \alpha_5^2\alpha_3^2}{(\alpha_1\alpha_2 - \alpha_3^2)^4}$$

Thus, the controllability conditions of the two-link underactuated manipulator (Pendubot) at top position are verified and satisfied [24].

4.2. Controllability in Horizontal Plane

In the horizontal plane, the components of potential energy are deleted from equation from equations 3.28 and 3.29.Thus, the equations of motion for the underactuated manipulator in horizontal plane, are expressed as follows:

$$(\alpha_1 + \alpha_2 + 2\alpha_3 \cos(\theta_2))\ddot{\theta}_1 + (\alpha_2 + \alpha_3 \cos(\theta_2))\ddot{\theta}_2 - \alpha 3 \sin(\theta_2)\dot{\theta}_2^2 - 2\alpha_3 \sin(\theta_2)\dot{\theta}_1\dot{\theta}_2 = \tau_1 \quad \{4.5)$$

$$\alpha_2\ddot{\theta}_2 + (\alpha_2 + \alpha_3 \cos(\theta_2))\ddot{\theta}_1 - \alpha 3 \sin(\theta_2)\dot{\theta}_1^2 = 0_1 \quad (4.6)$$

Let's rewrite equations 4.5 and 4.6 in the following manner:

$$\ddot{\theta}_1 = \frac{1}{\alpha_1\alpha_2 - \alpha_3^2 \cos^2(\theta_2)}\left[\alpha_2\alpha_3 \sin(\theta_2)(\dot{\theta}_1 + \dot{\theta}_2)^2 + \alpha_3^2 \cos(\theta_2)\sin(\theta_2)\dot{\theta}_1^{\,2} + \alpha_2\tau_1\right] \quad (4.7)$$

$$\ddot{\theta}_2 = \frac{1}{\alpha_1\alpha_2 - \alpha_3^2 \cos^2(\theta_2)}\begin{bmatrix} -\alpha_3(\alpha_2 + \alpha_3 \cos(\theta_2)\sin(\theta_2)(\dot{\theta}_1 + \dot{\theta}_2)^2 - \\ (\alpha_1 + \alpha_3 \cos(\theta_2))\sin(\theta_2)\dot{\theta}_1^{\,2} - (\alpha_2 + \alpha_3 \cos(\theta_2))\tau_1) \end{bmatrix} \quad (4.8)$$

Differentiating equations 4.7 and 4.8 with respect to the states and evaluating them at any arbitrary position $\left[\theta_1 = \theta_{ar1}, \dot{\theta}_1 = 0 \theta_2 = \theta_{ar2}, \dot{\theta}_2 = 0\right]$, reveals the following state space equation:

$$\dot{x} = A\,x + B\,u$$

Where

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ N_1 \\ 0 \\ N_2 \end{bmatrix} \quad (4.9)$$

The terms N1 and N2 are constants resulted from evaluating the partial differential equation at arbitrary positions $[\theta_1 = \theta_{ar1} \quad \theta_2 = \theta_{ar2}]$.

$$N_1 = \frac{\alpha_2}{\alpha_1\alpha_2 - \alpha_3^2(\cos\theta_{arb})^2} \quad , \quad N_2 = \frac{\alpha_2 + \alpha_3 os\,\theta_{arb}}{\alpha_1\alpha_2 - \alpha_3^2(\cos\theta_{arb})^2}$$

Therefore

$$A = \begin{bmatrix} 0 & 1 & 0 & \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ N_1 \\ 0 \\ N_2 \end{bmatrix}$$

$$AB = \begin{bmatrix} N_1 \\ 0 \\ N_2 \\ 0 \end{bmatrix} \qquad A^2B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad A^3B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

and

$$\begin{bmatrix} B & AB & A^2B & A^3B \end{bmatrix} = \begin{bmatrix} 0 & N_1 & 0 & 0 \\ N_1 & 0 & 0 & 0 \\ 0 & N_2 & 0 & 0 \\ N_2 & 0 & 0 & 0 \end{bmatrix}$$

It is clear that the matrix $\begin{bmatrix} B & AB & A^2B & A^3B \end{bmatrix}$ which determines the controllability of the system is singular (rank=2 <4). Thus, as per [3] the two link manipulator is uncontrollable in the absence of gravity terms and friction terms. Therefore positioning the underactuated manipulator at desired position is done with aid of holding brakes.

The same result can be reached by the analysis of total energy in the system as follows:

Let the matrix D introduced in equation 3.30 be [12]

Where

$$D = \begin{bmatrix} (\alpha_1 + \alpha_2 + 2\alpha_3 \cos(\theta_2)) & (\alpha_2 + \alpha_3 \cos(\theta_2)) \\ (\alpha_2 + \alpha_3 \cos(\theta_2)) & \alpha_2 \end{bmatrix} \tag{4.10}$$

and let the matrix C equals:

$$C = \begin{bmatrix} -\alpha 3 \sin(\theta_2)\dot{\theta}_2 & -\alpha 3 \sin(\theta_2)\dot{\theta}_2 - \alpha 3 \sin(\theta_2)\dot{\theta}_1 \\ -\alpha 3 \sin(\theta_2)\dot{\theta}_1 & 0 \end{bmatrix} \tag{4.11}$$

Then the equations of motion of the manipulator in horizontal plan can be expressed as follows:

$$D(\theta)\ddot{\theta} + C(\theta, \dot{\theta}) = \tau \tag{4.12}$$

The kinetic energy of manipulator can be expressed as:

$$K = \frac{1}{2}\dot{\theta}^T D(\theta)\dot{\theta} \tag{4.13}$$

The total energy E of the manipulator plane includes the kinetic energy of manipulator only:

$$E = K = \frac{1}{2}\dot{\theta}^T D(\theta)\dot{\theta} \tag{4.14}$$

Differentiating equation 4.14 with respect to time, the change of manipulator energy is obtained as from:

$$\dot{E} = \dot{\theta}^T D(\theta)\ddot{\theta} + \frac{1}{2}\dot{\theta}^T \dot{D}(\theta)\dot{\theta} \tag{4.15}$$

$$\dot{E} = \dot{\theta}^T (-C(\theta, \dot{\theta})\dot{\theta} - \tau) + \frac{1}{2}\dot{\theta}^T \dot{D}(\theta)\dot{\theta} \tag{4.16}$$

$$\dot{E} = \dot{\theta}^T \tau + \frac{1}{2}\dot{\theta}^T (\dot{D}(\theta) - 2(-C(\theta, \dot{\theta}))\dot{\theta} \tag{4.17}$$

The matrix

$$(\dot{D}(\theta) - 2(-C(\theta,\dot{\theta})) \quad = \begin{bmatrix} 0 & \alpha_3 \sin(\theta_2)(2\dot{\theta}_1 + \dot{\theta}_2) \\ \alpha_3 \sin(\theta_2)(2\dot{\theta}_1 + \dot{\theta}_2) & 0 \end{bmatrix}$$

is a skew matrix thus

$$\frac{1}{2}\dot{\theta}^T (\dot{D}(\theta) - 2(-C(\theta,\dot{\theta})) \quad = 0 \tag{4.18}$$

As a result

$$\dot{E} = \dot{\theta}^T \tau = \dot{\theta}_1{}^T \tau_1 \tag{4.19}$$

$$E = \int_0^t \dot{\theta}_1{}^T \tau_1 \, dt \quad = E(t) - E(0) \tag{4.20}$$

Since the manipulator does not have any potential energy, the manipulator should reach the desired position with zero energy (zero kinetic energy). However to reach the desired position of link one and achieve the dynamic coupling, torque should be applied to joint one (active joint) to move it to the desired position. Thus, from (4.20), the energy is accumulated in the system without any dissipation of energy (friction is neglected). Therefore, the system can not reach zero kinetic energy in the absent of gravity force and friction .As result holding brakes are used to control the system and achieve the desired position.

CHAPTER 5: SWINGING UP CONTROLLERS

5.1. Feedback Linearization Technique

As discussed in chapter three, the equations of motion for two link underactuated manipulator are given by:

$$d_{11}\ddot{\theta}_1 + d_{12}\ddot{\theta}_2 + h_1 + \varphi_1 = \tau_1 \tag{5.1}$$

$$d_{12}\ddot{\theta}_1 + d_{22}\ddot{\theta}_2 + h_2 + \varphi 2 = 0 \tag{5.2}$$

Solving equation 5.1 for the angular acceleration of link two gives:

$$\ddot{\theta}_2 = \frac{-h_2 - d_{12}\ddot{\theta}_1 - \varphi_2}{d_{22}}. \tag{5.3}$$

Substituting (5.3) into (5.1) yields:

$$\bar{d}_{11}\ddot{\theta}_1 + \bar{h}_1 + \bar{\varphi}_1 = \tau_1 \tag{5.4}$$

Where:

$$\bar{d}_{11} = \left(d_{11} - \frac{d_{12}}{d_{22}}\right), \quad \bar{h}_1 = \left(h_1 - \frac{d_{12}}{d_{22}}h_2\right), and \quad \bar{\varphi}_1 = \left(\varphi_1 - \frac{d_{12}}{d_{22}}\varphi_2\right)$$

To cancel the nonlinear terms in equation 5.4, let's define the input torque as follows:

$$\tau_1 = \bar{d}_{11}v_1 + \bar{h}_1 + \bar{\varphi}_1 \tag{5.5}$$

Where v_1 is the linear controller to be designed.

Then the dynamic equations of manipulator become

$$\ddot{\theta}_1 = v_1 \tag{5.6}$$

$$d_{22}\ddot{\theta}_2 + h_2 + \varphi_2 = -d_{12}v_1 \tag{5.7}$$

Let v_1 be a PID controller and let it be defined as:

$$v_1 = \ddot{\theta}_1^d + K_d(\dot{\theta}_1^d - \dot{\theta}_1) + K_p(\theta_1^d - \theta_1) \tag{5.8}$$

Where

$\ddot{\theta}_1^d$,,$\dot{\theta}_1^d$,θ_1^d are the desired angular acceleration, velocity and position of first link respectively.

Substituting equation 5.8 into 5.6 reveals:

$$\ddot{\theta}_1^d - \ddot{\theta}_1 + K_d(\dot{\theta}_1^d - \dot{\theta}_1) + K_p(\theta_1^d - \theta_1) = \ddot{e} + K_d\dot{e} + K_p e = 0 \tag{5.9}$$

Let's design the controller so that the response of first link exhibits a settling time of 0.33 second, then the gains K_d and K_p take the values:

$K_d = 27.5$

$K_p = 350$

By selecting the gains K_d and K_p, the first link can easily reach the desired position, i.e., $\theta_1 = \pi/2$, however second link can not reach the desired position $\theta_2 = 0$ without pumping energy to the system Figure 5.1 demonstrates this result.

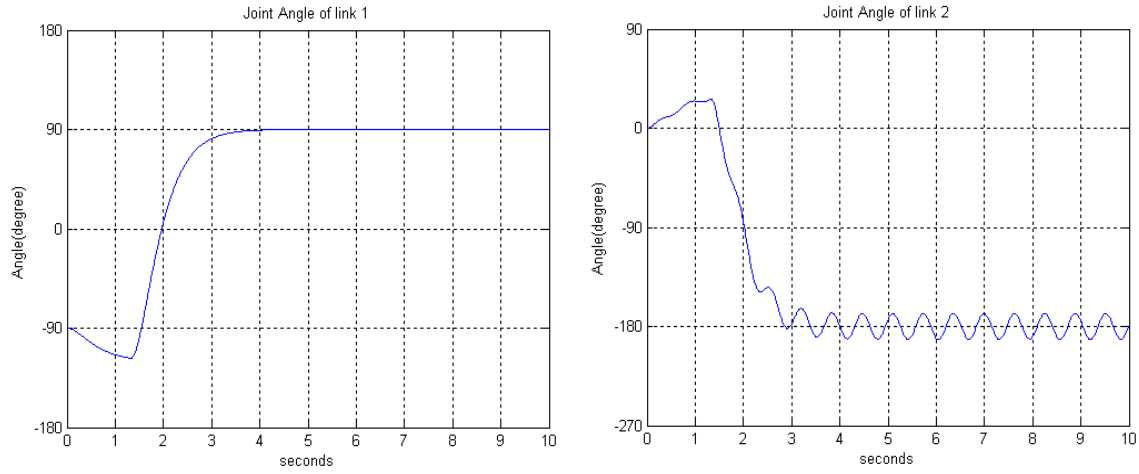


Figure 5.1 Joint angle of link one and link two without pumping of energy

Consequently, pumping sufficient amount energy to the mechanical system is required to keep the first link at the desired position, while steering the second link to homoclinic orbit [13 described by:

$$\frac{1}{2}\alpha_2\dot{\theta}_2^2 = \alpha_5 g(1-\cos(\theta_2)) \tag{5.10}$$

By trials and error, a typical input torque of -0.0435 N.m. for 0.5 second is sufficient to attract the second link to the homoclinic and balance the manipulator as shown in Figure 5.2 .

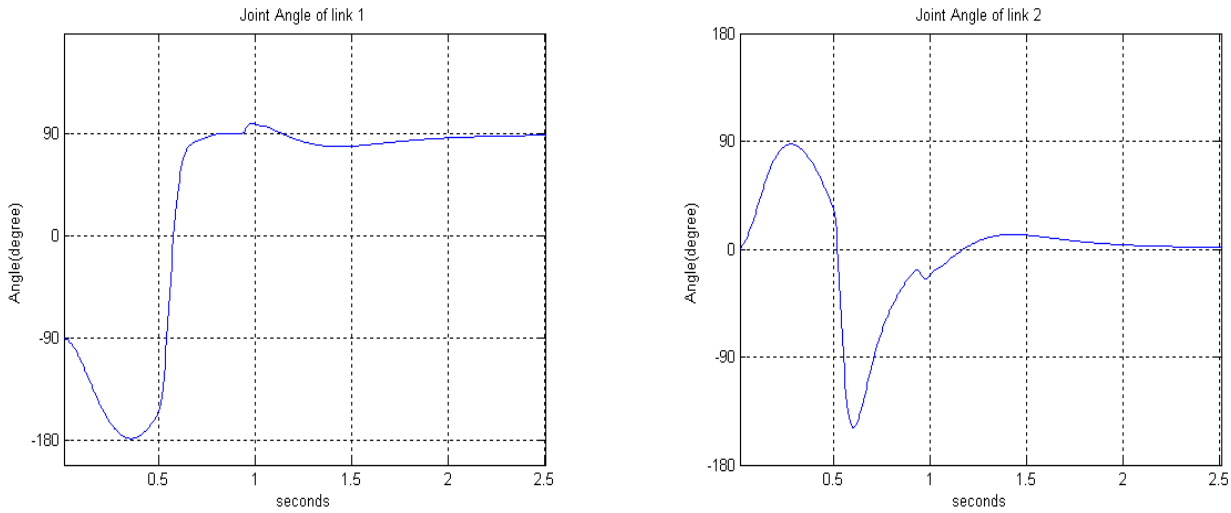


Figure 5.2 Joint angle of link one and link two with pumping of energy

5.2. Sliding Mode Controller

The sliding mode controller is derived using the state dynamics described by equation (5.4). In a state space form where $\tilde{x}_1 = \theta_1$ and $\dot{\theta}_1 = \dot{x}_1 = \tilde{x}_2$, then:

$$\begin{aligned} \dot{\tilde{x}}_1 &= \tilde{x}_2 \\ \dot{\tilde{x}}_2 &= f_1(\tilde{x}) + b_1\tau_1 \end{aligned} \tag{5.11}$$

Where

$$f_1(\bar{x}) = \frac{(\bar{c}_{11}\dot{\theta}_1 + \bar{c}_{12}\dot{\theta}_2 + \bar{\varphi}_1)}{\bar{d}_{11}},\ b_1 = \frac{1}{\bar{d}_{11}}\ ,\ \bar{d}_{11} = \left(d_{11} - \frac{d_{12}\,d_{21}}{d_{22}}\right)\ ,\bar{\varphi}_1 = \left(\varphi_1 - \frac{d_{12}}{d_{22}}\varphi_2\right)$$

$$\bar{c}_{11} = \left(c_{11} - \frac{d_{12}\,c_{21}}{d_{22}}\right)\ ,\bar{c}_{12} = c_{12}$$

Consider a sliding surface of the form [16]:

$$S = \lambda_1(\theta_1 - \theta_1^d) + (\dot{\theta}_1 - \dot{\theta}_1^d) = \lambda\tilde{x}_1 + \tilde{x}_2 \tag{5.12}$$

The goal is to choose the scalar λ such that the system restriction on the surface (5.12) is of the desired stable characteristics.

Consider a Lyapunov function:

$$V_l = \frac{1}{2}S^2 \tag{5.13}$$

The sliding mode controller must be chosen so that the velocity vectors are directed toward the sliding surface. This is accomplished by guarantying $\dot{V}_l \le 0$ .Therefore,

$$\dot{V}_l \le 0 \tag{5.14}$$

$$\dot{V}_l = S.\dot{S} = S(\lambda \cdot \tilde{x}_2 + f_1(\tilde{x}) + b_1\tau_1) \tag{5.15}$$

To satisfy equation 5.14 on may choose the input torque as follows:

$$\tau_1 = \tilde{\tau}_1 - \frac{(\mu\,\mathrm{sgn}(S) + k \cdot S_1)}{b_1} \tag{5.16}$$

Where

$$\tilde{\tau}_1 = -\frac{(\lambda \cdot x_2 + f_1(\tilde{x}))}{b_1} \tag{5.17}$$

and λ, k, μ are constants to be designed for the sliding surface. To design these constants we first need to find their limits.

The maximum value of μ is computed from the maximum torque produced by servo motor as follows:

$$\mu_{\max} = \tau_{\max} = 0.5\ N.m \tag{5.18}$$

Where $\tau_{\max}$ is the maximum torque produced by servo motor.

The maximum value of λ is computed as follows [16]:

$$\lambda_{\max} = \frac{(\tau_{\max} - \bar{d}_{11} \cdot f_{1\ \max})}{\bar{d}_{11} \cdot \dot{\theta}_{1\ \max}} \tag{5.19}$$

Where $f_1 \max$ is defined as:

$$f_1 \max = \frac{(\bar{c}_{11} \max\ \dot{\theta}_1 \max + \bar{c}_{12} \max\ \dot{\theta}_2 \max + \bar{\varphi}_1 \max)}{\bar{d}_{11}} \tag{5.20}$$

The maximum velocity of joint one is the maximum speed of the servo motor; however, practically we do not reach this high velocity. As a result we used the maximum velocity and acceleration during the simulation of classical feedback linearization technique simulation.

$\dot{\theta}_{1\ \max} = 40 \quad rad/\sec, \dot{\theta}_{2\ \max} = 15 \quad rad/\sec, \ddot{\theta}_{1\ \max} = 1400 \quad rad/\sec^2$

$\bar{\varphi}_{1\ \max} = 0.0886 \quad N, |\bar{c}_{11\ \max}| = 0.0091 \quad rad/\sec, |\bar{c}_{12\ \max}| = 0.0077 \quad rad/\sec$

From equation 5.19 and 5.20, the values $f_1 \max$ and $\lambda_{\max}$ are evaluated to be:

$f_{1\ \max} = 266.7$

$\lambda_{\max} = 14.7$

The value of k is selected so that the poles of the following equation are in the left side plane

$$(\ddot{\theta}_1 - \ddot{\theta}_1^d) + k \cdot |S| = 0 \tag{5.21}$$

For our application we selected the values of our parameters to be:

$\lambda = 8, \quad k = 24, \quad \mu = 0.3$

and got stable and satisfactory results as shown in Figure 5.3.

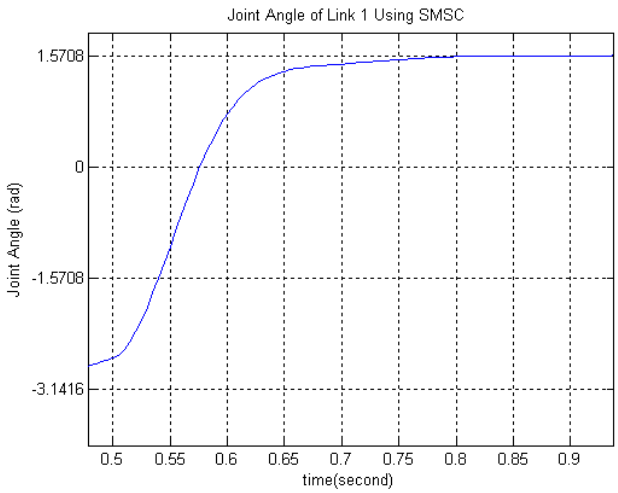


Figure 5.3 Joint Angle of link one using sliding mode swinging up controller

5.3. Neuro Fuzzy Swing up Controller

To build a Neuro fuzzy model for the swing up controller, we have used the states of the fist link of manipulator and the output torque of feedback linearization technique and combine them in one training matrix. Then we have used ANFIS algorithm to build up the Neuro-Fuzzy model. The structure of ANFIS algorithm and the structure of the fuzzy controller, which are developed in this study, are shown inFigure 5.4 and Figure 5.5**.** The membership functions of the inputs are shown in Figure 5.6.

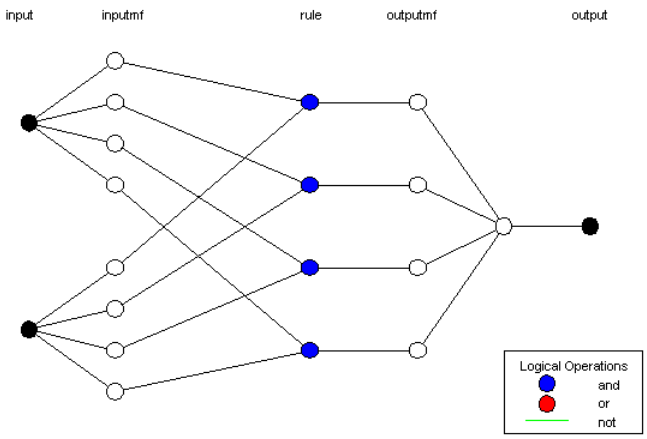


Figure 5.4 Structure of ANFIS algorithm for swing up controller

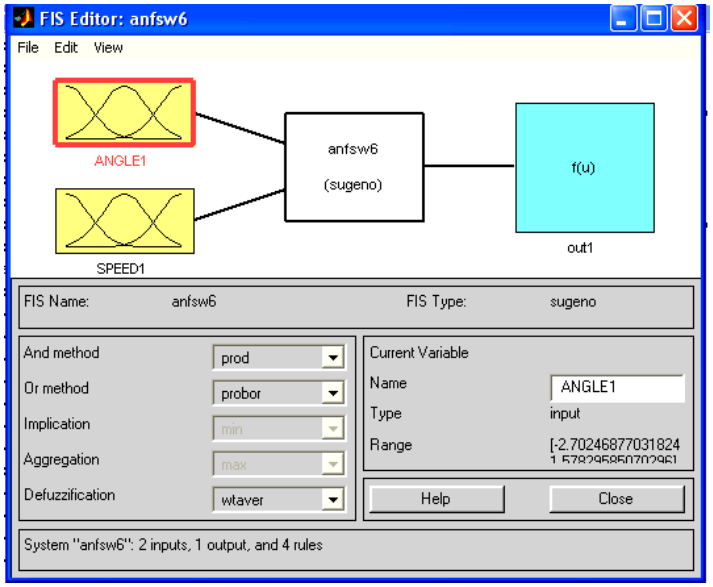


Figure 5.5 Structure of fuzzy controller for swinging up the manipulator at the upright position

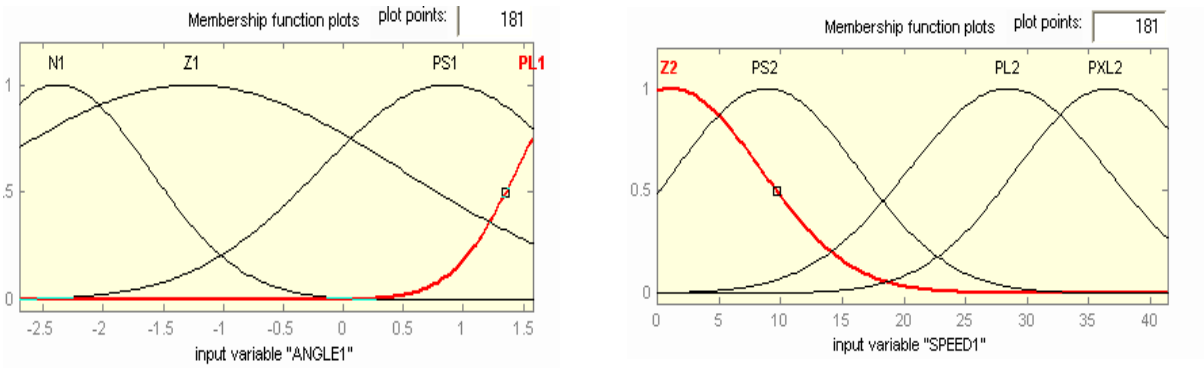


Figure 5.6 Membership functions of fuzzy controller inputs (swing-up)

As a result, a fuzzy logic controller is developed to swing up the manipulator independently. The swinging up process using Neuro Fuzzy method was robust and successful in swinging up the Pendubot. A detailed discussion of the results is presented in chapter seven.

5.4. Tuning PD Gains Using Genetic Algorithm

The manipulator parameters introduced in equation 5.4 were assumed precisely known and the friction was discarded. However the manipulator parameters are not precisely known and constantly changing, also friction does exist. To overcome these constraints, one could use genetic algorithm to select the parameters K_d & K_p of equation 5.8 adaptively.

$$v_1 = \ddot{\theta}_1^d + K_d(\dot{\theta}_1^d - \dot{\theta}_1) + K_p(\theta_1^d - \theta_1)$$

Genetic algorithm select K_d & K_p from specified ranges. The ranges of K_d & K_p are selected so that the poles of characteristic equation 5.7 are always in the left side plane.

$$K_d \in [20\ 28]$$

$$K_p \in [350\ 450]$$

The set of poles for the corresponding range limits are listed below:

$$K_d = 20, , K_p = 350 , p_{1,2} = \text{-}10.0000 \pm 15.8114\text{i}$$

$$K_d = 28, , K_p = 350 , p_{1,2} = \text{-}14.0000 \pm 12.4097\text{i}$$

$$K_d = 20, , K_p = 450 , p_{1,2} = \text{-}10.0000 \pm 18.7083\text{i}$$

$$K_d = 28, , K_p = 450 , p_{1,2} = \text{-}14.0000 \pm 15.9374\text{i}$$

Genetic algorithm is described by a flow chart shown in Figure 5.7. The Genetic algorithm is built inside S-function block using Simulink of Matlab. It takes the states of manipulator as inputs and produces the control torque as output.

Genetic algorithm firstly creates a random population of 20 members (chromosome), each composed from two traits. The chromosome is a possible optimum solution. Then Genetic algorithm separates the genes of each trait. In order to select the best population members, the fitness of each population member is evaluated. The population members of the maximum fitness are the best population members. The best population members are selected as parents for the next generation. Ellisim is the process of reproducing the best parents to the next generations without changes. If the Ellisim is enabled, the best members are passed to the next generation without any changes other wise all population members of the next generation are created from the Cross Over and Mutation processes of parents.

In this control strategy, we have selected the mutation and cross over probabilities to be 0.05 and 0.8 respectively. Also we have chosen a population size of 20 members, two traits per chromosome. In each sample time of simulation, we generate 50 generations in order to select the best value of K_d & K_p .

The fitness function used to tune the PD gains using genetic algorithm is selected to be:

$$fitness(chromosome, generation) = \frac{1}{SD} \tag{5.22}$$

Where SD is the sum of differences between the desired and ten-estimated future states of manipulator and it is described by equation 5.23 as follows:

$$SD = \sum_{n=1}^{n=10} ((\dot{\theta}_1^d - \dot{\theta}_1^n) + (\theta_1^d - \theta_1^n) + (\dot{\theta}_2^d - \dot{\theta}_2^n) + (\theta_2^d - \theta_2^n)) \tag{5.23}$$

The simulation results, as described in chapter seven, show that the Genetic Algorithm successfully tuned the PD gains of the swing up controller.

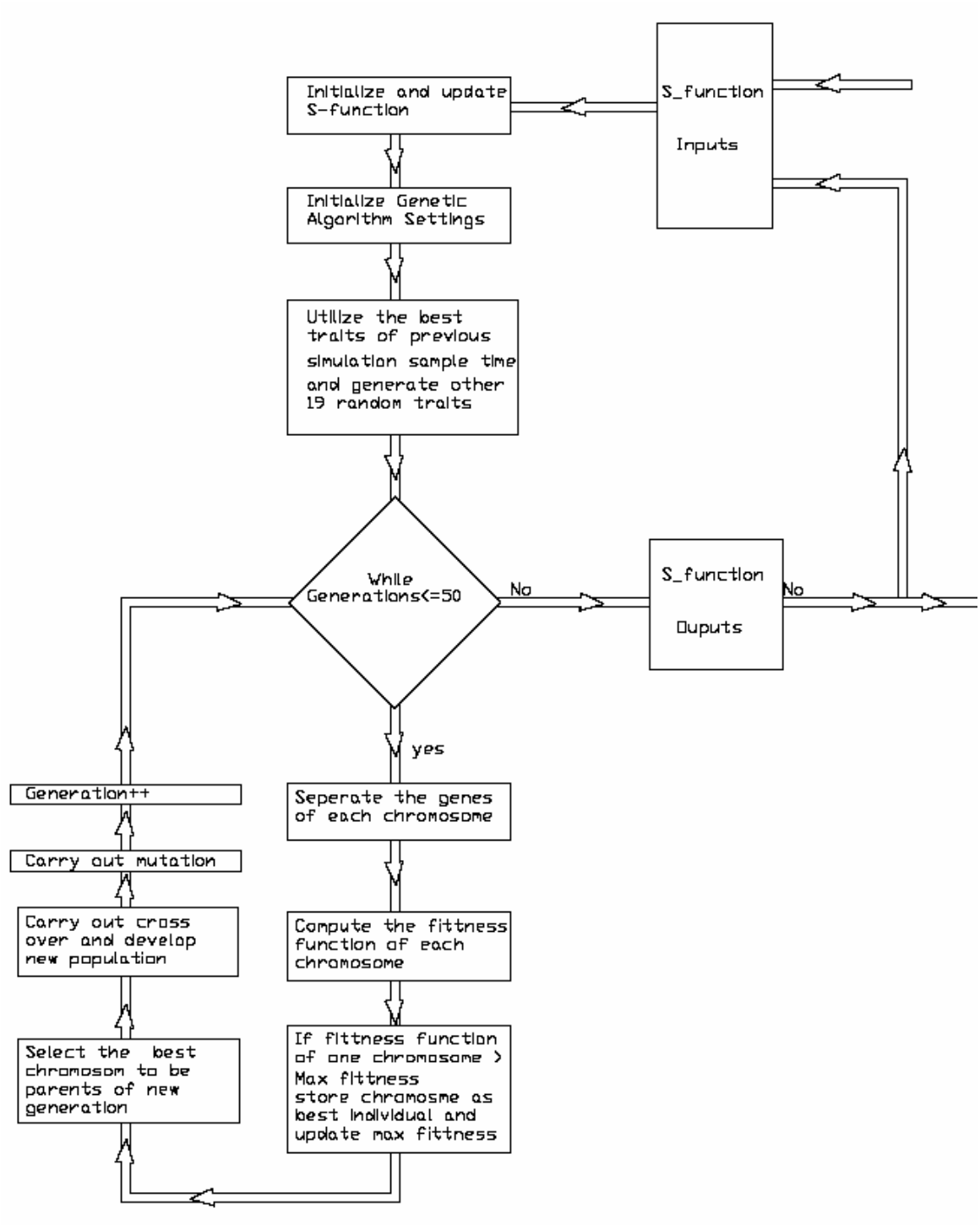


Figure 5.7 Flow chart of genetic algorithm

CHAPTER 6: BALANCING CONTROLLERS

The attractive orbit at which the controller should be switched to the balancing controller where the second link of the manipulator moves toward the equilibrium point $(\theta_1 = \pi/2\ ,\ \theta_2 = 0)$ is defined by [12] as

$$\frac{1}{2}\alpha_2\dot{\theta}_2^2 = \alpha_5 g(1-\cos(\theta_2)) \tag{6.1}$$

The control algorithm is switched from the swinging controller to the balancing controller when $\left|\theta_1 - \pi/2\right| \langle\ 0.2 \quad rad$ and $\left|\theta_2\right| \langle\ 0.3 \quad rad$

The following balancing controllers will be developed and the results of each will be compared.

6.1. Linear Quadratic Regulator (LQR)

The linearized model of the manipulator at a desired unstable position $\left[\theta_1 = \pi/2 \quad \theta_2 = 0\right]$ is obtained to develop a linear balancing controller for the two-link underactuated manipulator. Evaluating the parameters $(\alpha_1\ ,\alpha_2\ ,\alpha_3\ ,\alpha_1\ ,\alpha_4\ ,\alpha_5)$ of equation 3.28 and 3.29

$\alpha_1 = m_1 l_{lc1}^2 + m_2 l_1^2 + I_1$, $\alpha_2 = m_2 l_{lc2}^2 + I_2$, $\alpha_3 = m_2 l_1 l_{lc2}$, $\alpha_4 = m_2 l_{c1} + m_2 l_1$, $\alpha_5 = m_2 l_{c2}$

and substituting them into (4.3), the linearized model of the manipulator is obtained as follows:

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 108.84 & 0 & -25.03 & 0 \\ 0 & 0 & 0 & 1 \\ -101.54 & 0 & 103.25 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 2.0069 \\ 0 \\ -3.066 \end{bmatrix} \tau_1 \tag{6.2}$$

The linear quadratic regulator (LQR) is an optimal state- feedback controller that minimizes the quadratic cost criterion J [13]:

$$J = \int_0^{\infty} (x^T Q\, x + u^T R\, u) dt$$

The weighting matrices Q and R are designed and modified to achieve an acceptable tradeoff between performance and control effort.

In practice, an appropriate choice to obtain acceptable values x and u is to initially choose diagonal matrices Q and R such that [13]:

. $Q_{ii} = 1/\max\ imum \quad accepted \quad value \quad [x_{ii}]$

$R_{ii} = 1/\max imum \quad accepted \quad value \quad [u^2{}_{ii}]$

By several simulation trials we found that best Q & R that achieves the balancing of our manipulator to be as follows:

$$Q = \rho H_1^T H_1$$

Where $\rho = 1$, $H_1 = \begin{bmatrix}(1/\pi) & 0 & 1 & 0\end{bmatrix}$

$$Q = \begin{bmatrix} 0.4053 & 0 & 0.6366 & 0 \\ 0 & 0 & 0 & 0 \\ 0.6366 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R = \begin{bmatrix}1\end{bmatrix}$$

By solving Riccati equation, we obtain the gain vector:

$$K = \begin{bmatrix}-3.4085 & -0.5190 & -3.1058 & -0.3632\end{bmatrix}$$

The above gain vector that is used in state feedback control to balances the manipulator at the right position. Detailed discussion of results is presented in chapter seven.

6.2. Sliding Mode Balancing Controller

A sliding-mode controller is a variable-structure controller (VSC) whose structure between the switching surfaces is changed to achieve robust control characteristics.

Since the system under study is underactuated, i.e. we have one control input for two degrees of freedom manipulator, we used hierarchical sliding mode controller [36]

The equation of motion of two link manipulator (3.30) can be expressed as:

$$\begin{bmatrix}\ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix} = D^{-1}(\tau - C\begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}2\end{bmatrix} + G) = D^{-1}\tau - D^{-1}(C\begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}2\end{bmatrix} + G) \tag{6.3}$$

Let $\tilde{x}_1 = \theta_1^d - \theta_1$, $\tilde{x}_2 = \dot{\theta}_1^d - \dot{\theta}_1$, $\tilde{x}_3 = \theta_2^d - \theta_2$, $\tilde{x}_4 = \dot{\theta}_2^d - \dot{\theta}_2$

Equation 6.3 can be written as follows:

$$\begin{aligned}\dot{\tilde{x}}_1 &= \tilde{x}_2\\ \dot{\tilde{x}}_2 &= f_1(\tilde{x}) + b_1 \cdot \tau_1\\ \dot{\tilde{x}}_3 &= \tilde{x}_4\\ \dot{\tilde{x}}_4 &= f_2(\tilde{x}) + b_2 \cdot \tau_2\end{aligned} \tag{6.4}$$

Where

$\tau_2 = 0$ (*underactuated*) , and $\tilde{x} = [\tilde{x}_1 \quad \tilde{x}_2 \quad \tilde{x}_3 \quad \tilde{x}_4]^T$

$$f_1(x) = \frac{1}{\alpha_1\alpha_2 - \alpha_3^2\cos^2(\theta_2)}\begin{bmatrix}\alpha_2\alpha_3\sin(\theta_2)(\dot{\theta}_1 + \dot{\theta}_2)^2 + \alpha_3^2\cos(\theta_2)\sin(\theta_2)\dot{\theta}_1^2 -\\ \alpha_2\alpha_4 g\cos(\theta_1) + \alpha_3\alpha_5 g\cos(\theta_2)\cos(\theta_1 + \theta_2)\end{bmatrix} \tag{6.5}$$

$$b_1 = \alpha_2(\alpha_2\,\alpha_1 - (\alpha_3\cos(\theta_2))^2) \tag{6.6}$$

$$f_2(x) = \frac{1}{\alpha_1\alpha_2 - \alpha_3^2\cos^2(\theta_2)} \begin{bmatrix} -\alpha_3(\alpha_2 + \alpha_3\cos(\theta_2)\sin(\theta_2)(\dot{\theta}_1 + \dot{\theta}_2)^2 - \\ (\alpha_1 + \alpha_3\cos(\theta_2))\sin(\theta_2)\dot{\theta}_1^{\,2} + \\ (\alpha_2 + \alpha_3\cos(\theta_2))(\alpha 4g\cos(\theta_1)) - \\ (\alpha_1 + \alpha_3\cos(\theta_2))\alpha_5 g\cos(\theta_1 + \theta_2) \end{bmatrix} \tag{6.7}$$

$$b_2 = -\left(\frac{\alpha_2 + \alpha_3\cos(\theta_2)}{\alpha_2\,\alpha_1 - (\alpha_3\cos(\theta_2)^2}\right) \tag{6.8}$$

Let's define the sliding surfaces of the first joint and second joint as follows:

$$S_1 = \lambda_1\tilde{x}_1 + \tilde{x}2 \qquad \lambda_1 > 0 \tag{6.9}$$

$$S_2 = \lambda_2\tilde{x}_3 + \tilde{x}_4 \qquad \lambda_2 > 0 \tag{6.10}$$

Since our system is underactuated, we need to define a second level sliding surface that simultaneously stabilizes the above surfaces. Let's define the second level sliding surface as follows [36]:

$$S = \beta \cdot S_1 + \delta \cdot S_2 \tag{6.11}$$

Also let's define the input control torque applied on joint one as:

$$\tau_1 = \tau_{equiv1} + \tau_{equiv2} + \tau_{sw} \tag{6.12}$$

Where

$$\tau_{equiv1} = -\frac{(f_1(x) + \lambda_1 \cdot \tilde{x}_2)}{b_1} \tag{6.13}$$

$$b_1 = (\alpha_2(\alpha_2\,\alpha_1 - (\alpha_3\cos(\theta_2))^2) \neq 0 \text{ , since } \frac{\sqrt{\alpha_2\,\alpha_1}}{\alpha_3} = 2.4228 > 1$$

$$\tau_{equiv2} = -\frac{(f_2(x) + \lambda_2 \cdot \tilde{x}_4)}{b_2} \qquad b_2 \neq 0 \tag{6.14}$$

Choosing Lyapunov function as

$$V_l = \frac{1}{2} S^2 \tag{6.15}$$

Thus from Lyapunov stability theorem $\dot{V}_l$ should be positive definite

$$\dot{V}_l \leq 0 \tag{6.16}$$

$$\dot{V}_l = S\,\dot{S} = S(\beta\,\dot{s}_1 + \delta\,\dot{s}_2)$$

$$\begin{aligned}
\dot{V}_l &= S(\beta(\lambda_1 \dot{\tilde{x}}_1 + \dot{\tilde{x}}_2) + \delta(\lambda_2\,\dot{\tilde{x}}_1 + \dot{\tilde{x}}_2)) \\
&= S[\beta(\lambda_1 \tilde{x}_2 + f_1) + b_1(\tau_{equiv1} + \tau_{equiv2} + \tau_{sw}) + \delta(\lambda_2 \tilde{x}_4 + f_2) + b_2(\tau_{equiv1} + \tau_{equiv2} + \tau_{sw})] \\
&= S[\beta\, b_1(\tau_{equiv2} + \tau_{sw}) + \delta\, b_2(\tau_{equiv1} + \tau_{sw}\ )] \\
&= S[\delta \cdot b_2 \cdot \tau_{equiv1} + \beta \cdot b_1 \cdot \tau_{equiv2} + \tau_{sw}(\delta \cdot b_2 + \beta \cdot b1)]
\end{aligned} \tag{6.17}$$

To satisfy the equation 6.16, let's define τ_{sw} as follows

$$\tau_{sw} = -\frac{(\delta \cdot b_2 \cdot \tau_{equiv1} + \beta \cdot b_1 \cdot \tau_{equiv2} - \mu \cdot \mathrm{sgn}(S) - k \cdot S)}{(\delta \cdot b_2 + \beta \cdot b_1)} \tag{6.18}$$

Where $\mu > 0$, $k > 0$

From equations 6.12, 6.13, 6.14, and equation 6.18.The total input torque τ_1 equals:

$$\tau = \frac{\beta \cdot b_1}{\beta \cdot b_1 + \delta \cdot b_2} \tau_{equiv1} + \frac{\delta \cdot b_2}{\beta \cdot b_1 + \delta \cdot b_2} \tau_{equiv2} - \hat{\mu} \cdot \mathrm{sgn}(S) - \hat{k} \cdot S \tag{6.19}$$

Where

$$\hat{\mu} = \frac{\mu}{\beta \cdot b_1 + \delta \cdot b_2} \quad , \hat{k} = \frac{k}{\beta \cdot b_1 + \delta \cdot b_2}$$

Substituting equation 6.18 into equation 6.17 we obtain:

$$\dot{V}_l = -k \cdot S^2 - \mu|s| < 0 \qquad \mu > 0\,,\, k > 0 \tag{6.20}$$

Therefore the Second Level sliding surface is stable.

To select the parameters $\delta, \beta, \lambda_1, \lambda_2$, we varied the parameters $\delta, \beta, \lambda_1, \lambda_2$ within specific ranges and evaluated the summation of sliding surfaces S for a period of time z. β is selected relative to δ and λ_1 is selected relative to λ_2. The set of parameters $\delta, \beta, \lambda_1, \lambda_2$ that evolves the minimum summation of sliding surface are the selected ones. The parameters μ, k must be greater than zero. The Detailed discussion of results and selection of parameters are presented in chapter seven.

6.3. Genetic Sliding Mode Balancing Controller

In the previous section, we have developed the sliding mode balancing controller. However the reaching time of the sliding surface is not optimum and the switching function produces chattering around the sliding surface. Therefore, we have developed genetic algorithm and combined it with sliding mode controller to optimize the reaching time of sliding surface and reduce the chattering.

Genetic Algorithm is used to search for the best values of k *and* μ in equation 6.18 that minimize chattering and reduce the hitting time of sliding surface. Genetic algorithm is built inside an S-function block in Matlab Simulink as described in details in Chapter 6. The algorithm uses the states of Manipulator as input (four inputs), in addition to manipulator parameters, to compute the control torque of equation 6.18.

The Genetic Algorithm divides the genes of each population members (chromosome) into two traits, the first trait is devoted for the optimal value of k, and the second trait is devoted for the optimal value of μ. It generates 20 generations for 20 population members during each simulation sample time. During generation of population we have selected the *cross over* probability and *mutation* probability to be 0.5 and 0.05 respectively.

To evaluate the fitness function, we estimate the control Torque based on the random values of traits according to equation 6.18. Then we estimate the predicted future states of manipulator using the forward Euler method as follows:

$$x_{2-est} = -(f_1(\tilde{x}) + b_1 * \tau_{est}) \cdot \Delta T + x_2 \tag{6.21}$$

$$x_{4-est} = -(f_2(\tilde{x}) + b_2 * \tau_{est}) \cdot \Delta T + x_4 \tag{6.22}$$

$$x_{1-est} = x_{2-est} \cdot \Delta T + x1 \tag{6.23}$$

$$x_{3-est} = x_{4-est} \cdot \Delta T + x_3 \tag{6.24}$$

Where ΔT : is the sampling time.

By estimating the states, the sliding surfaces are also estimated as follows:

$$s_{1-est} = \lambda_1 \tilde{x}_{1\_est} + \tilde{x}_{2-est} \qquad \lambda_1 > 0 \tag{6.25}$$

$$s_{2-est} = \lambda_2 \tilde{x}_{3\_est} + \tilde{x}_{4\_est} \qquad \lambda_2 > 0 \tag{6.26}$$

$$S_{\_est} = \beta \cdot s_{1\_est} + \delta \cdot s_{2\_est} \tag{6.27}$$

Thus, the e fitness function for each population member is selected to be as follows:

$$fittness(popultion\ member, number\ of\ generation) = 1000 * \exp(-abs(S_{\_est})) \tag{6.28}$$

Genetic sliding mode controller has improved the performance of sliding mode controller with a cost. The detailed discussion of results will be presented in Chapter seven.

6.4. Neuro Fuzzy Balancing Controller

Similar to Neuro Fuzzy Swinging up Controller, we have collected the states of the manipulator (four states) and the output torque of state feedback controller using LQR gains, and combined them in one training matrix. Then we have used ANFIS algorithm to build up the Neuro Fuzzy Model. The structure of ANFIS algorithm and structure of fuzzy controller are shown inFigure 6.1 and Figure 6.2.The membership functions of the inputs are shown in Figure 6.3.

Afterwards fuzzy logic controller using Simulink is to balance the manipulator independently. The detailed discussion of results is presented in chapter seven.

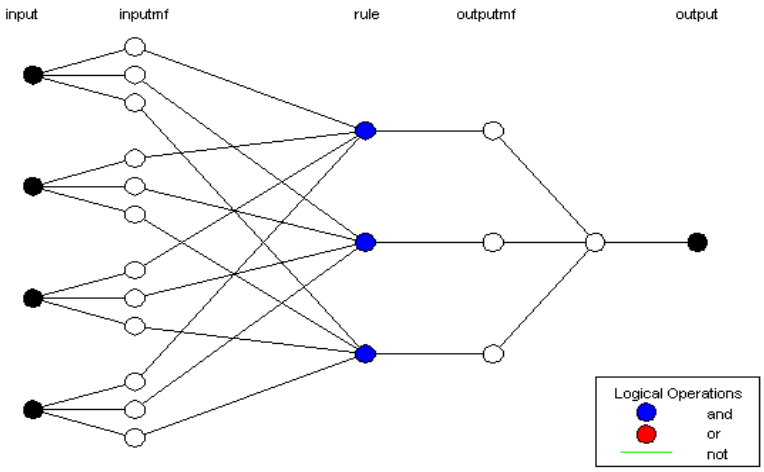


Figure 6.1 Structure of ANFIS algorithm for balancing controller

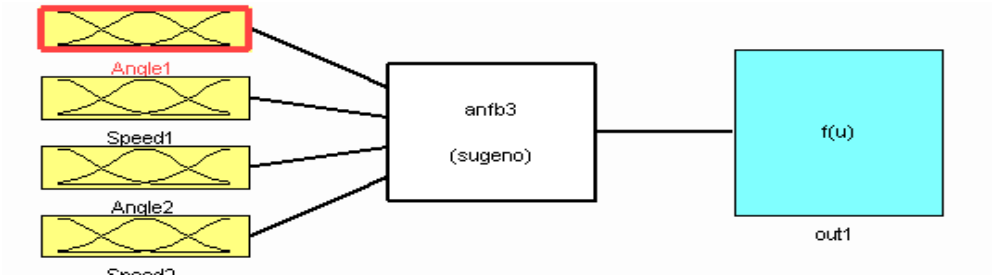


Figure 6.2 Structure of Fuzzy controller for balancing the manipulator at the upright position

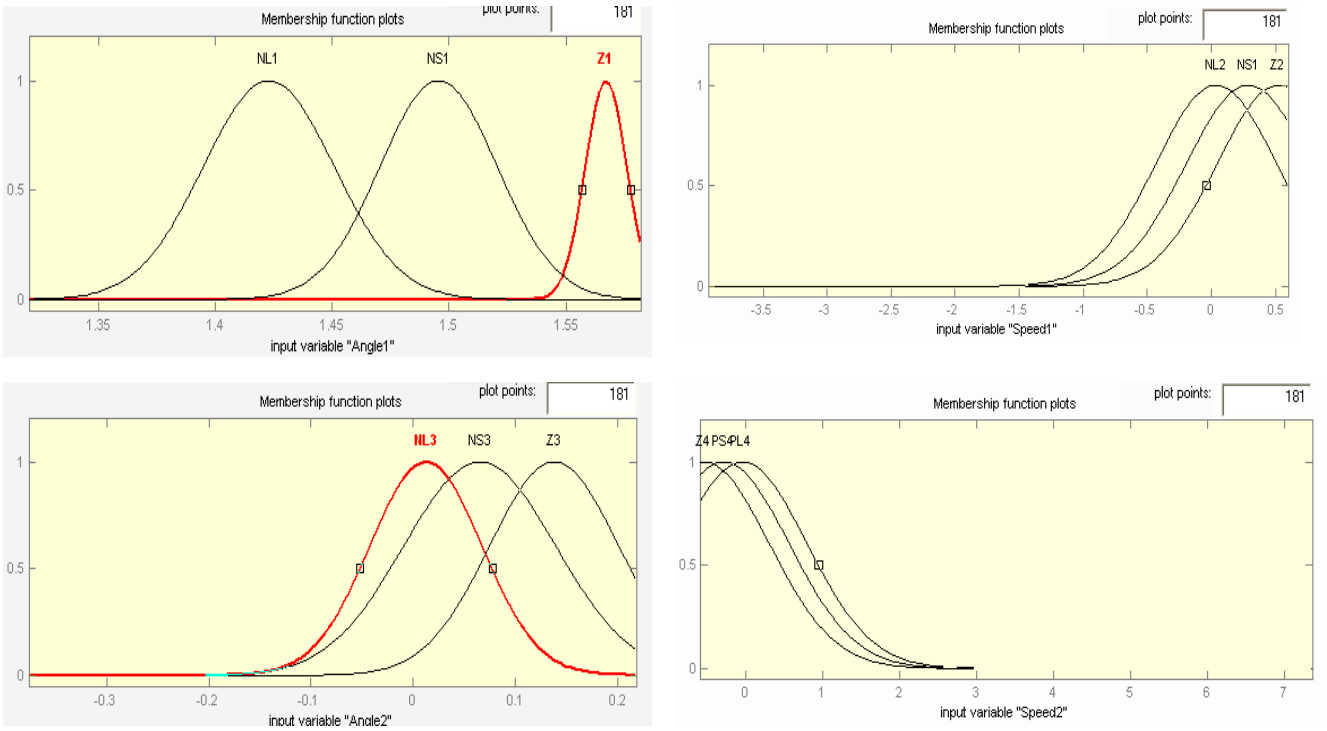


Figure 6.3 Membership functions of fuzzy controller inputs (balancing controller)

CHAPTER 7: EXPERIMENTAL AND SIMULATION RESULTS

7.1. Simulation Results for Feedback Linearization and LQR Techniques

As mentioned in previous chapters, stabilizing underactuated mechanical systems has been achieved in two stages, the swinging up stage and balancing stage. In this section, we summarize the results of the classical swinging up and balancing techniques. Feedback linearization technique is used for swinging up stage and LQR algorithm is used for balancing stage.

The swinging up and balancing torques is shown in Figure 7.1 and Figure 7.2 respectively. The combination of swinging up torque and balancing torque using the adequate switch along with initial pumping torque (-0.35 N.m) for 500 millisecond is shown in Figure 7.3 The joint angle of the first link second link are shown in Figure 7.4 and Figure 7.5 respectively .Also the angular velocity of the first link second link are of are shown in Figure 7.6 and Figure 7.7 respectively.

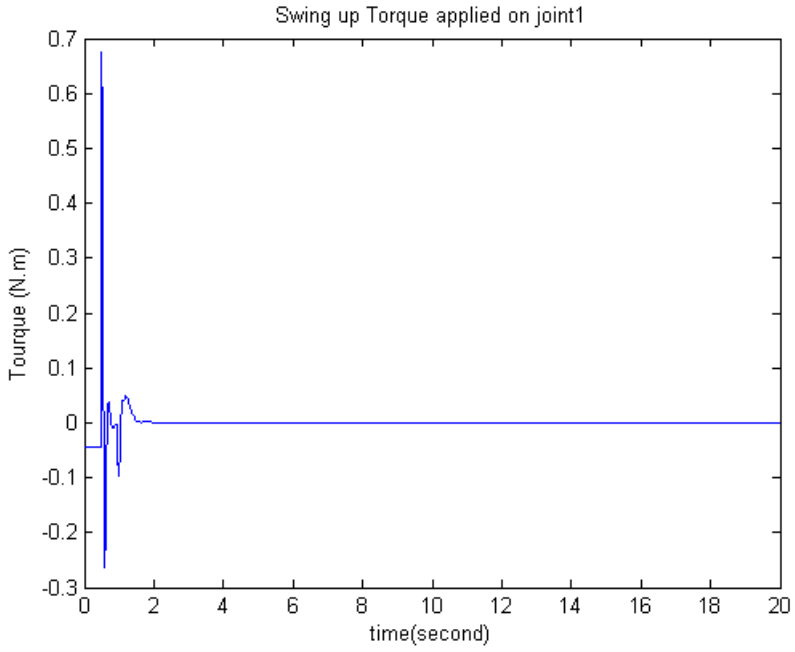


Figure 7.1 Swinging up torque applied to joint one

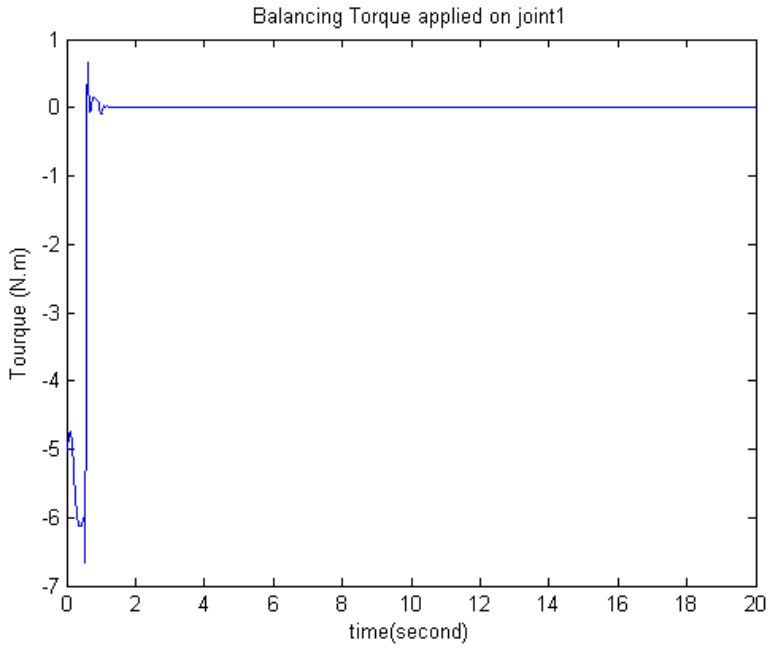


Figure 7.2 Balancing torque applied to Joint one

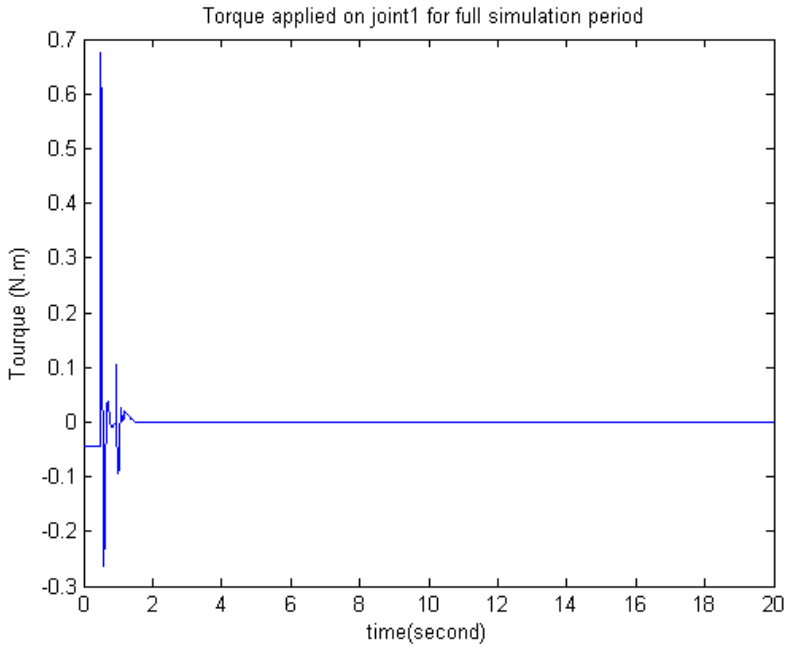


Figure 7.3 Swinging and balancing torque on joint one

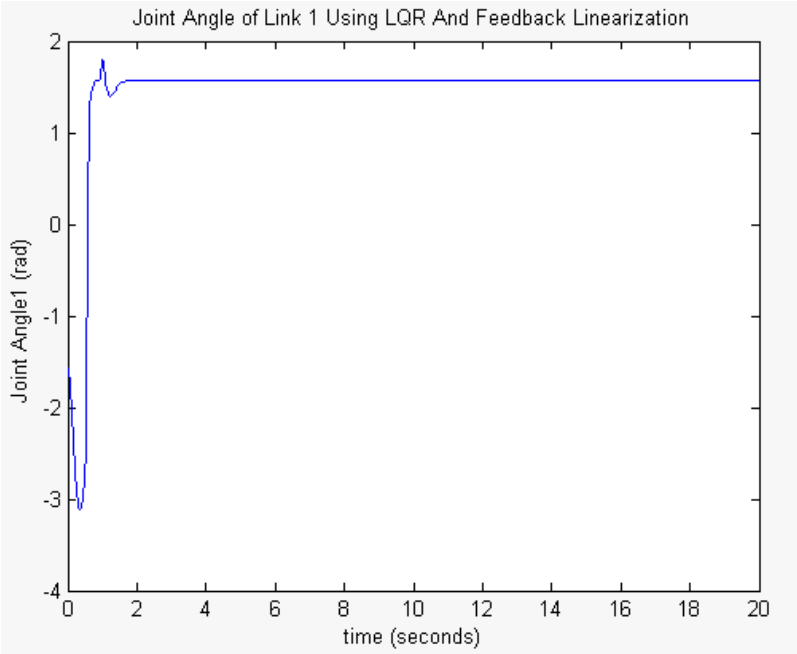


Figure 7.4 Joint angle of link one using LQR and feedback linearization

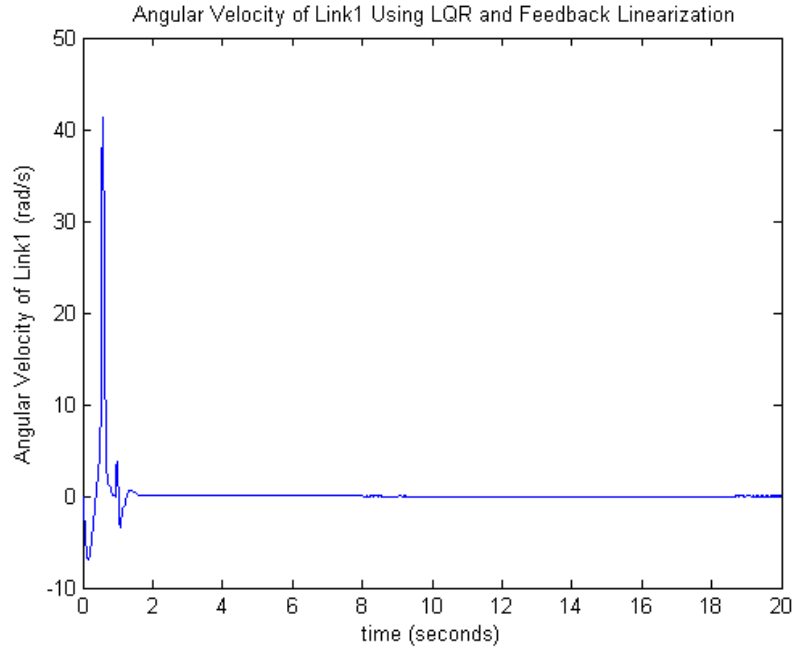


Figure 7.5 Joint angle of link two using LQR and feedback linearization

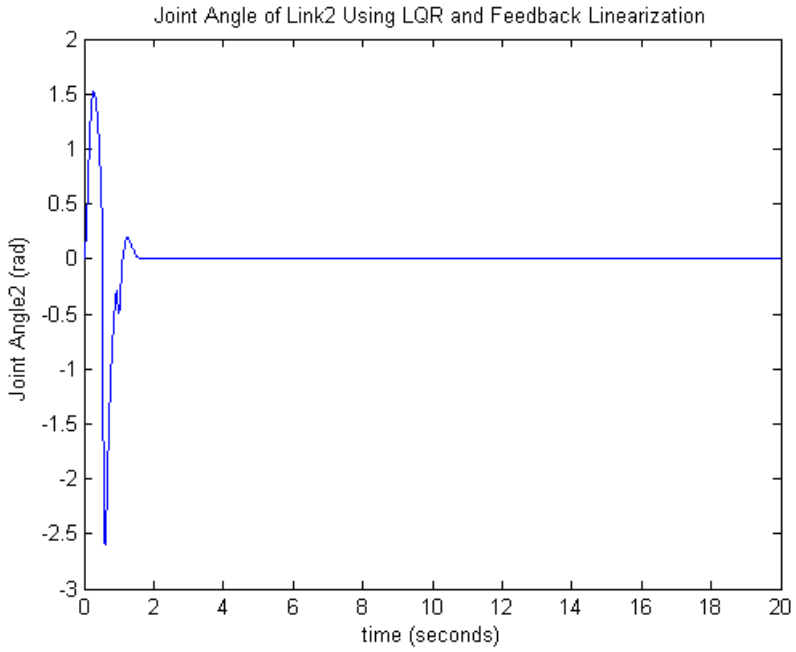


Figure 7.6 Angular velocity of link one using LQR and feedback linearization

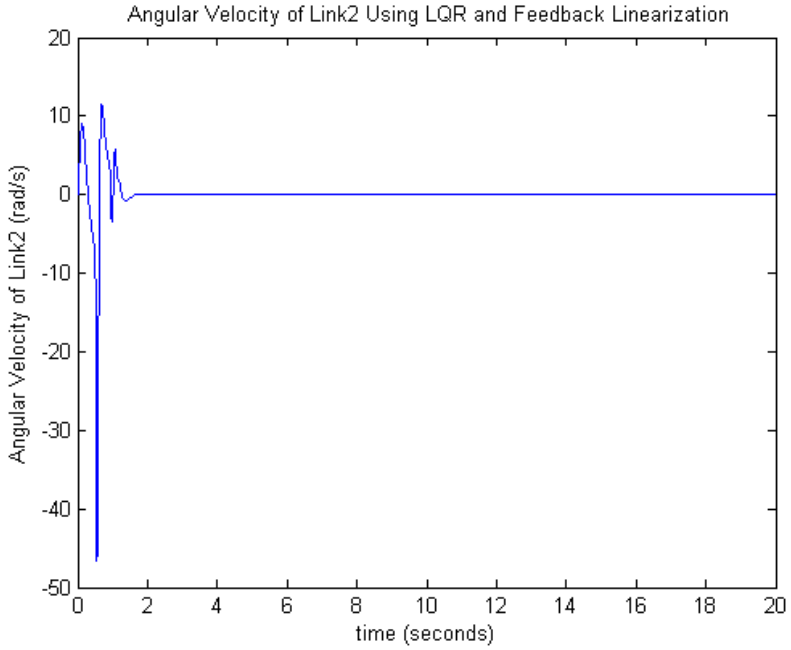


Figure 7.7 Angular velocity of link two using LQR and feedback linearization

Clearly one observes that feedback linearization technique and LQR techniques are able to balance the manipulator at the upright unbalanced position. However both techniques assume precise knowledge of manipulator parameters and have neglected the friction. These assumptions may result in unstable system in real time. In addition, both techniques can not accommodate any external disturbance.

7.2. Simulation Results Using Sliding Mode Swing up Controller

In this section we demonstrate the simulation results using sliding mode controller to swing up the underactuated manipulator at the upright position. Also we compare the results of sliding mode swinging up controller with the feedback linearization technique. For comparison study, the LQR method is still adopted for balancing the manipulator.

The swinging up torque of sliding mode controller is shown in Figure 7.8. The combination of swinging up control actions produced by both the feedback linearization technique and the sliding mode controller are shown in Figure 7.9. From Figure 7.8 and Figure 7.9, one can observe that the pumping torque remains unchanged, and the swinging up torque produced by sliding mode control methodology slightly differs from feedback linearization technique. However, as shown in Figure 7.10, Figure 7.11, Figure 7.12 and Figure 7.13, the sliding mode control methodology is successful in swinging the manipulator to the upright position. The joint angle of link one and joint angle of link two are shown in Figure 7.10 and Figure 7.11 respectively. The angular velocity of link one and angular velocity of link two are shown in Figure 7.12 and Figure 7.13.

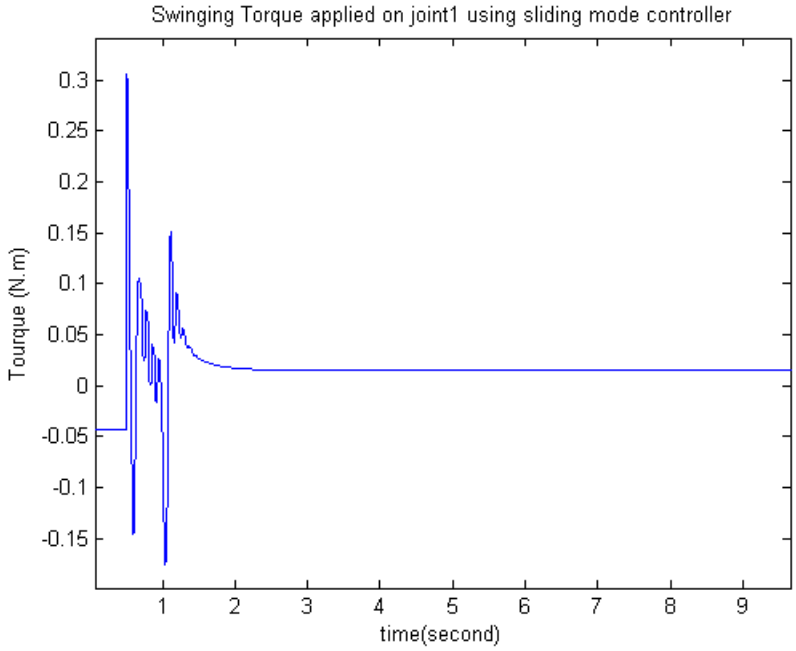


Figure 7.8 Swinging torque applied on joint one using sliding mode controller

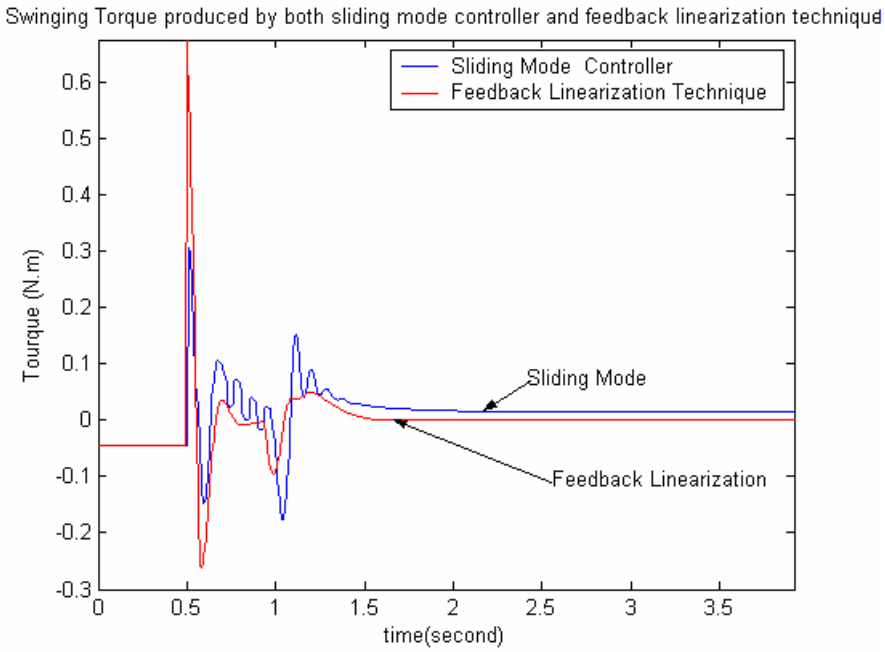


Figure 7.9 Comparison of swing torques produced by sliding mode and feedback linearization controllers

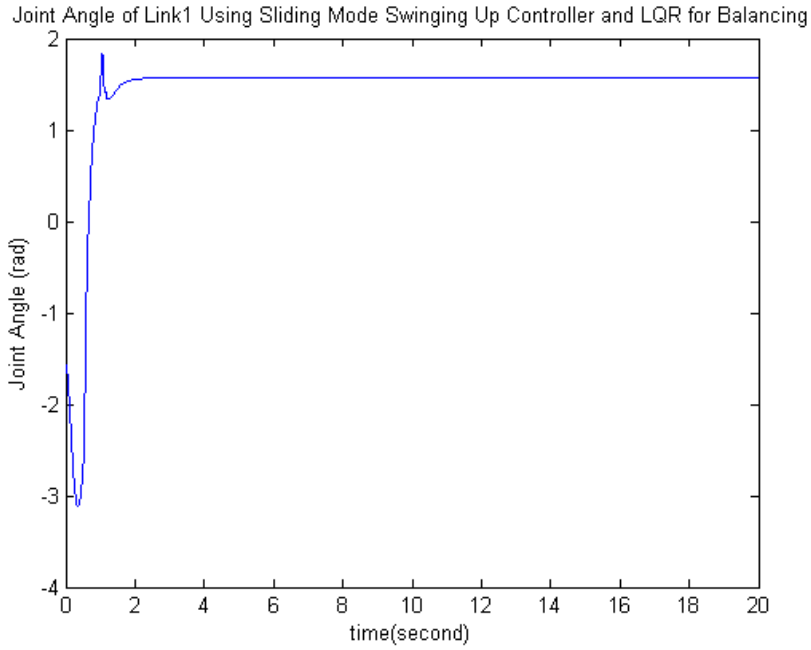


Figure 7.10 Joint angle of link one using sliding mode swinging up controller

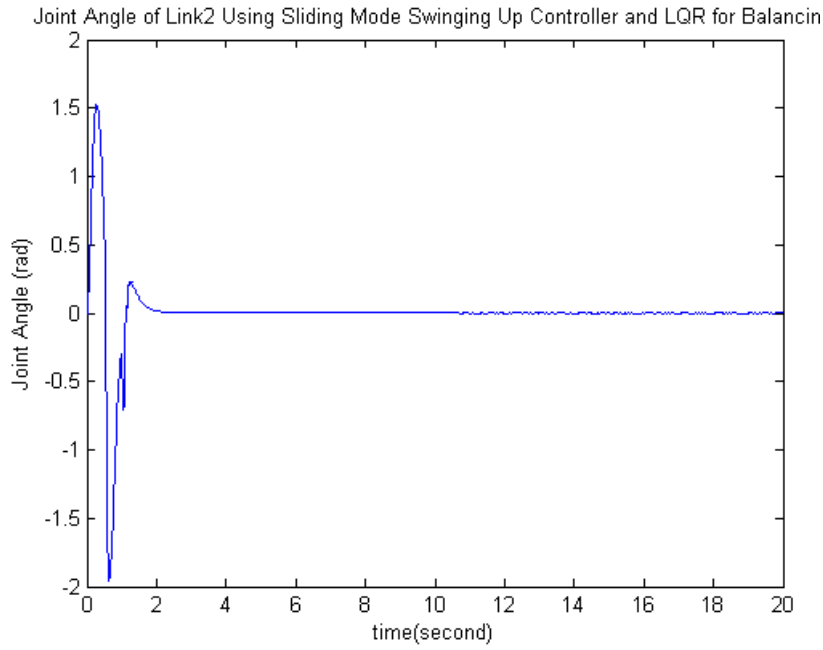


Figure 7.11 Joint angle of link two using sliding mode swinging up controller

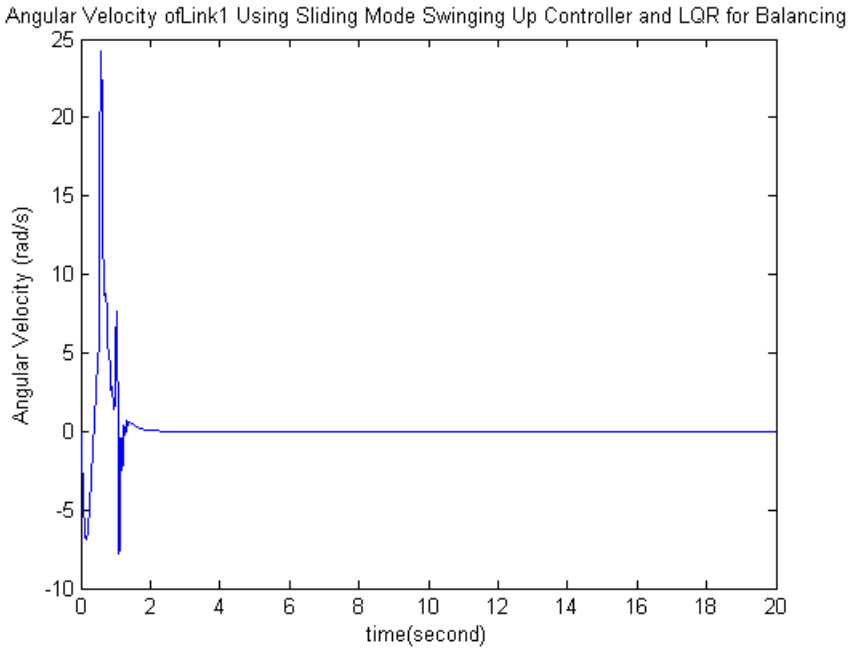


Figure 7.12 Angular velocity of link one using sliding mode swinging up controller

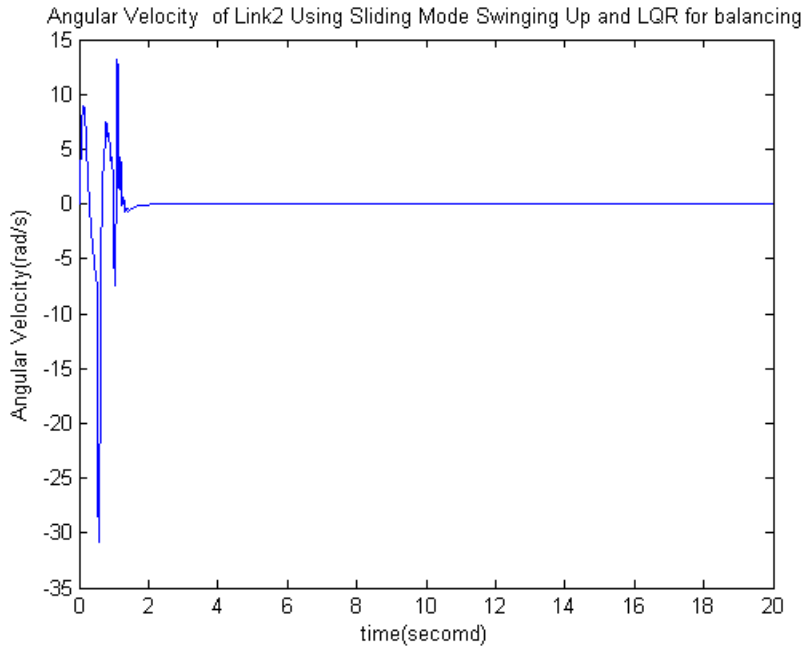


Figure 7.13 Angular velocity of link two using sliding mode swinging up controller

7.3. Simulation Results Using Neuro Fuzzy Swing up Controller

In this section we demonstrate the simulation results using Neuro Fuzzy controller to swing up the manipulator at the upright position .As discussed in chapter five, the Neuro Fuzzy controller was trained by the results of feedback linearization using ANFIS. For comparison the LQR method is still adopted for balancing the manipulator.

The swinging up torque of the Neuro Fuzzy controller is shown in Figure 7.14. The combination of swinging up torques produced by both feedback linearization technique and Neuro Fuzzy controller is shown Figure 7.15. After 0.93 seconds the controller is switched to the balancing controller. As shown in Figure 7.14 and Figure 7.15, ANFIS is successful in training Neuro Fuzzy controller. The joint angle of link one and joint angle of link two are shown in Figure 7.16 and Figure. The angular velocity of link one and angular velocity of link two are shown in Figure 7.18 and Figure 7.19

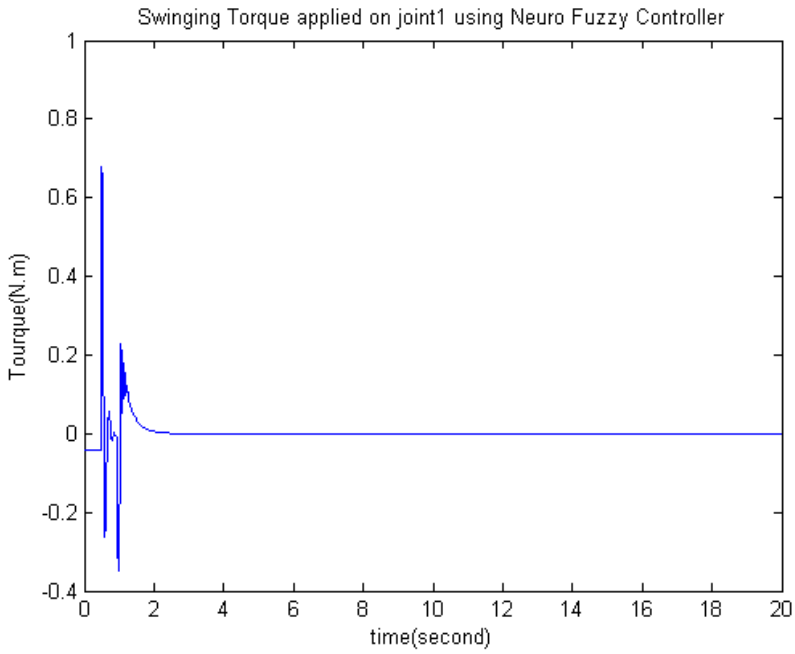


Figure 7.14 Swinging torque applied on joint one using Neuro Fuzzy controller

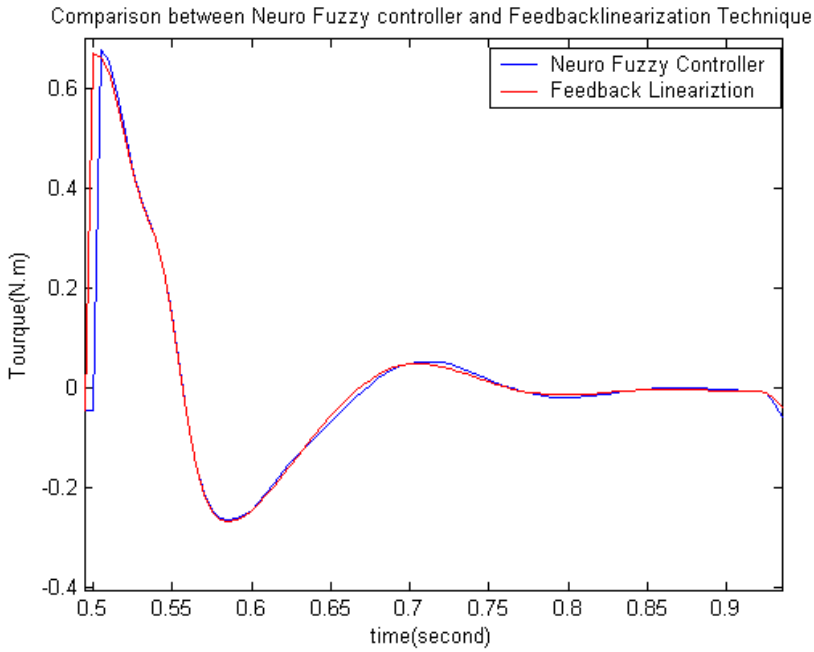


Figure 7.15 Comparison of swing torques produced by Neuro Fuzzy controller and the feedback linearization controllers

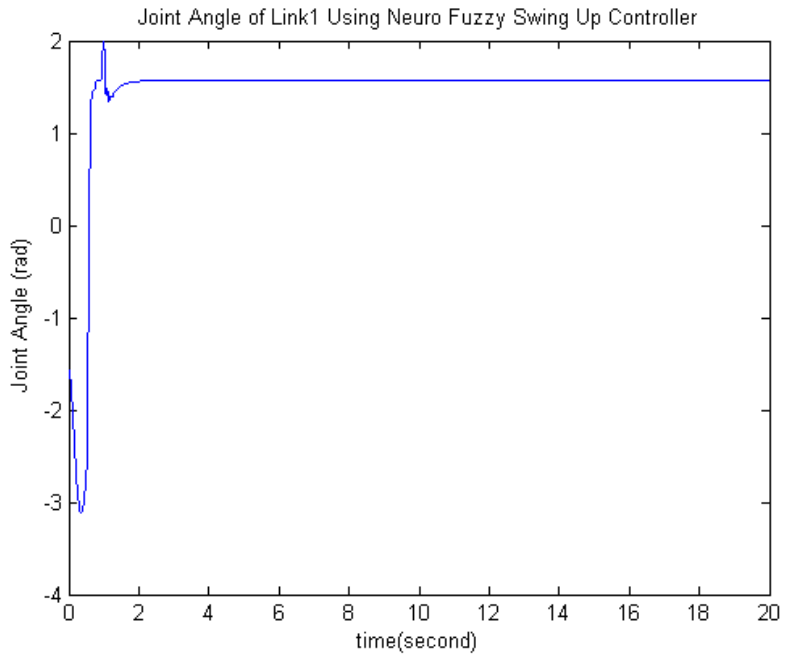


Figure 7.16 Joint angle of link one using Neuro Fuzzy swinging up controller

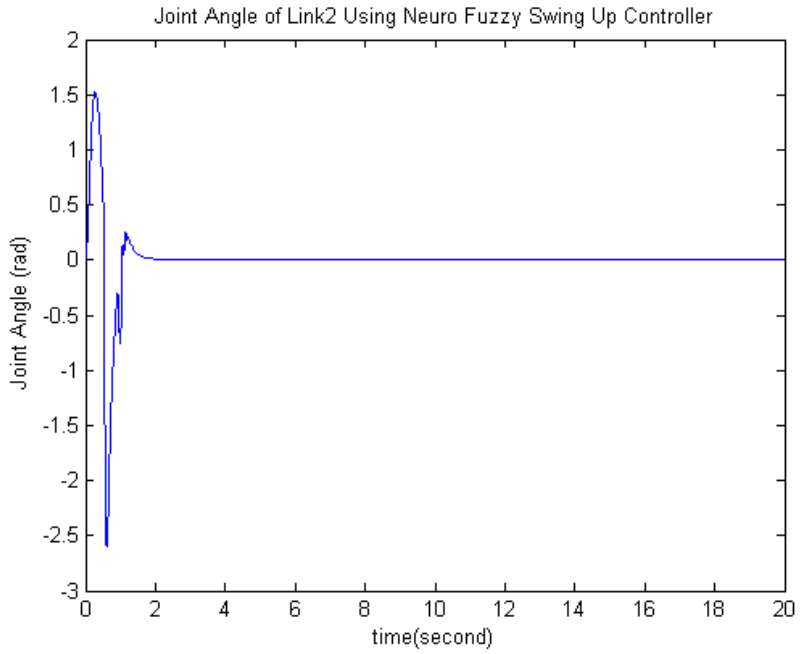


Figure 7.17 Joint angle of link two using Neuro Fuzzy swinging up controller

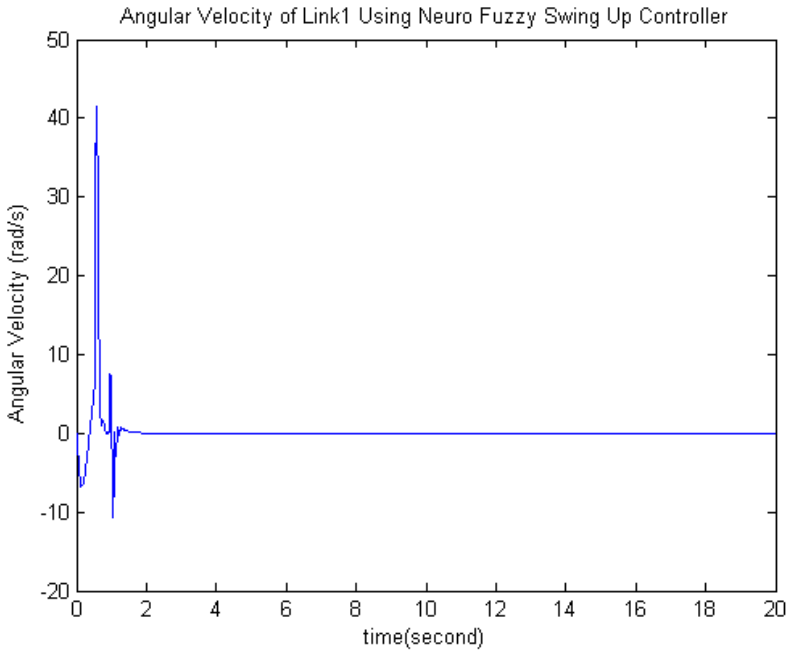


Figure 7.18 Angular velocity of link one using Neuro Fuzzy swinging up controller

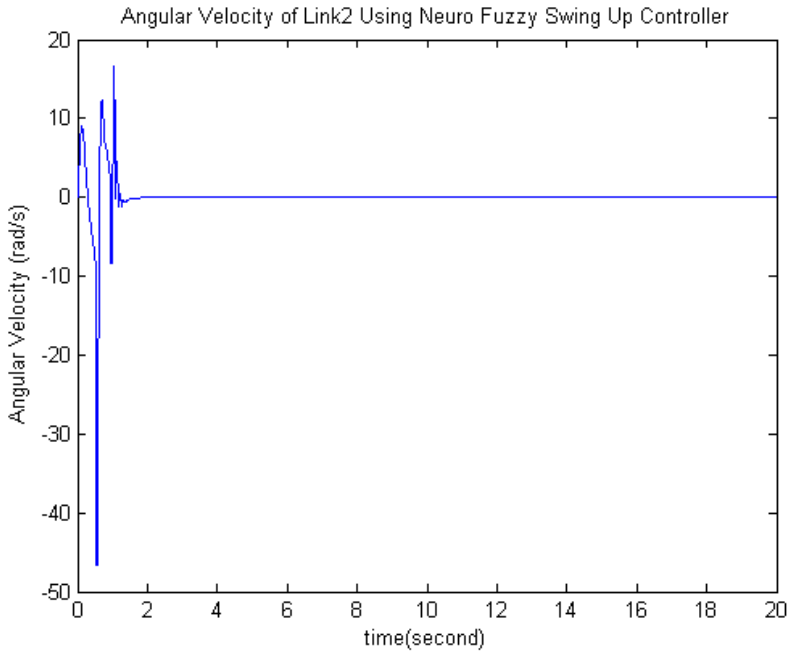


Figure 7.19 Angular velocity of link two using Neuro Fuzzy swinging up controller

7.4. Simulation Results for Tuning PD Gains Using Genetic Algorithm

In this section we present the simulation results of tuning the PD gains of feedback linearization technique using genetic algorithm. As shown in Figure 7.20

and Figure 7.21, the technique of tuning the PD gains of equation 5.9 using genetic algorithm is successful and efficient to swings up the manipulator at the upright position. Figure 7.20 shows the Joint angle of link one, and Figure 7.21 shows the joint angle of link two. The control torque of feedback linearization technique using tuned PD gains is shown in Figure 7.22. Figure 7.23 shows that the magnitude of output torque using tuned gains is higher than the output torque without tuning the gains. The selected PD gains K_d & K_p using Genetic algorithm are shown in Figure 7.24 and Figure 7.25.

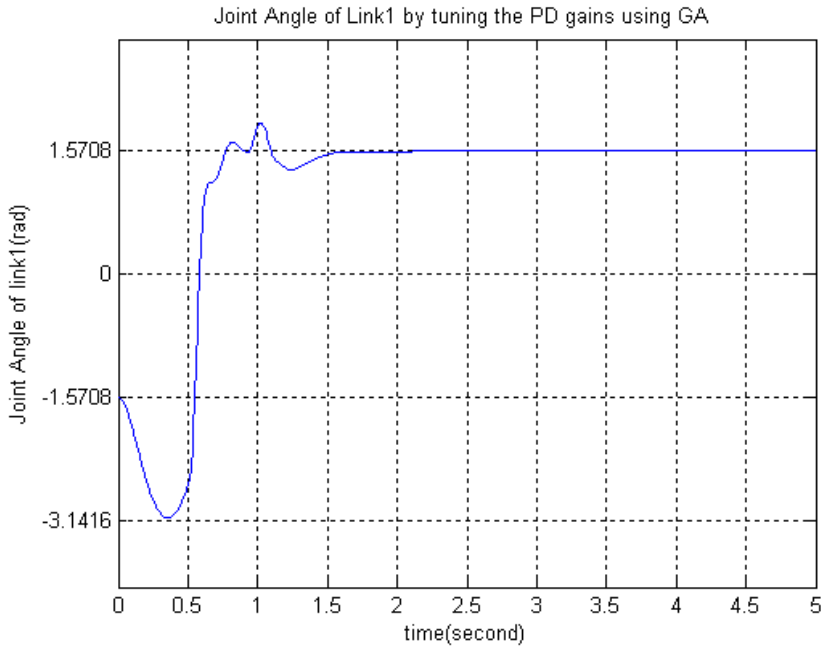


Figure 7.20 Joint angle of link one using tuned PD gains by GA

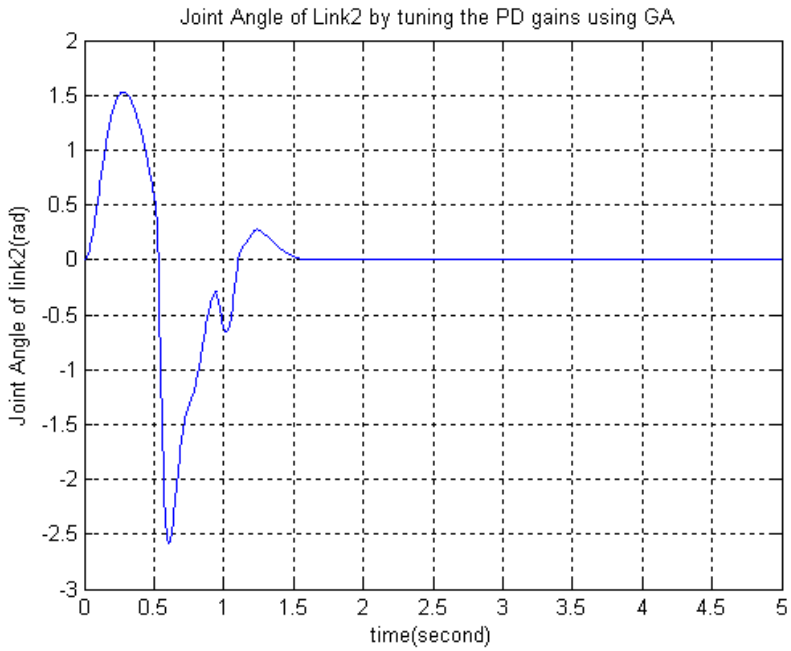


Figure 7.21 Joint angles of link two using tuned PD gains by GA

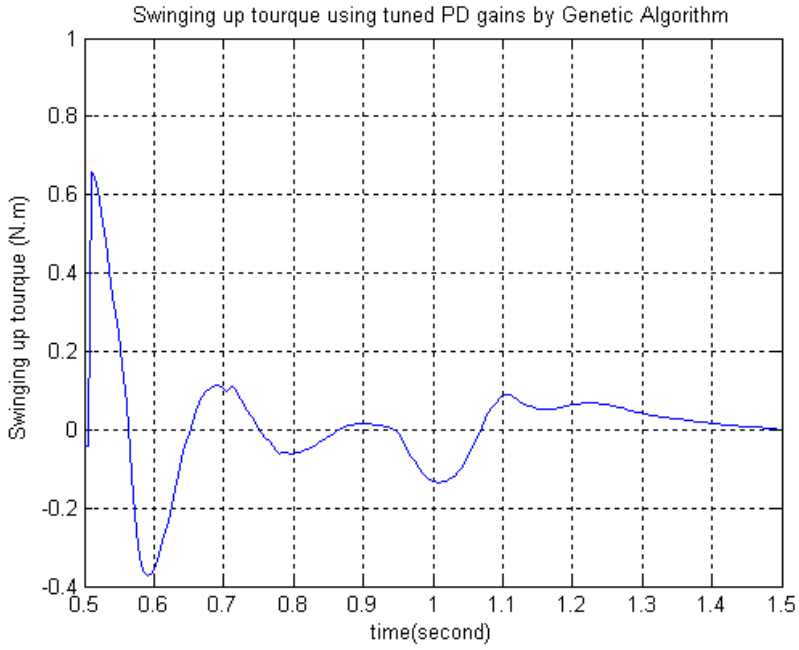


Figure 7.22 Swinging up torque using tuned PD gains by GA

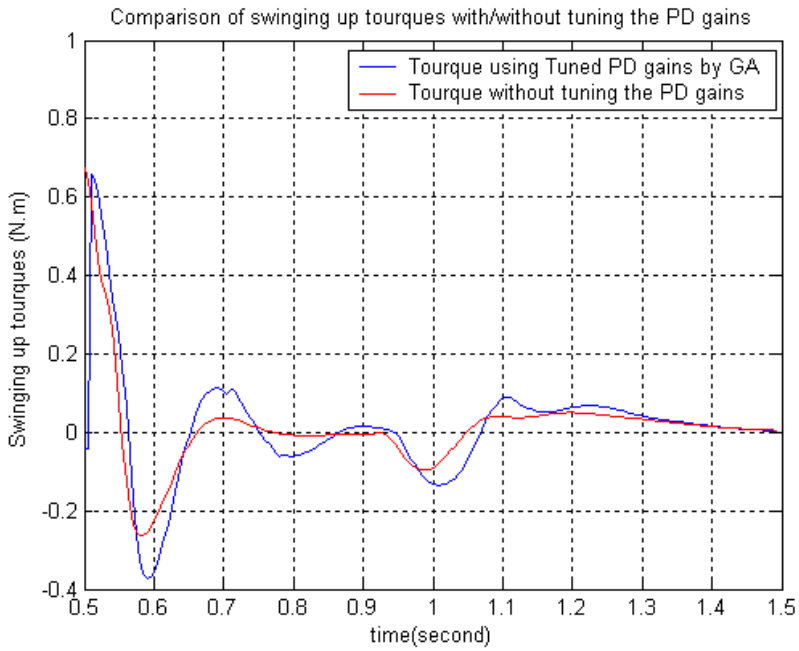


Figure 7.23 Comparison of swinging up torque with/without tuning the PD gains

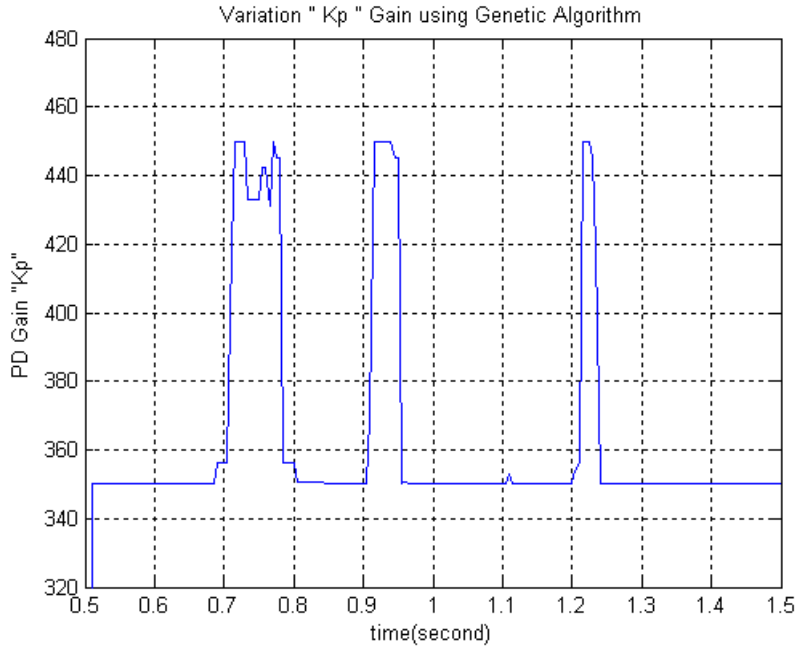


Figure 7.24 Variation of the Gain" K_p "using tuned PD Gains

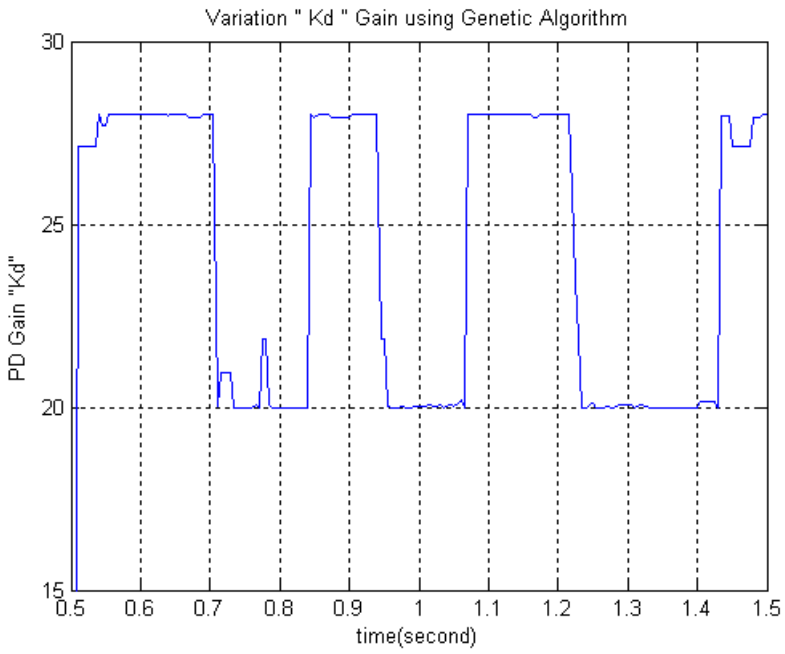


Figure 7.25 Variation of the Gain" K_d "using tuned PD gains

7.5. Simulation Results using Sliding Mode Balancing Controller (SMBC)

In this section we present the simulation results of the sliding mode balancing controller and compare these results with LQR results.

In order to select the best combination of parameters $\delta, \beta, \lambda_1, \lambda_2$, the summation of sliding surfaces for a period of time is found for a range of parameter values. Figure 7.26 shows the summation of sliding surfaces against the ratio of δ, β and λ_1, λ_2.

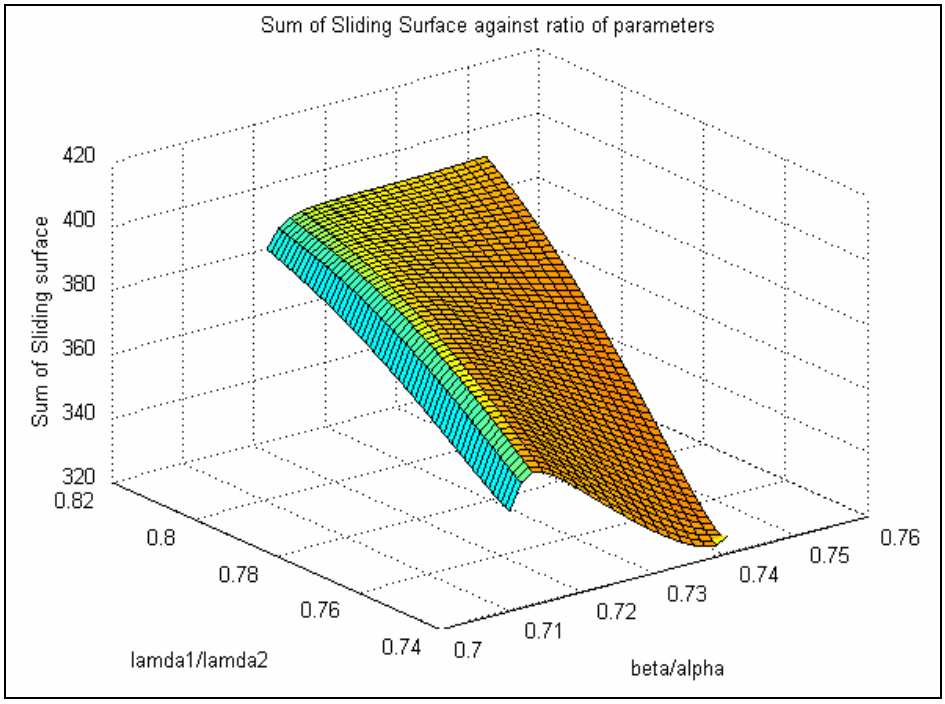


Figure 7.26 Sum of sliding surface against ratio of surface parameters

The minimum value of the sum of sliding surface is found on the following ratios:

$$\frac{\beta}{\delta} = 0.7387$$

$$\frac{\lambda_1}{\lambda_2} = 0.74$$

Let $\lambda_2 = 8.5$, $\delta = 0.62$, then $\lambda_1 = 6.29$, $\beta = 0.458$

The parameters μ , k are selected to be as follows:

$$\mu = 0.02, \quad k = 13$$

The balancing torque produced by the sliding mode controller is shown in Figure 7.27. The combination of balancing torques produced by both LQR and sliding mode controller is shown Figure 7.28. The chattering problem of sliding mode controller is magnified and shown in Figure 7.29. The joint angle of link one and joint angle of link two are shown in Figure 7.30 and Figure 7.31. The comparison of link two joint angle using LQR method and sliding mode balancing controller is shown in Figure

7.32. The angular velocity of link one and angular velocity of link two are shown in Figure 7.33 and Figure 7.34.

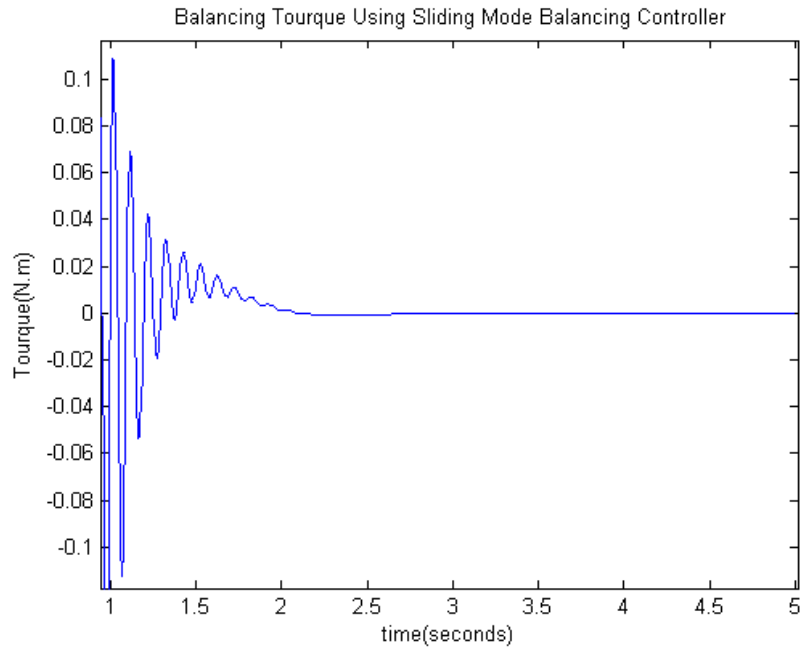


Figure 7.27 Balancing torque applied on joint one using sliding mode controller

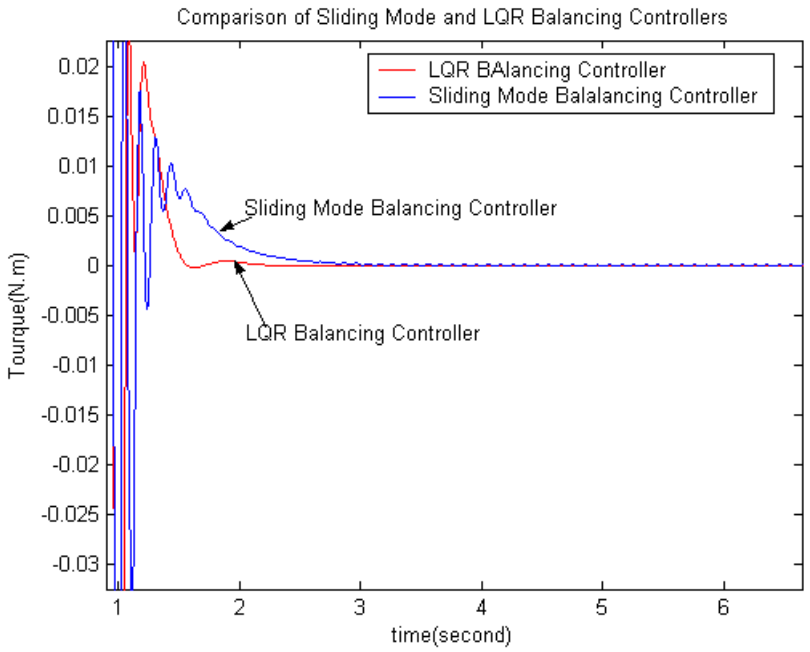


Figure 7.28 Comparison between sliding mode and LQR balancing controllers

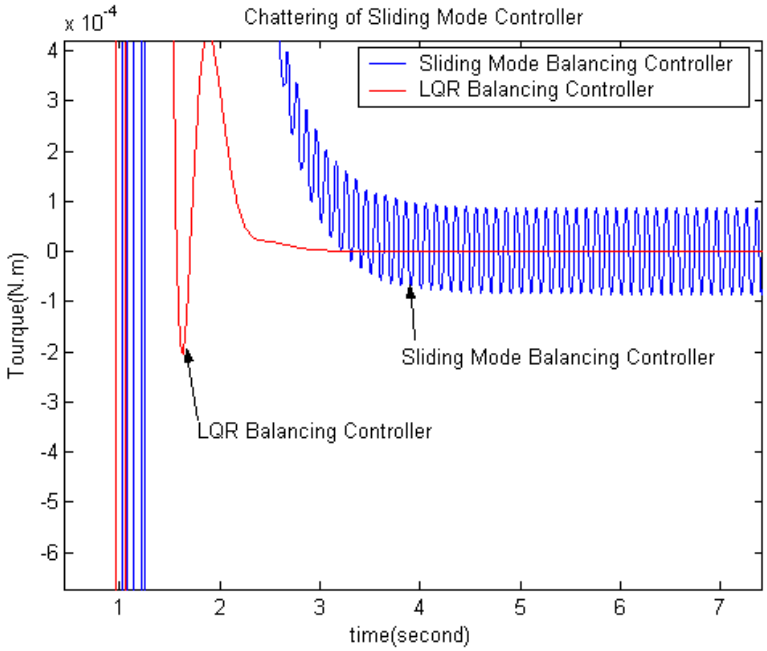


Figure 7.29 Chattering of sliding mode controller

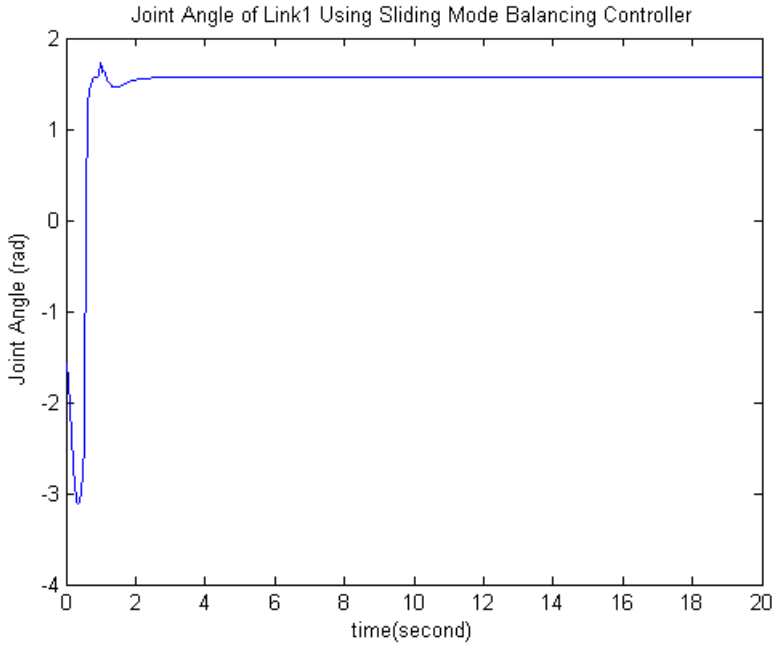


Figure 7.30 Joint angle of link one using the sliding mode balancing controller (SMBC)

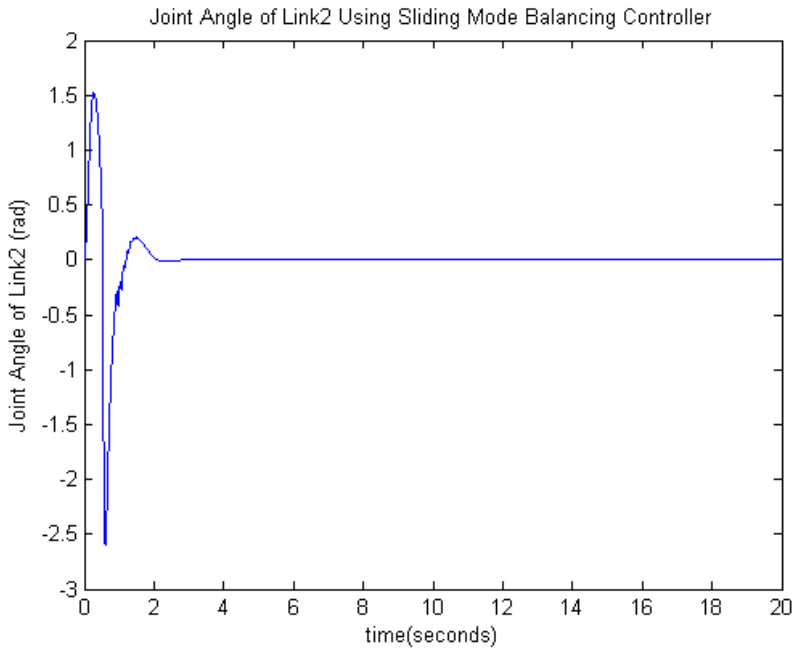


Figure 7.31 Joint angle of link two using sliding mode balancing controller (SMBC)

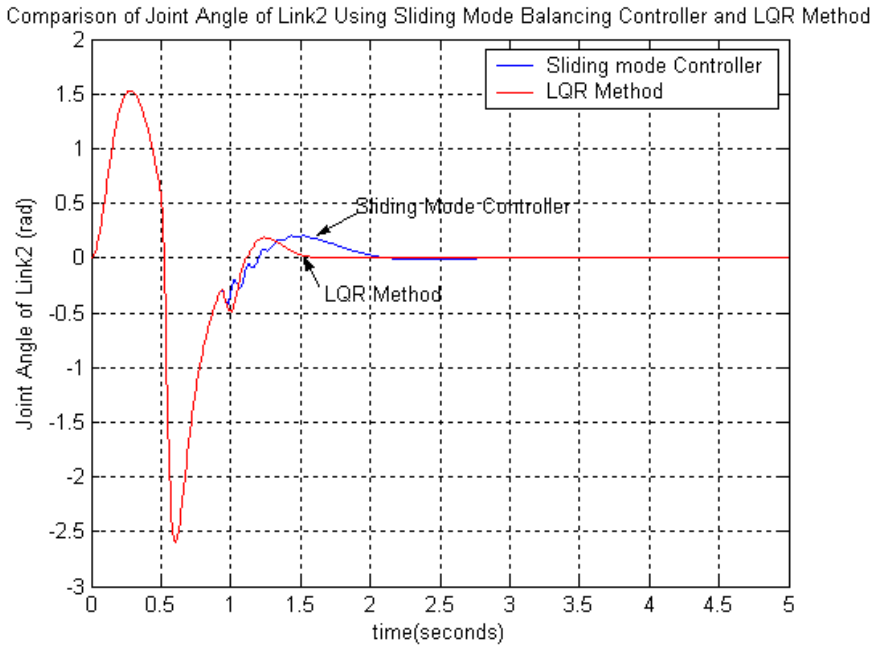


Figure 7.32 Comparison between joint angle of link two using sliding mode balancing controller and LQR technique

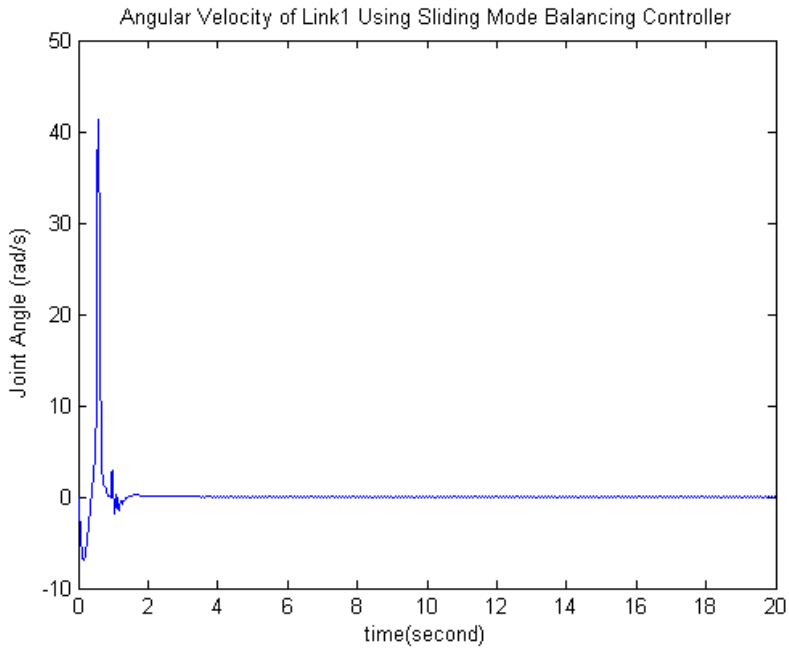


Figure 7.33 Angular velocity of link one using sliding mode balancing controller (SMBC)

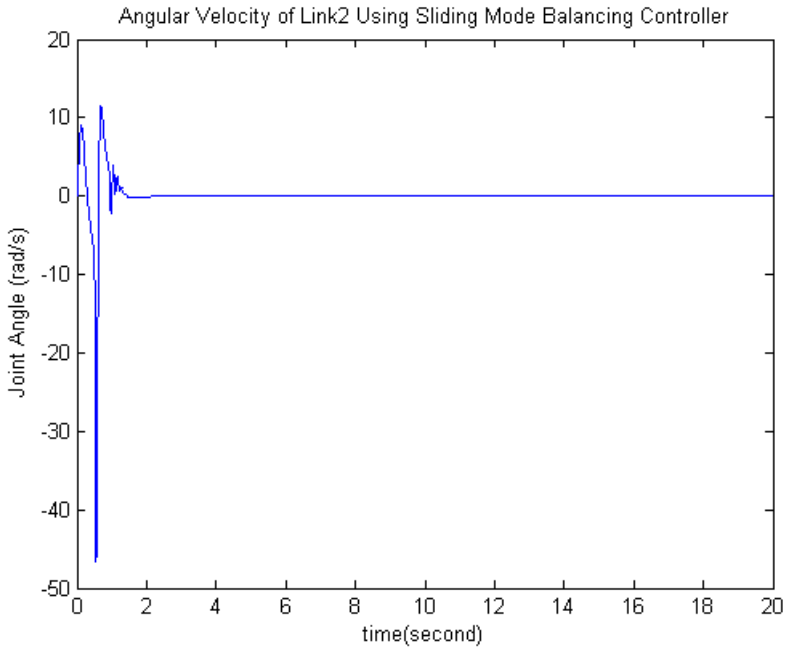


Figure 7.34 Angular velocity of link two using sliding mode balancing controller (SMBC)

As shown in the above figures sliding mode controller (SMBC) has successfully balanced the manipulator at the unbalance position.

7.6. Simulation Results Using a Genetic Sliding Mode Balancing Controller (GSMBC)

In this section we present the simulation results of combining genetic algorithm with sliding mode balancing controller and compare these results with LQR results and sliding mode solely.

The balancing torque produced by genetic sliding mode controller is shown in Figure 7.35. The joint angle of link one and angular velocity of link one are shown in Figure 7.36 and Figure 7.37. The joint angle of link two and angular velocity of link two are shown in Figure 7.38 and Figure 7.39. The comparison of joint angle for link one using genetic sliding mode, sliding mode controllers and LQR method is shown in Figure 7.40. The comparison joint angle for link two using genetic sliding mode, sliding mode controllers and LQR method is shown in Figure 7.41. As shown in Figure 7.40 and Figure 7.41 genetic sliding mode balancing controller reaches the sliding surface in shorter time than the sliding mode controller solely. In addition, the joint angles of link one and link two in sliding mode and genetic sliding mode are smoother than those in LQR algorithm.

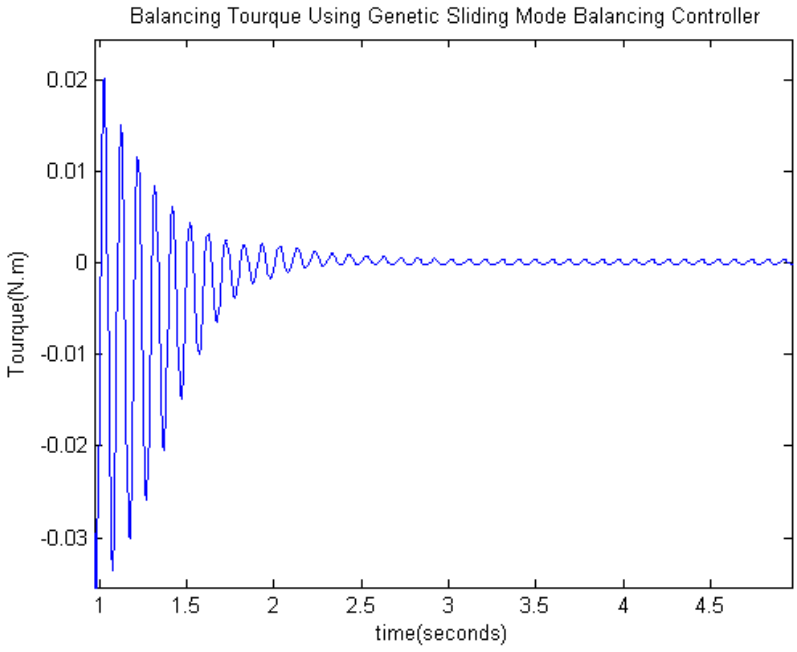


Figure 7.35 Balancing torque applied on joint one using genetic sliding mode controller (GSMBC)

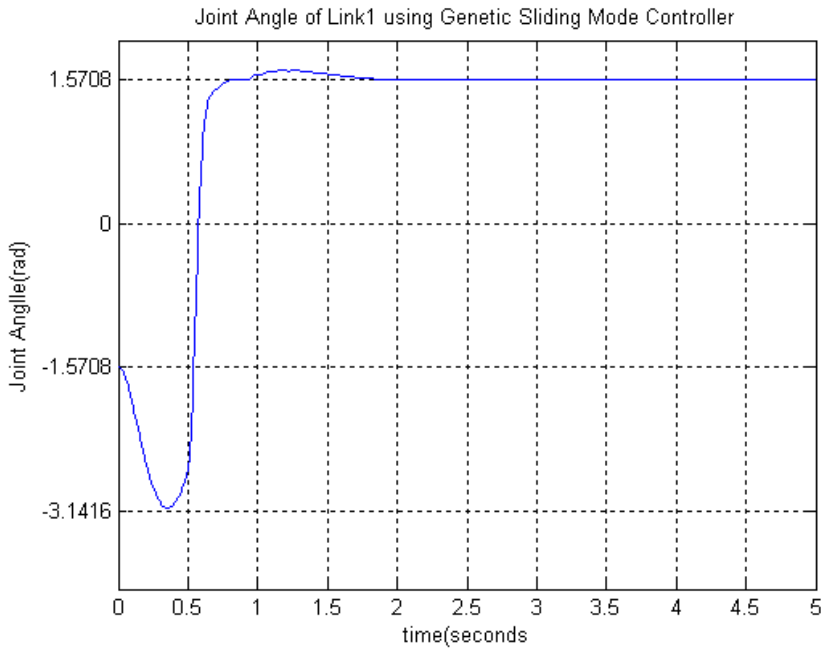


Figure 7.36 Joint angle of link one using genetic sliding mode balancing controller (GSMBC)

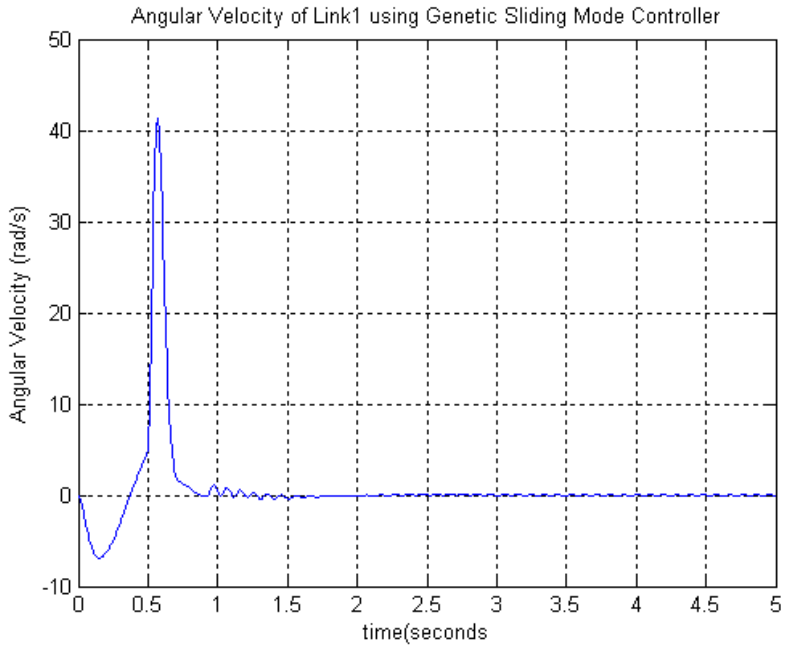


Figure 7.37 Angular velocity of link one using genetic sliding mode balancing controller (GSMBC)

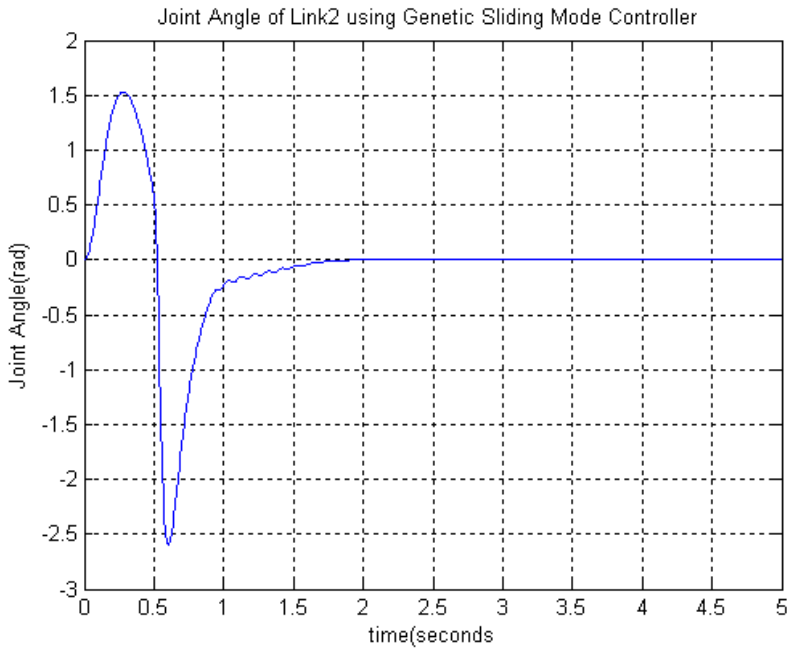


Figure 7.38 Joint angle of link two using genetic sliding mode balancing controller (GSMBC)

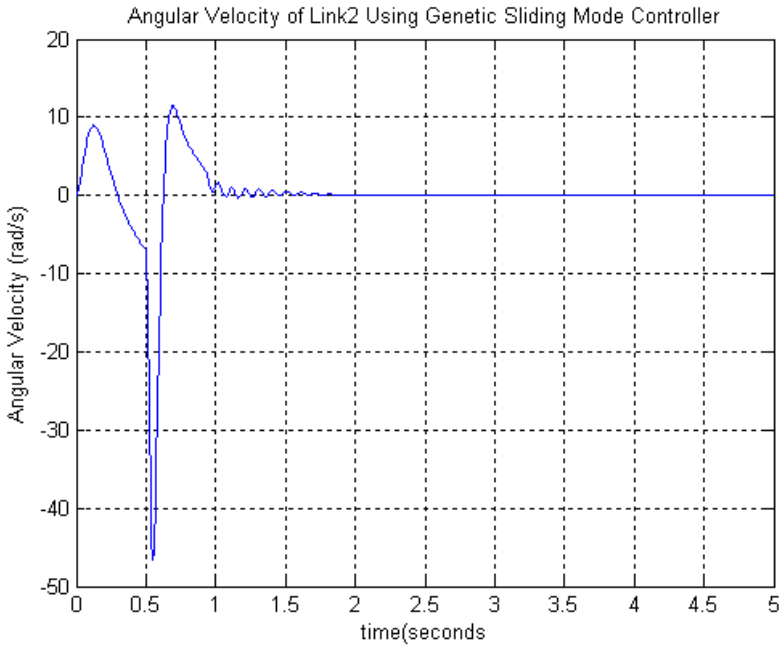


Figure 7.39 Angular velocity of link two using genetic sliding mode balancing controller (GSMBC)

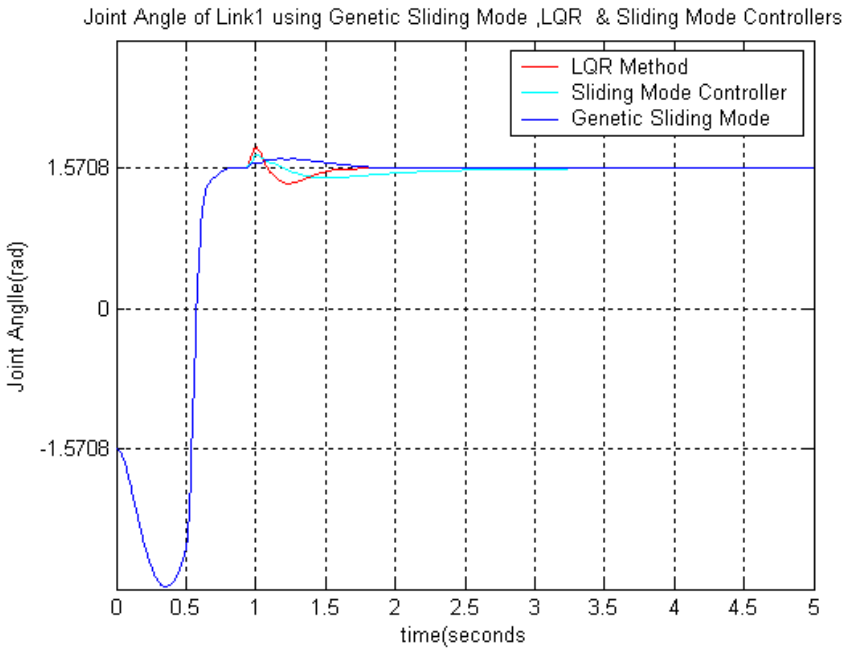


Figure 7.40 Joint angle of link one using LQR, sliding mode and genetic sliding mode balancing controllers

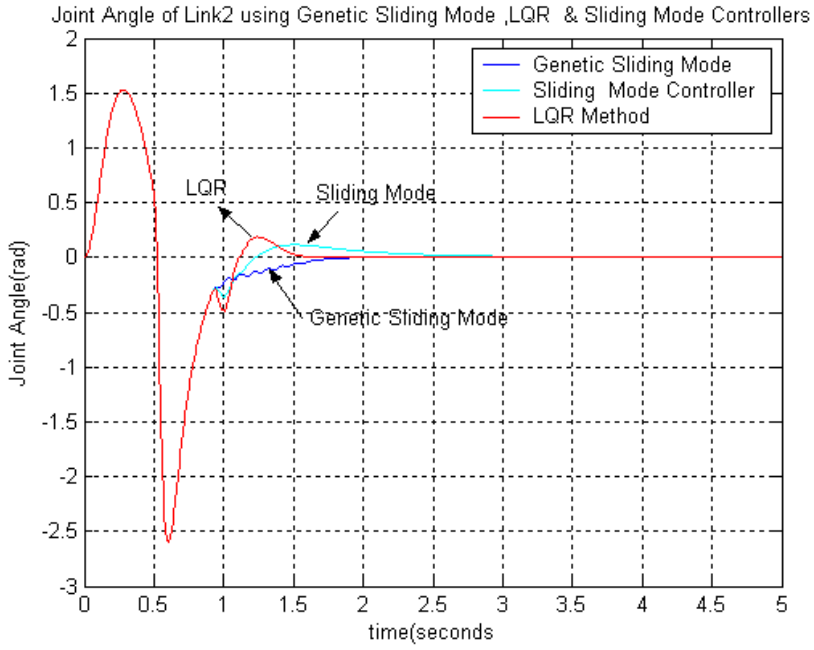


Figure 7.41 Joint angle of link two using LQR, sliding mode and genetic sliding mode balancing controllers

7.7. Simulation Results Using Neuro Fuzzy Balancing Controller

In this section we demonstrate the simulation results using Neuro Fuzzy controller to balance the manipulator at the upright position .As discussed in chapter six, the Neuro Fuzzy controller was trained by the results of LQR algorithm using ANFIS. For comparison purpose, feedback linearization technique is adopted for swinging up the manipulator.

The balancing torque of Neuro Fuzzy controller is shown in Figure 7.42. The combination of balancing torques produced by LQR algorithm and Neuro Fuzzy controller is shown Figure 7.43. The joint angle of link one and joint angle of link two are shown in Figure 7.44 and Figure 7.45 respectively. The angular velocity of link one and angular velocity of link two are shown in Figure 7.46 and Figure 7.47 respectively. As shown in the above mentioned figures, ANFIS was successful in training Neuro Fuzzy controller. Thus Neuro Fuzzy controller has successfully balanced the manipulator at the upright position.

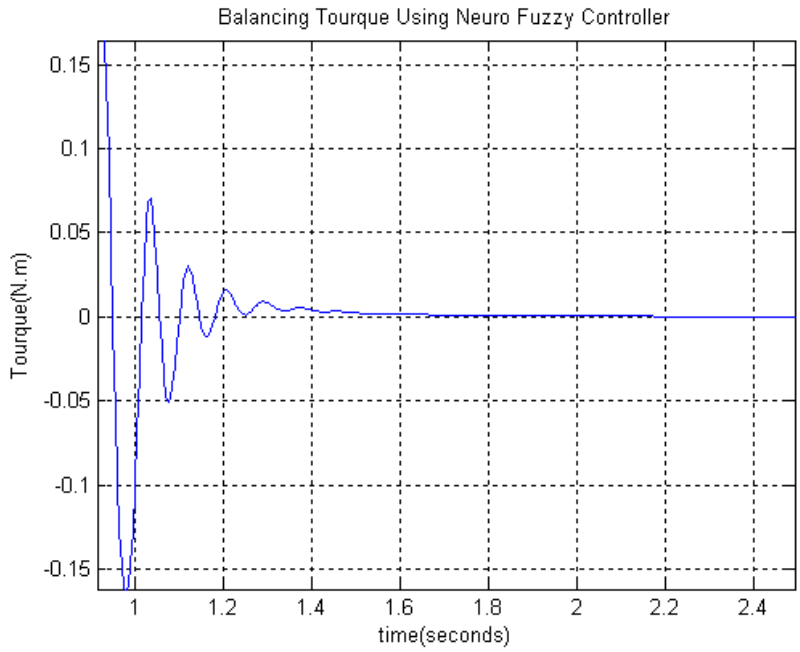


Figure 7.42 Balancing torque applied on joint one using Neuro Fuzzy controller (NFBC)

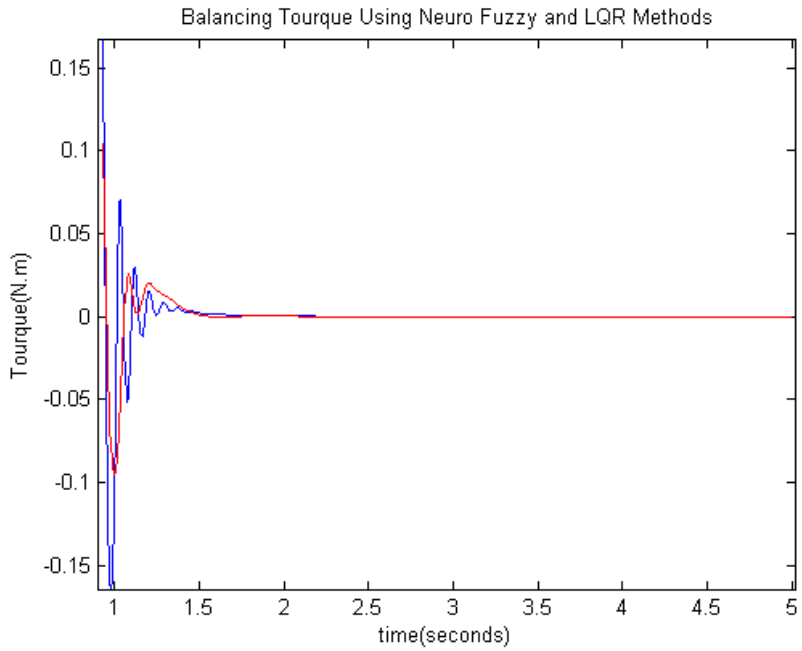


Figure 7.43 Comparison between the balancing torques produced by Neuro Fuzzy controller (NFBC) and LQR method

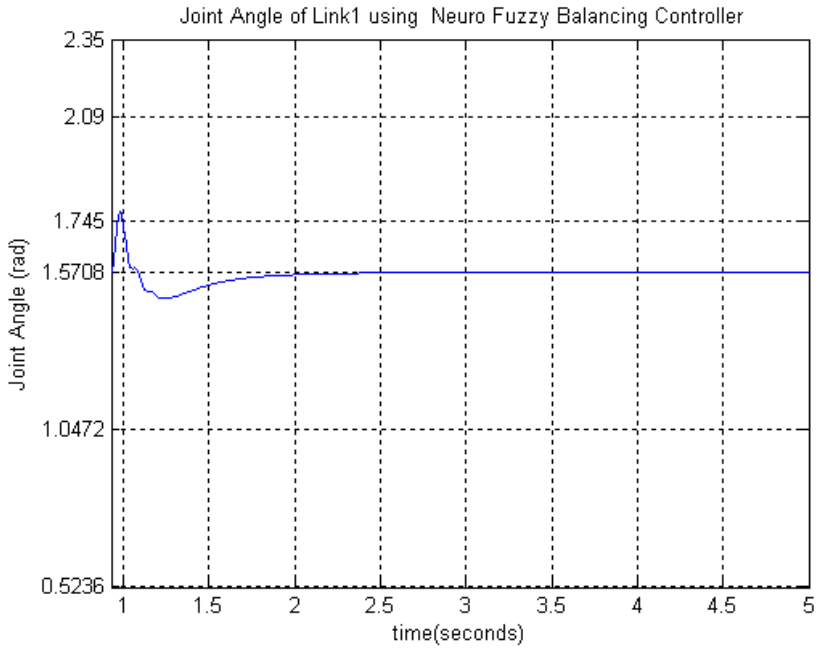


Figure 7.44 Joint angle of link one using Neuro Fuzzy balancing controller (NFBC)

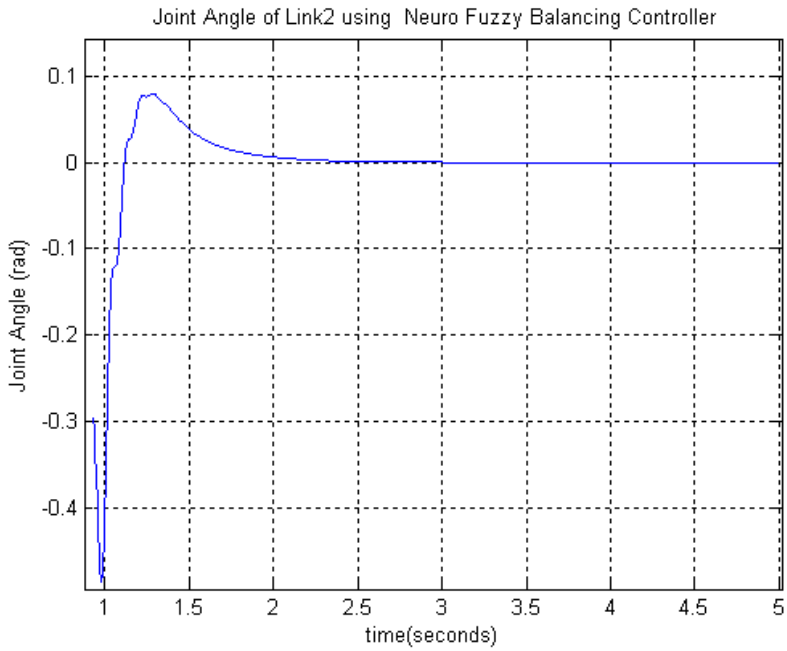


Figure 7.45 Joint angle of link two using Neuro Fuzzy balancing controller (NFBC)

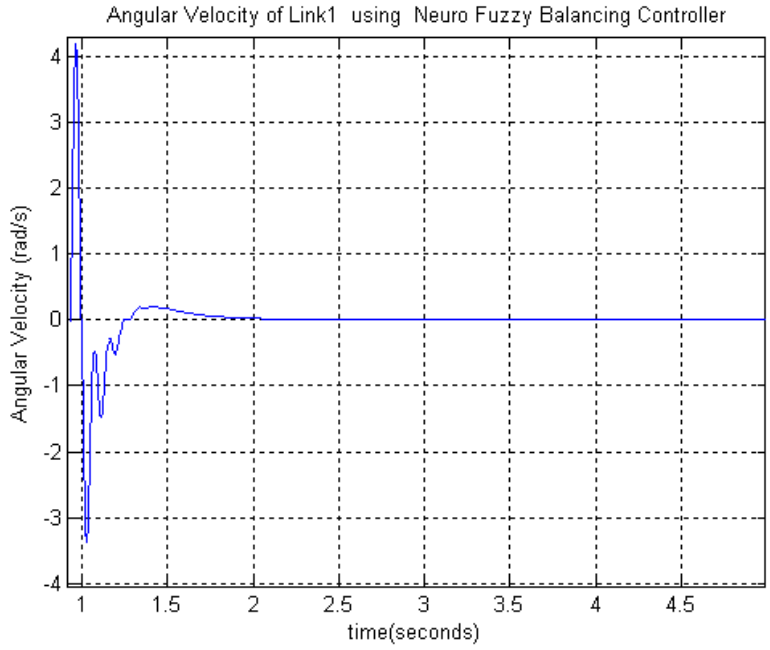


Figure 7.46 Angular velocity of link one using Neuro Fuzzy balancing controller (NFBC)

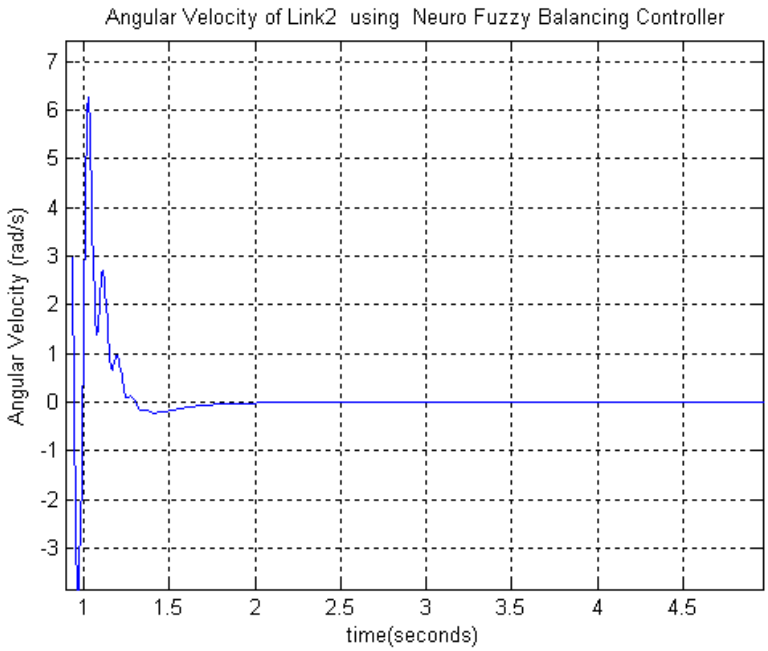


Figure 7.47 Angular velocity of link two using Neuro Fuzzy balancing controller (NFBC)

7.8. Effect of Noise on the Performance of Feedback Linearization Technique and LQR Method

In this section, we present the effect of signal noises on the performance of feedback linearization techniques and LQR method. A white noise is added to the encoder signals as shown in Figure 7.48. By addition of white noise the feedback linearization technique and LQR method are unable to stabilize the manipulator at the upright position Figure 7.49 and Figure 7.50 demonstrate this result. However as shown in Figure 7.49, feedback linearization technique was able to position the upright position but after longer time. Figure 7.49 shows the joint angle of link one in the presence of noise and Figure 7.50 shows the Joint angle of link two.

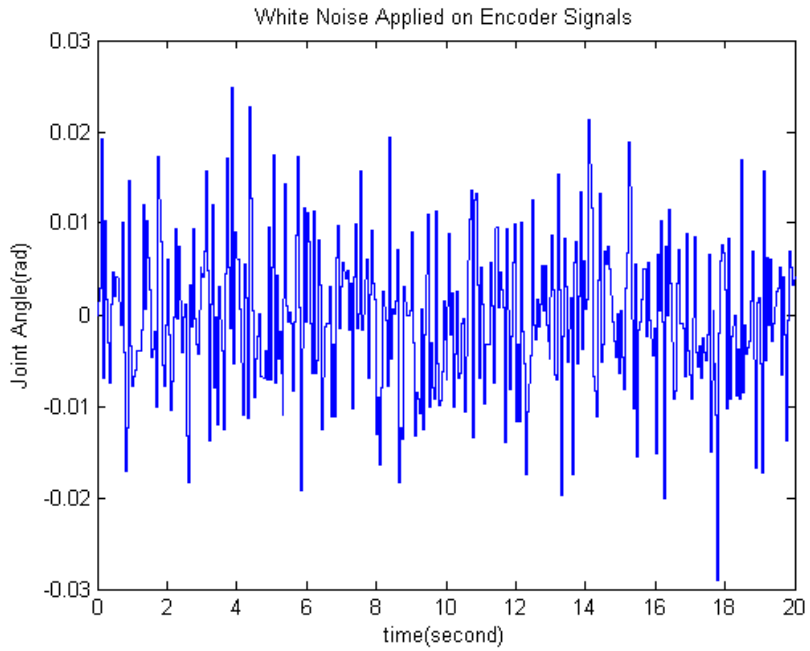


Figure 7.48 White noise applied on encoder signals

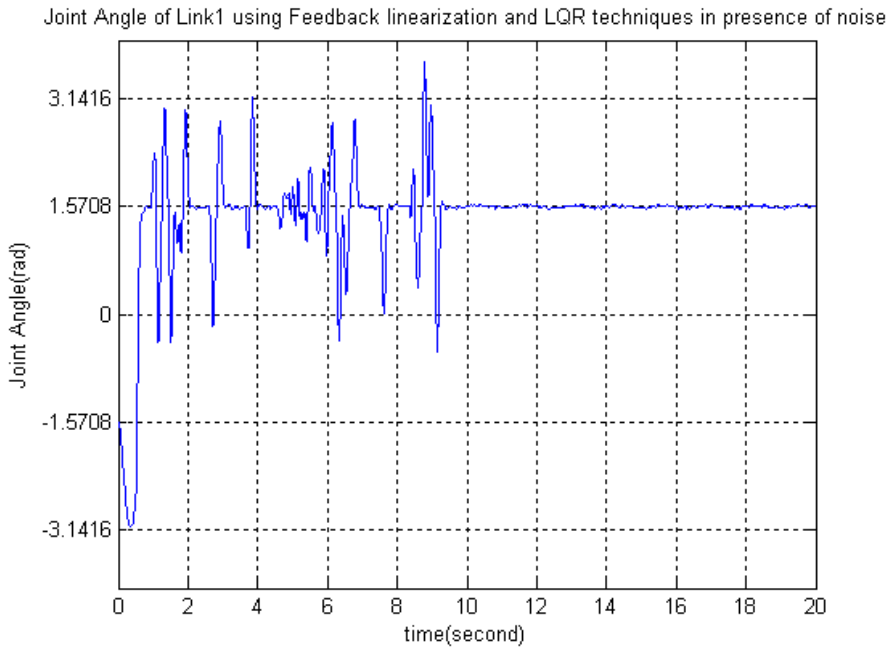


Figure 7.49 Joint angles of link one using feedback linearization technique and LQR method in presence of noise

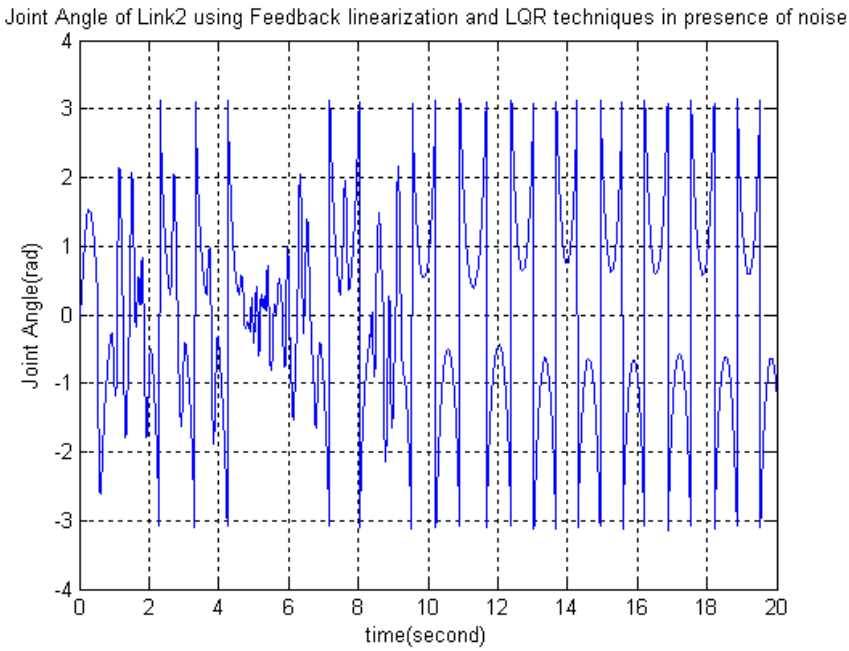


Figure 7.50 Joint angles of link two using feedback linearization technique and LQR method in presence of noise

7.9. Effect of Noise on the Performance of Sliding Mode Swinging up and Balancing Controllers

In this section, we present the effect of signal noises on the performance of sliding mode balancing and swinging up controllers. By adding white noise to the encoder signals, sliding mode controllers manage to tolerate the noises and they balance the manipulator at the upright position. Figure 7.51 and Figure 7.52 demonstrate this result. Figure 7.51 shows the joint angle of link one in the presence noise and Figure 7.52 shows the joint angle of link two

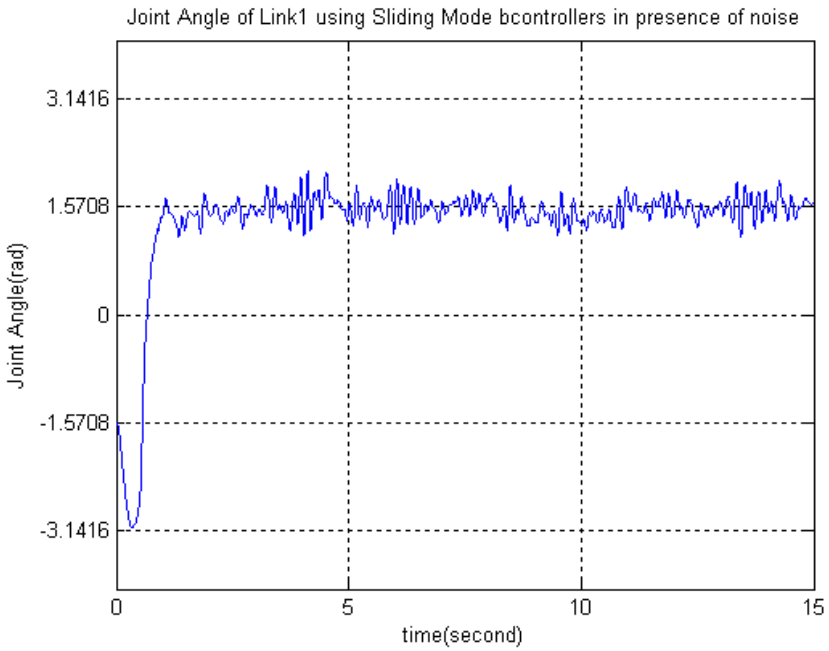


Figure 7.51 Joint angle of link one using sliding mode balancing and swinging up controllers in the presence of noise

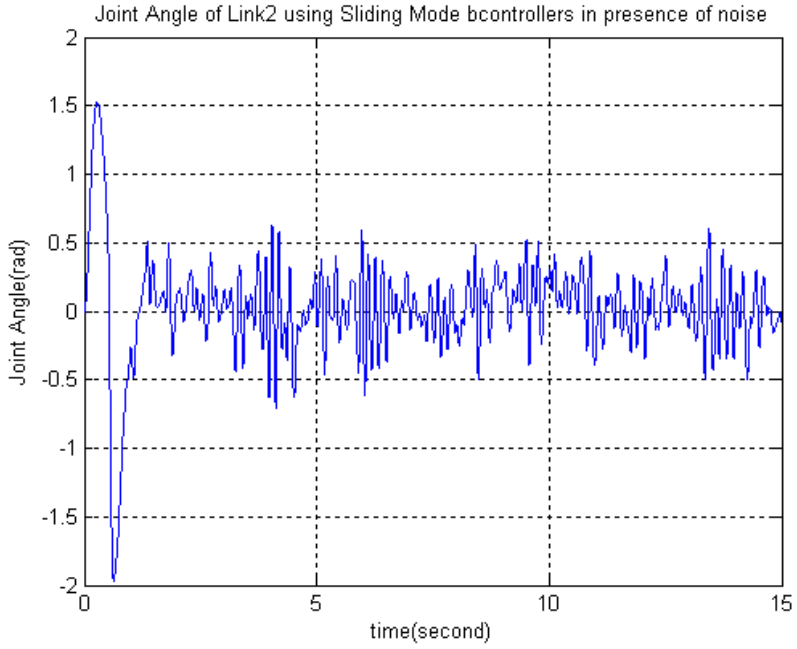


Figure 7.52 Joint angle of link two using sliding mode balancing and swinging up controllers in the presence of noise

7.10. Effect of Noise on the Performance of Genetic Sliding Mode Balancing Controllers (GSMBC)

In this section, we present the effect of signal noises on the performance of a genetic sliding mode balancing controller (GSMBC). By adding white noise to the encoder signals, the genetic sliding mode balancing controller manages to tolerate the noises and it relatively balances the manipulator at the upright position. Figure 7.53 and Figure 7.54 demonstrate this result. Figure 7.53 shows the joint angle of link one in the presence noise and Figure 7.54 shows the joint angle of link two.

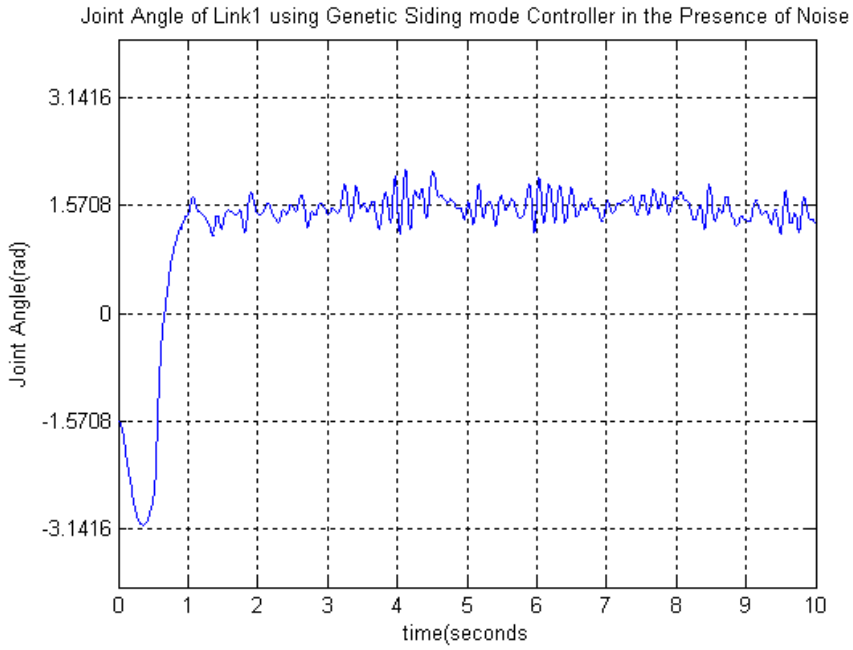


Figure 7.53 Joint angle of link one using genetic sliding mode balancing controller (GSMBC) in the presence of noise

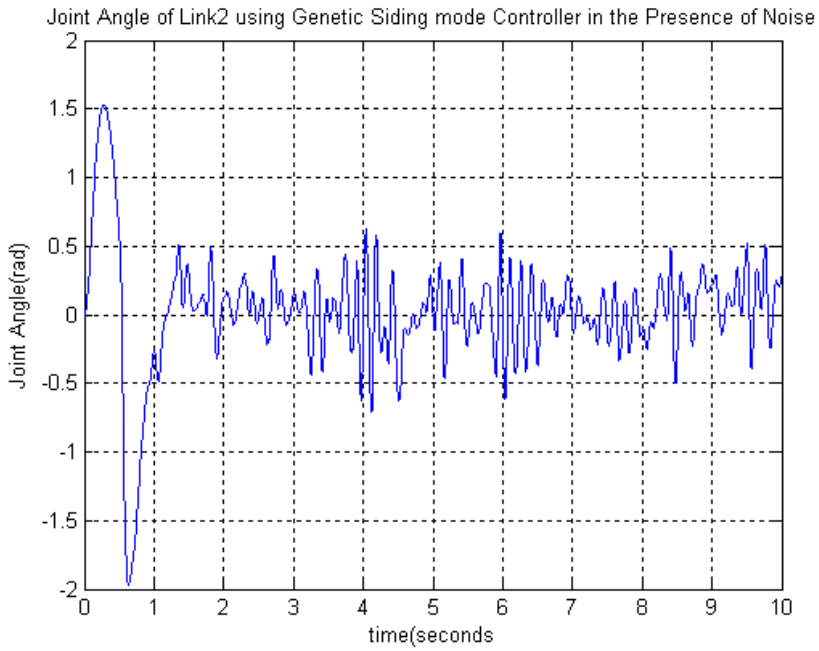


Figure 7.54 Joint angle of link two using genetic sliding mode balancing controller (GSMBC) in the presence of noise

7.11. Effect of Noise on the Performance of Neuro Fuzzy Swinging Up and Balancing Controllers

In this section, we present the effect of signal noises on the performance of Neuro Fuzzy controllers. By adding white noise to the encoder signals, Neuro Fuzzy controllers were unable to stabilize the manipulator at the upright position; however the swinging up Neuro Fuzzy controller is able to position the first link only at the upright position but after longer time. Figure 7.55 and Figure 7.56 demonstrate this result. Figure 7.55 shows the joint angle of link one in the presence noise and Figure 7.56 shows the joint angle of link two.

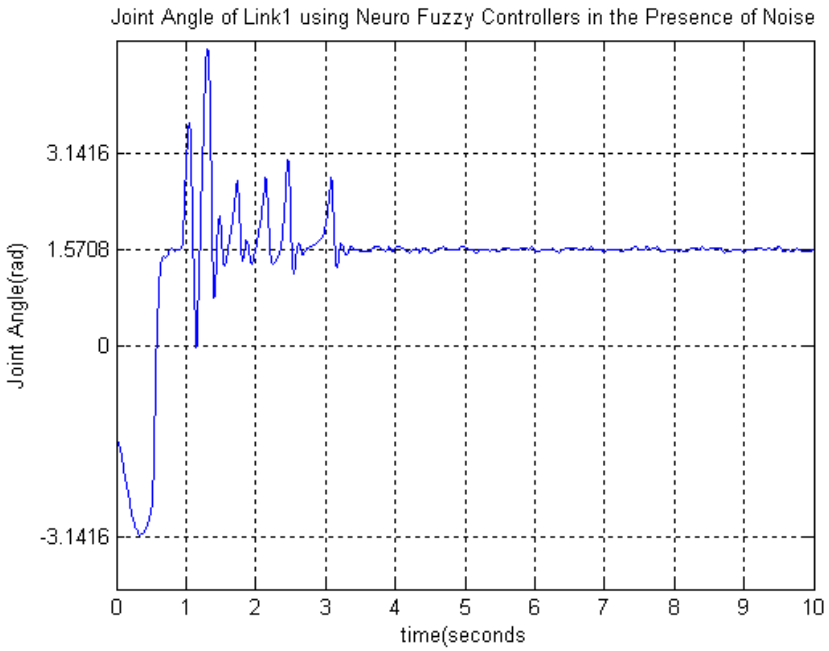


Figure 7.55 Joint angle of link one using Neuro Fuzzy controllers in the presence of noise

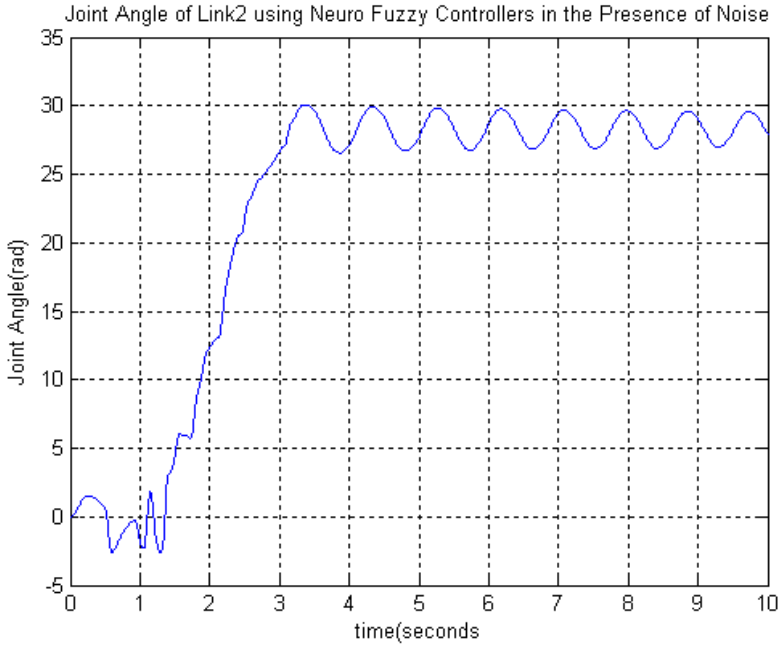


Figure 7.56 Joint angle of link two using Neuro Fuzzy controllers in the presence of noise

7.12. Effect of Noise on the Performance of Genetic Algorithm Used for Tuning the PD Gains of Feedback Linearization Technique

In this section, we present the effect of noise on the performance of genetic algorithm used for tuning the PD gains of feedback linearization technique. The feedback linearization is used for swinging up the manipulator to the upright position, the sliding mode balancing controller is used for balancing the manipulator at the upright position ,since it is robust and can tolerate noise as discussed in section 7.9.By adding of white noise to the encoder signals, the feedback linearization with tuned PD gains was able to swing up the manipulator to the upright position, then the sliding mode balancing controller managed to balance the manipulator at the upright position. Figure 7.57 and Figure 7.58 demonstrate this result. Figure 7.57 shows the joint angle of link one in the presence noise and Figure 7.58 shows the joint angle of link two.

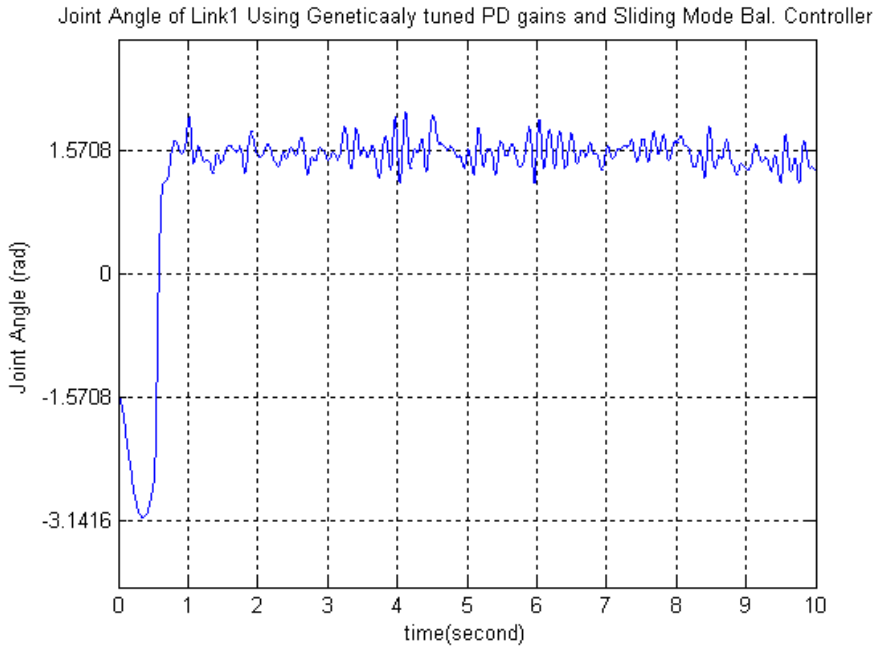


Figure 7.57 Joint angle of link one using GA used for tuning PD Gains of feedback linearization technique in the presence of noise

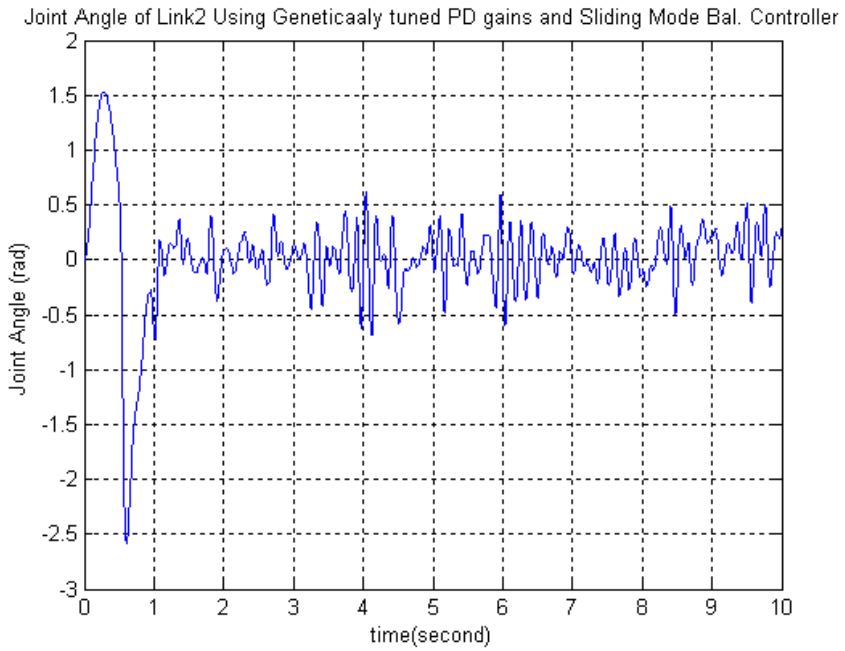


Figure 7.58 Joint angle of link two using GA used for tuning PD Gains of feedback linearization technique in the presence of noise

7.13. Effect of External Disturbance on the Performance of Feedback Linearization Technique and LQR Method

In this section and the next three sections, we study the effect of external disturbance on the performance of the control strategies considered in this thesis. The external disturbance is a force applied on the free end of the second link and it is transformed a disturbance torque applied at the second joint of the manipulator. The external disturbance is applied after stabilizing the manipulator at the upright position (after four seconds) as shown in Figure 7.59.

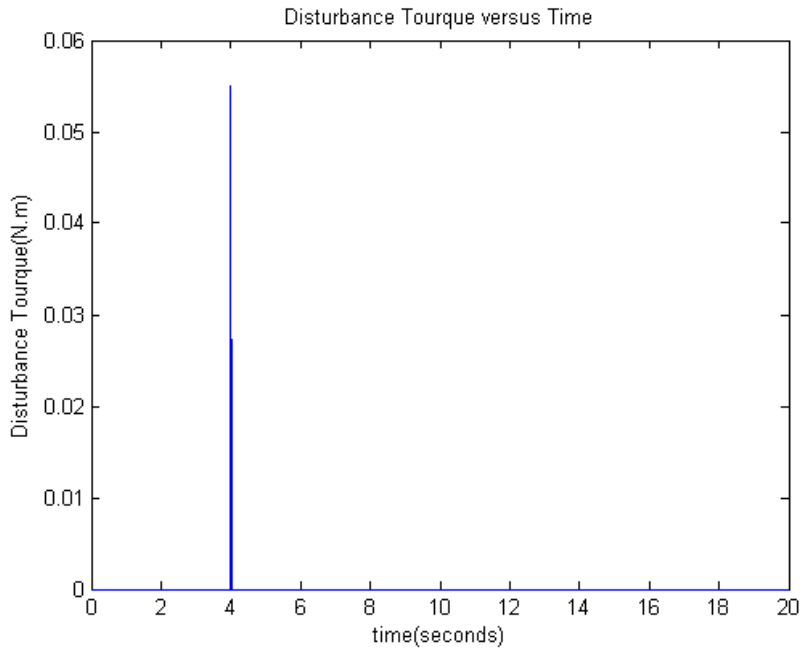


Figure 7.59 External disturbance torque

In this particular section we study the effect of external disturbance on performance of feedback linearization and LQR control strategies. As shown in Figure 7.60 and Figure 7.61feedback linearization and LQR methods could not maintain the manipulator at the upright position after the interference of disturbance torque.

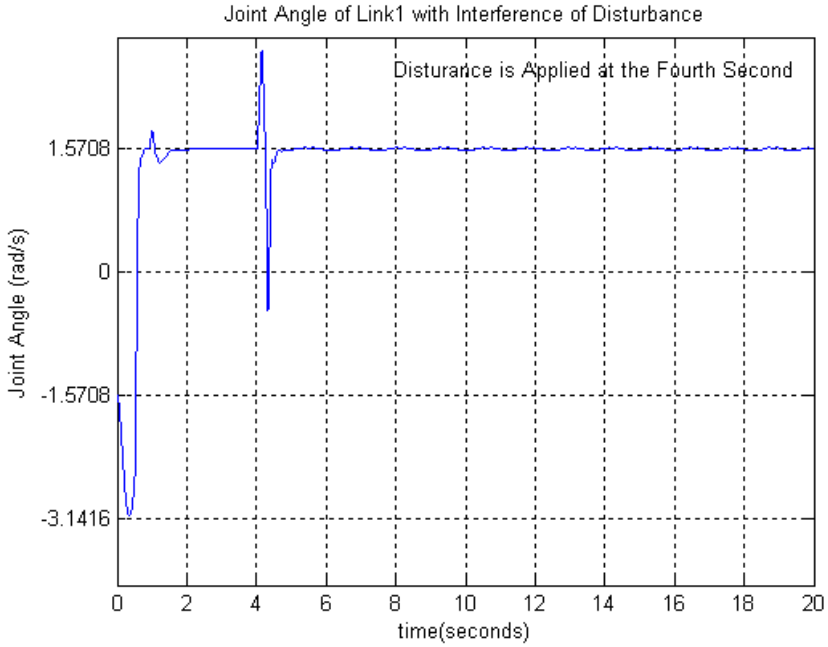


Figure 7.60 Joint angle of link one with the interference of disturbance using feedback linearization and LQR techniques

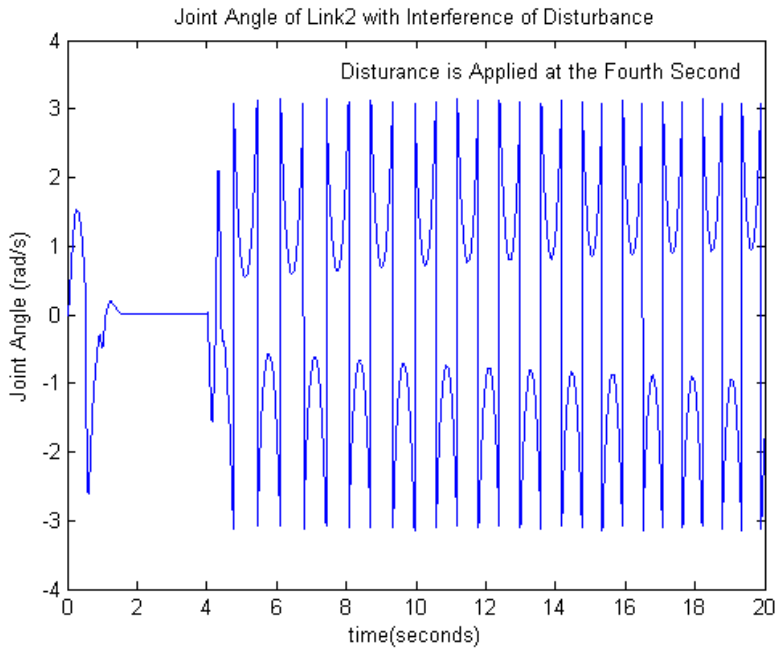


Figure 7.61 Joint angle of link two with the interference of disturbance using feedback linearization and LQR techniques

7.14. Effect of External Disturbance on the Performance of Sliding Mode Controllers

In this section we study the effect of external disturbance on performance of swinging up and balancing sliding mode controllers. As shown in Figure 7.62 and Figure 7.63, the sliding mode controllers were able to stabilize the manipulator at the upright position after the interference of disturbance. Figure 7.62 and Figure 7.63 show the joint angle of link one and link two respectively.

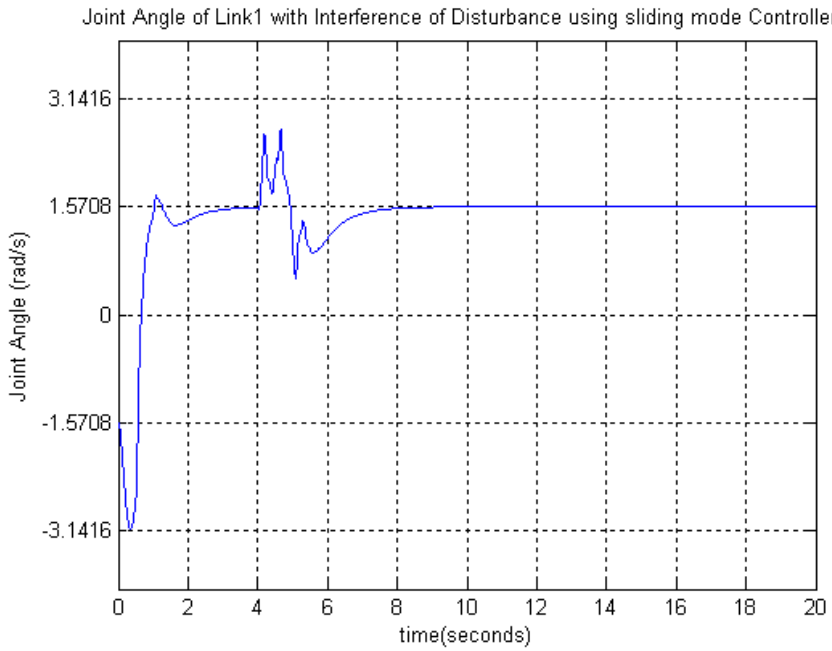


Figure 7.62 Joint angle of link one with the interference of disturbance using sliding mode controllers

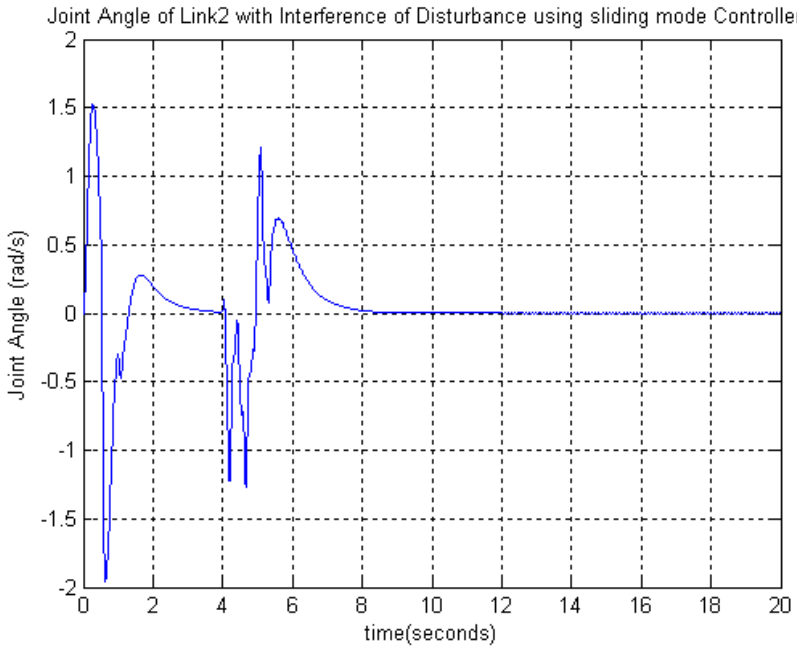


Figure 7.63 Joint angle of link two with the interference of disturbance using sliding mode controllers

7.15. Effect of External Disturbance on the Performance of Genetic Sliding Mode Balancing Controller

In this section we study the effect of external disturbance on performance of Genetic sliding mode balancing controllers, for swinging up the sliding mode controller is adopted. As shown in Figure 7.64 and Figure 7.65, the Genetic sliding mode balancing controller was able to stabilize the manipulator at the upright position after the interference of disturbance. Figure 7.64 and Figure 7.65 show the joint angle of link one and link one respectively.

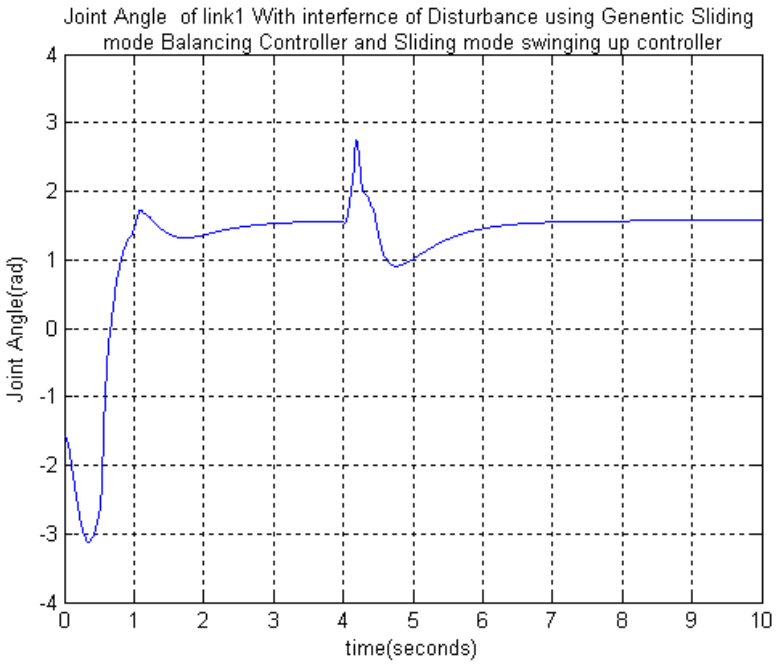


Figure 7.64 Joint angle of link one with the interference of disturbance using genetic sliding mode balancing controller and sliding mode swinging up controller

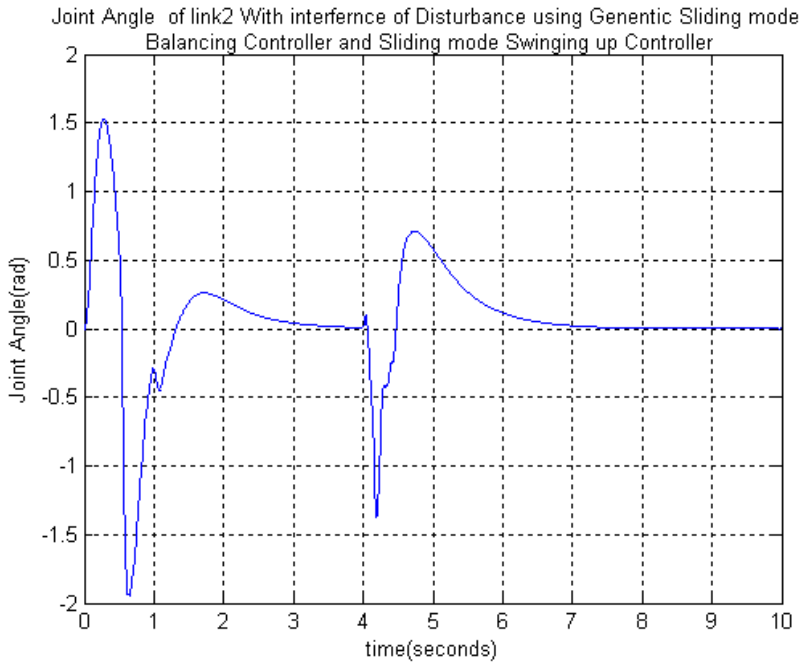


Figure 7.65 Joint angle of link two with the interference of disturbance using genetic sliding mode balancing controller and sliding mode swinging up controller

7.16. Effect of External Disturbance on the Performance of Neuro Fuzzy Controllers

To complete the study for the effect external disturbance, we study the effect of external disturbance on performance of Neuro Fuzzy Controllers. As shown in Figure 7.66 and Figure 7.67, Neuro Fuzzy Controllers could not to stabilize the manipulator at the upright position after the interference of disturbance. This result is expected since the training data of Neuro Fuzzy controllers does not include the states which the manipulator reached after disturbance. Figure 7.66 and Figure 7.67 show the joint angle of link one and link two respectively.

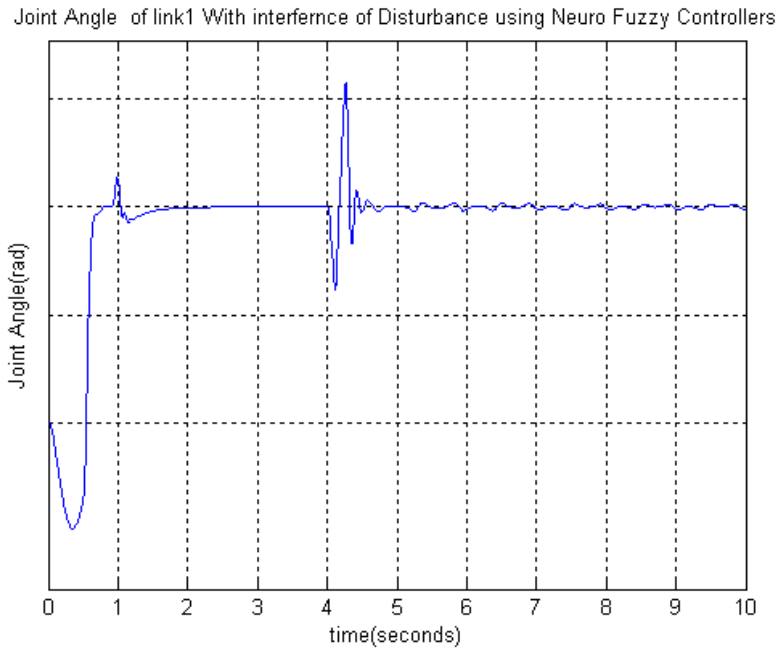


Figure 7.66 Joint angle of link one with the interference of disturbance using Neuro Fuzzy controllers

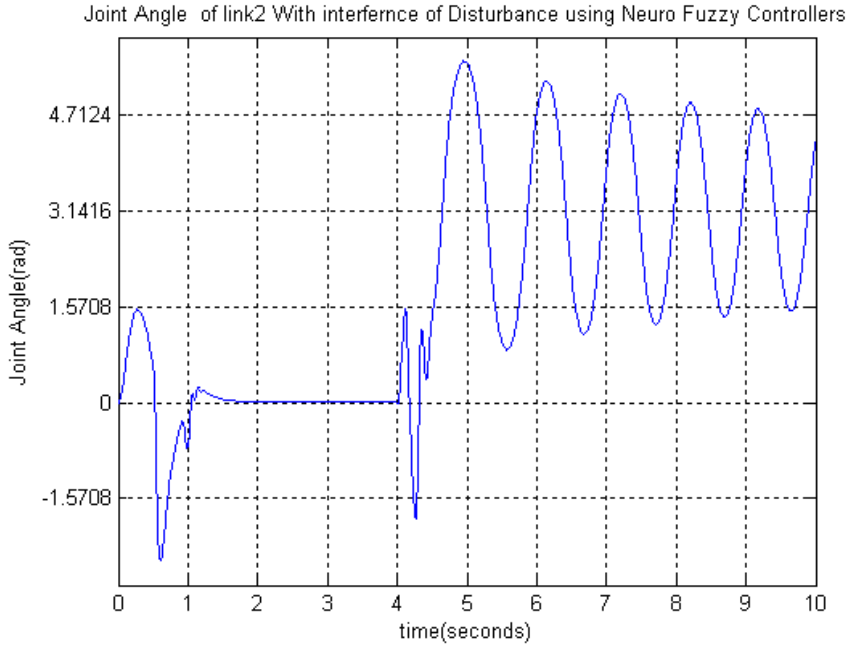


Figure 7.67 Joint angle of link two with the interference of disturbance using Neuro fuzzy controllers

CHAPTER 8: CONCLUSIONS

In this thesis we have studied various control strategies for two links under actuated manipulator. The conclusion of the results of each control strategy can be summarized as follows:

Feedback linearization and LQR methods are perfect for ideal cases, .i.e., all manipulator parameters are precisely known and noise level is limited. However, feedback linearization and LQR methods are very sensitive to the system noise and external disturbances. They could not stabilize the manipulator at the upright position in the presence of a certain levels of noise and disturbances. By tuning the PD gains using Genetic Algorithm, Feedback Linearization tolerates better the parameter uncertainties and noise. However the computational time of the genetic algorithm is relatively long which prevents the real time implication for short time constant systems.

Sliding mode swinging up controller and sliding mode balancing controller are successful techniques for balancing the manipulator at the upright position. They are robust and tolerate the certain levels of noise and external disturbance applied on other control techniques. However the sliding mode balancing controller have minor chattering problem around the sliding surface. To overcome the chattering and reduce the hitting time of sliding surface, the Genetic Algorithm is integrated with sliding mode controller. The genetic sliding mode controller improves the robustness, reduce the chattering and reduce the hitting time of sliding surface.

Neuro fuzzy swinging and balancing controllers are successful techniques in balancing the manipulator at upright position. But they are very sensitive to system noise and external disturbances. In other words, Neuro fuzzy controllers could tolerate the certain levels of noise and disturbances applied on other techniques. This could be justified by noticing that the membership functions and rules were trained from LQR

method and feedback linearization techniques, where the noise level was very limited and external disturbances were neglected.

REFERENCES

[[1] Abraham, A. (2002). Neuro Fuzzy Systems: State of art modeling techniques, *School of Computing and Information technology, Monash University, Australia.*

[2] Arai, H. (1996).Controllability of a 3-DOF manipulator with a passive joint under a nonholonomic constraint. *Proceedings of the 1996 IEEE InternationalConference on Robotics and Automation*, vol. 4, pp. 3707-3713

[3] Arai, H., & Tachi, S. (1991, August). Position control of a manipulator with passive joints using dynamic coupling. *IEEE Transactions on Robotics and Automation,* vol. 7, no. 4, pp. 528-534.

[4] Arai, H., & Tanie, K. (1993, February). Dynamic control of a manipulator with passive joints in operational space. *IEEE Transactions on Robotics and Automation,* vol. 9, no. 1, pp. 85-93.

[5] Bedrossian, N.S. (1993). Nonlinear control of an underactuated two-link manipulator. *Proceedings of American Control Conference*, vol. 3, pp. 2234-2238.

[6] Bergerman, M. (1996, December). Dynamics and control of underactuated manipulators. *Pennsylvania, Pittsburgh*.

[7] Block, D.J. (1996). Mechanical design and control of pendubot.*University of Illinois at Urbana –Champaign.*

[8] Brown, S.C., & Passino, K.M. (1997). Intelligent Control for an Acrobat. *Department Electrical Engineering, Ohio University.*

[9] Bugeja, M. (2003). Non-linear swing up and stabilizing control of an Inverted pendulum. *EUROCON, Ljubljana Slovenia*.

[10] Carroll, K. L. (1998). Control algorithms for stabilizing underactuated robots. *Journal of Robotic System*s 15 (12) 681-697 (1998), John Wiley & Sons, Inc.

[11] Erdem, E. B. (2001). Analysis and real-time implementation of state-dependent Riccati equation controlled systems. *University of Illinois at Urbana-Champaign.*

[12] Fantoni, I. &. Lozano, R. (2002). *Non-linear control for underactuated mechanical systems*. London. Springe-Verlag London Limited.

[13] Franklin, J.F., & Powil, G.V. ((2001). *Feedback control of dynamics system (4th ed.).* New Jersey, Prentice-Hall International INC.

[14] Fujinaka, T., & Kishda, Y. (2000). Stabilization of Double Inverted Pendulum with self tuning Neuro-PID.*IEEE.*0-7695-0619-4/00.

[15] Giua, C. G., & Usai, A. (1998). An implicit gain-scheduling controller for cranes, *IEEE Trans. Control Syst. Technol*., 6,(1),pp.15–20.

[16] ILIEV, B. (2004). Minimum-time sliding mode control of robot manipulator, *Obrebo University.*

[17] Iwashiro, M. & Furuts, K. (1996). Energy based control of pendulum .*IEEE* 0-7803-2975-9/9.

[18] Keleher, P.G, & Stoiner, R.J (2002). Adaptive terminal sliding mode control of a rigid robotic manipulator with uncertain dynamics incorporating constraint inequalities. *Anziam* J.43 (E) pp EE102-E153.

[19] Lai, X., &.Gai, Z. (2000).Motion control of acrobat using fuzzy and sliding mode control strategy. *Proceedings of 3rd Asian Control Conference,* pp.2340-2345 July 4-7, 2000, Shanghai China.

[20] Marco, P., & Raul, L. (2002). Application of several Neuro control schemes to 2-DOF. *Instituto Tecnológico de Estudios Superiores de Occidente, Guadalajara Jal., México.*

[21] Moore, M., L., & Passino, K. M. (2001). Genetic Adaptive control for an Inverted wedge. *Eliever Science Ltd*. pp. 0952-1976(00)00046-6

[22] Mullhuapt, Ph., & Srinvasan, B. (2002). Analysis of exclusively kinetic two link underactuated mechanical systems. *Automatica* 38 (2002) 1565-1573.0

[23] Niemann, H., & Stoustrup, J. (2004). Passive fault tolerant control of double inverted pendulum –a case study example. *Orested .DTU Automation Technical university of Denmark.*

[24] Ogata, K. (1995). *Discrete time control signal (2nd ed)*.New Jersey. Prentice-Hall International INC.

[25] Ortega, R., & Spong, M.W. (2000, January 12). Stabilization of underactuated mechanical system via interconnecting damping assignment. *Systems & Control Letters* 45 (2002) 371– 385.

[26] Passino, K. M., & Yurkovich, S. (1998). *Fuzzy control*. California, Addison Wesley Longman.

[27] Pirjanian, p. (2004). Genetic Algorithm. Retrieved May 17, 2005, from . ilab.usc.edu/classes/2004cs460/notes/

[28] Poznyak, E. A., & Yunes, A. G. (1998). Adaptive architecture of polynomial artificial neural network to forecast nonlinear time series .*Laboratorio de Investigación y Desarrollo de Tecnología Avanzada.*

[29] Rojo, F. L., & Sanchez, E. (2002). Real-time Neuro Fuzzy control of underactuated manipulator. *ITESO University* A.P 31-175.45051 Gudaajara, Jal. Mexico

[30] Scheme, N., & Heimann, J. (2000). Dynamics and control of underactuated manipulation systems: A discrete approach. *Robotics and Automation Systems* 30 (2000) 237-248.

[31] Slotine, J. J., & Li, W. (1991). Applied nonlinear control. New Jeruey:Prentice-Hall International INC.

[32] Spong, M.W., & Vidyasgar, M. (1992). *Robot dynamics and control*. New York, John Wiley and Sons.

[33] Spong, M.W. (1994). Swing up of control the Acrobat. *Proceedings of the 1994 International Conference on Robotics and Automation,* pp. 2356-2361.

[34] Spong, M.W., & Praly, L. Control of underactuated mechanical system using switching and saturation. *National Science Foundation under graints* CMS-9402229 and INT-9415757.

[35] Toumi, K. Y., & Asada, H. (1987, September). The design of open-loop manipulator arms with decoupled and configuration-invariant inertia tensors. *Transactions of the ASME, Journal of Dynamic Systems, Measurement, and Control*, vol. 109, pp.268-275.

[36] Wang, W., & Zhaoand, J. (2004,November). Design of a stable sliding-mode controller for a class of second- order underactuated systems .*IEE Proc.-Control Theory* Appl.,Vol.151, No.6.

[37] Wong, C.C., & Chang, S.Y. (1998). Parameter selection in sliding mode control using genetic algorithm. *Tamkang Journal of Science and Engineering,* Vol.1, No.2 pp. 115-122 (1998).

[38] Yan, O. (2003, December). Output tracking of underactuated rotary inverted pendulum by non linear controller. *Proceedings of the 42nd IEEE Conference on Decision and Control*, Muai Hawaii, USA .

[39] Yu, K.H., & Takahashi, T. (1995). Dynamics and motion control of a two-link robot manipulator with a passive joint. Proceedings of the 1995 International Conference on Intelligent Robots and Systems, vol. 2, pp. 311-316.

[40] Zhong, W, & Rock, H. (2001). Energy and passivity based control of double Inverted pendulum on a Cart. *IEEE*.0-7803-6733-2/01/ 2001.

APPENDIX A

S_FUNCTIONS IN C PROGRAMMING LANGUAGE

A.1 SLIDING MODE BALANCING CONTROLLER

```
/*
 *
 *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 2.0 ---
 *
 *   This file is a wrapper S-function produced by the S-Function
 *   Builder which only recognizes certain fields.  Changes made
 *   outside these fields will be lost the next time the block is
 *   used to load, edit, and resave this file. This file will be overwritten
 *   by the S-function Builder block. If you want to edit this file by hand,
 *   you must change it only in the area defined as:
 *
 *       %%%-SFUNWIZ_wrapper_XXXXX_Changes_BEGIN
 *           Your Changes go here
 *       %%%-SFUNWIZ_wrapper_XXXXXX_Changes_END
 *
 *   For better compatibility with the Real-Time Workshop, the
 *   "wrapper" S-function technique is used.  This is discussed
 *   in the Real-Time Workshop User's Manual in the Chapter titled,
 *   "Wrapper S-functions".
 *
 *   Created: Sat Jan  7 12:16:31 2006
 */

/*
 * Include Files
 *
 */
#include "tmwtypes.h"
/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include "mex.h"
#include <stddef.h>
#include <_syslist.h>
#include <stdlib.h>
#include <math.h>
#include"tmwtypes.h"
#include "matrix.h"
 # define POP_SIZE 20
# define MAX_GENERATION 100
# define NUM_TRAITS 2

/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 4
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
SRC_PATH,INC_PATH,LIB_PATH;

/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */
/*
 * Output functions
 *
 */
void cslide2_Outputs_wrapper(const real_T *u0,
                    real_T *y0,
                    const real_T *xD)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
```

```
  double h1,h2,h3,h4,h5,g,den;
  double Ueq1,Ueq2,s1,s2,S,c1,c2,af,beta,mu,k,stau;
  int sgn;
    double pi=22.0/7;

  double B[2]={0};
   double F[2]={0};

  //***********************************************

  h1=6.0000e-004;
  h2=3.6245e-004;
  h3= 1.9200e-004;
  h4= 0.0068;
  h5=0.0024;
  g=9.81;

  den=(h1*h2-pow(h3*cos(u0[2]),2));

  F[0]=(h2*h3*sin(u0[2])*pow((u0[1]+u0[3]),2)+pow(h3,2)*cos(u0[2])*sin(u0[2])*pow(u0[1],2)-
h2*h4*g*cos(u0[0])+h3*h5*g*cos(u0[0]+u0[2]))/den;
  F[1]=(-h3*(h2+h3*cos(u0[2]))*sin(u0[2])*pow((u0[1]+u0[3]),2)-
(h1+h3*cos(u0[2]))*h3*sin(u0[2])*pow(u0[3],2)+(h2+h3*cos(u0[2]))*h4*g*cos(u0[0])-
(h1+h3*cos(u0[2]))*h5*g*cos(u0[0]+u0[2]))/den;
   B[0]=h2/den;
   B[1]=-(h2+h3*cos(u0[2]))/den;
  c1=6.5673;
  c2=8.5511;
  af=0.62;
  beta=0.45;
  mu=0.002;
  k=13.0;
  s1=c1*(u0[0]-pi/2)+u0[1];
  s2=c2*u0[2]+u0[3];
  Ueq1=-(F[0]+c1*u0[1])/B[0];
  Ueq2=-(F[1]+c2*u0[3])/B[1];
  S=af*s1+beta*s2;
  if(S>=0){
          sgn=1;}
  else{
                  sgn=-1;
  }
  stau=(af*B[0]*Ueq1+beta*B[1]*Ueq2-mu*sgn-k*S)/(af*B[0]+beta*B[1]);
 y0[0]=stau;

/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

/*
 * Updates function
 *
 */
void cslide2_Update_wrapper(const real_T *u0,
                    const real_T *y0,
                    real_T *xD)
{
/* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO _END */
```

A.2 SLIDING MODE SWINGING UP CONTROLLER

```
/*
 *
 *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 2.0 ---
 *
 *   This file is a wrapper S-function produced by the S-Function
 *   Builder which only recognizes certain fields.  Changes made
 *   outside these fields will be lost the next time the block is
 *   used to load, edit, and resave this file. This file will be overwritten
 *   by the S-function Builder block. If you want to edit this file by hand,
 *   you must change it only in the area defined as:
 *
 *       %%%-SFUNWIZ_wrapper_XXXXX_Changes_BEGIN
 *           Your Changes go here
 *       %%%-SFUNWIZ_wrapper_XXXXXX_Changes_END
 *
 *   For better compatibility with the Real-Time Workshop, the
 *   "wrapper" S-function technique is used.  This is discussed
 *   in the Real-Time Workshop User's Manual in the Chapter titled,
 *   "Wrapper S-functions".
 *
 *   Created: Sat Jan  7 14:19:38 2006
 */


/*
 * Include Files
 *
 */
#include "tmwtypes.h"
/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include "mex.h"
#include <stddef.h>
#include <_syslist.h>
#include <stdlib.h>
#include <math.h>
#include"tmwtypes.h"
#include "matrix.h"
 # define POP_SIZE 10
# define MAX_GENERATION 10
# define NUM_TRAITS 2



/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 9
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
SRC_PATH,INC_PATH,LIB_PATH;



/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */

/*
 * Output functions
 *
```

```
 */
void cfd_slideT1_Outputs_wrapper(const real_T *u0,
                    real_T *y0,
                    const real_T *xD)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */

                      //*********************************************

                    double h1,h2,h3,h4,h5,g,den;
                    double Ueq1,s1,c1,mu,k,tau,taup;
                    int sgn;
                      double pi=22.0/7;

                    double B[2]={0};
                     double F[2]={0};
                     double D[2][2]={0};
                     double C[2][2]={0};
                     double G[2]={0};

                     double Ts, FB,v1,d11_b,c11_b,G_b,c12_b,b1;

                    //************************************************

                    h1=6.0000e-004;
                    h2=3.6245e-004;
                    h3= 1.9200e-004;
                    h4= 0.0068;
                    h5=0.0024;
                    g=9.81;
                    c1=8.0;
                    mu=30.0;
                    k=24.0;
                    Ts=0.005;

                    den=(h1*h2-pow(h3*cos(u0[2]),2));

                    D[1][1]=h1+h2+2*h3*cos(u0[2]);
                    D[1][2]=h2+h3*cos(u0[2]);
                    D[2][1]=h2+h3*cos(u0[2]);
                    D[2][2]=h2;

                    C[1][1]=-h3*sin(u0[2])*u0[3];
                    C[1][2]=-h3*sin(u0[2])*u0[3]-h3*sin(u0[2])*u0[1];
                    C[2][1]=h3*sin(u0[2])*u0[1];
                    C[2][2]=0;

                    G[1]=h4*g*cos(u0[0])+h5*g*cos(u0[0]+u0[2]);
                    G[2]=h5*g*cos(u0[0]+u0[2]);




                    d11_b=D[1][1]-D[1][2]*D[2][1]/D[2][2];
                    c11_b=C[1][1]-D[1][2]*C[2][1]/D[2][2];
                    c12_b=C[1][2];
                    G_b=G[1]-D[1][2]*G[2]/D[2][2];

                     FB=-(c11_b*u0[1]+c12_b*u0[3]+G_b)/d11_b;
                     b1=1/d11_b;
```

```
F[0]=(h2*h3*sin(u0[2])*pow((u0[1]+u0[3]),2)+pow(h3,2)*cos(u0[2])*sin(u0[2])*pow(u0[1],2)-
h2*h4*g*cos(u0[0])+h3*h5*g*cos(u0[0]+u0[2]))/den;
                        F[1]=(-h3*(h2+h3*cos(u0[2]))*sin(u0[2])*pow((u0[1]+u0[3]),2)-
(h1+h3*cos(u0[2]))*h3*sin(u0[2])*pow(u0[3],2)+(h2+h3*cos(u0[2]))*h4*g*cos(u0[0])-
(h1+h3*cos(u0[2]))*h5*g*cos(u0[0]+u0[2]))/den;
                         B[0]=h2/den;
                         B[1]=-(h2+h3*cos(u0[2]))/den;
                        // Defining the Sliding Surfaces
                         c1=8.0;
                        mu=30.0;
                        k=24.0;
                        s1=c1*(u0[0]-u0[4])+u0[1]-u0[5];  // Defining the sliding Surface of first link

                        if(s1>=0){
                                sgn=1;}
                        else{
                                        sgn=-1;
                        }


                        v1=-mu*sgn-k*s1;

                        Ueq1=-(FB+c1*u0[1]-v1)/b1;
                        tau=Ueq1;
                        taup=tau;
                           if (u0[7]<=0.50){
                        taup=-0.0425;
                       y0[0]=-0.0425;}
                       else{
                       y0[0]=taup;
                       }




/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

/*
 * Updates function
 *
 */
void cfd_slideT1_Update_wrapper(const real_T *u0,
                        const real_T *y0,
                        real_T *xD)
{
/* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO _END */
```

A.3 GENETIC SLIDING MODE BALANCING CONTROLLER

```
/*
 *
 *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 2.0 ---
 *
 *   This file is a wrapper S-function produced by the S-Function
 *   Builder which only recognizes certain fields.  Changes made
 *   outside these fields will be lost the next time the block is
 *   used to load, edit, and resave this file. This file will be overwritten
 *   by the S-function Builder block. If you want to edit this file by hand,
 *   you must change it only in the area defined as:
 *
 *       %%%-SFUNWIZ_wrapper_XXXXX_Changes_BEGIN
 *           Your Changes go here
 *       %%%-SFUNWIZ_wrapper_XXXXXX_Changes_END
 *
 *   For better compatibility with the Real-Time Workshop, the
 *   "wrapper" S-function technique is used.  This is discussed
 *   in the Real-Time Workshop User's Manual in the Chapter titled,
 *   "Wrapper S-functions".
 *
 *   Created: Sat Jan  7 14:31:55 2006
 */


/*
 * Include Files
 *
 */
#include "tmwtypes.h"
/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include "mex.h"
#include <stddef.h>
#include <_syslist.h>
#include <stdlib.h>
#include <math.h>
#include"tmwtypes.h"
#include "matrix.h"
 # define POP_SIZE 10
# define MAX_GENERATION 10
# define NUM_TRAITS 2


/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 4
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
SRC_PATH,INC_PATH,LIB_PATH;


/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */

/*
 * Output functions
 *
 */
void gcslide3_Outputs_wrapper(const real_T *u0,
```

```
                real_T *y0,
                const real_T *xD)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
//*****************************************

                double h1,h2,h3,h4,h5,g,den;
                double Ueq1,Ueq2,s1,s2,S,c1,c2,af,beta,mu,k,stau;
                int sgn;
                  double pi=22.0/7;

                double B[2]={0};
                 double F[2]={0};
                 //*********************************************

                 int DELTA,ELITISM,SELF_ENTERED,popcount;
                int pop_member,current_trait;
                double  MUTAT_PROB,CROSS_PROB, EPSILON;
                 double  HIGHTRAIT[NUM_TRAITS]={30,0.0001};
                 double LOWTRAIT[NUM_TRAITS]={10,0.0009};
                 int SIG_FIGS[NUM_TRAITS]={6,6};
                int  DECIMAL[NUM_TRAITS]={2,-3};
                  int CHROM_LENGTH,make_gene;
                  int TRAIT_START[NUM_TRAITS+1];
                  double trait[NUM_TRAITS][POP_SIZE][MAX_GENERATION];
                 int pop[14][POP_SIZE];
                 int parent_chrom[14][POP_SIZE];
                 int child[14][POP_SIZE];
                 double temp_trait[NUM_TRAITS][POP_SIZE];
                 double  sumfitness;
                  double x2_est,x1_est,x3_est,x4_est,s1_est,s2_est,S_est;
                 double lmda=6.0;
                 double Ts;
                 double tau[POP_SIZE];
                 double fitness[POP_SIZE]={0};

                  double epsl=0.0000001;
                 int  chrom_number;
                 double bestfitness[MAX_GENERATION]={0};  // Initialze the best fitness of each
generation
                 int bestmember;                 // initialize the rank of best pop. member
                 double best_fitness=0;   // Initialze the best fitness of whole generation
                 int xx=2;
                  double tau_best,Kt2_best,Kt1_best;
                  double  pointer,total;
                   int   member_count=0;
                  int gene ,parent_number1,parent_number2,site,rand_gene,site1 ;
                  int place_pointer,place;




                  // NUM_TRAITS=2;          /* Number of traits in each individual*/
                  MUTAT_PROB=0.1;       /* Probability of mutation (typically <.1)*/
                  CROSS_PROB=0.5;       /* Probability of crossover (typically near 1) */
                  SELF_ENTERED=0;       /* "0": a random initial population. */
                                /* "1": a specified initial population */
                                /* If you choose "1", enter it in the program. */
                //  POP_SIZE=1;          /* Number of individuals in the population */
```

```
ELITISM=1;            /* Elitism ON/OFF, 1/0 */
DELTA=100;            /* Number of generations to be counted  */
                      /* for the termination criteria. */
EPSILON = 0.01;       /* Range that the fitness must change */
                      /* in the termination criteria. */
//MAX_GENERATION=99;    /* Number

//************************************************
Ts=0.005;
h1=6.0000e-004;
h2=3.6245e-004;
h3= 1.9200e-004;
h4= 0.0068;
h5=0.0024;
g=9.81;

den=(h1*h2-pow(h3*cos(u0[2]),2));


F[0]=(h2*h3*sin(u0[2])*pow((u0[1]+u0[3]),2)+pow(h3,2)*cos(u0[2])*sin(u0[2])*pow(u0[1],2)-
h2*h4*g*cos(u0[0])+h3*h5*g*cos(u0[0]+u0[2]))/den;
                F[1]=(-h3*(h2+h3*cos(u0[2]))*sin(u0[2])*pow((u0[1]+u0[3]),2)-
(h1+h3*cos(u0[2]))*h3*sin(u0[2])*pow(u0[3],2)+(h2+h3*cos(u0[2]))*h4*g*cos(u0[0])-
(h1+h3*cos(u0[2]))*h5*g*cos(u0[0]+u0[2]))/den;
                 B[0]=h2/den;
                 B[1]=-(h2+h3*cos(u0[2]))/den;
                c1=6.5673;
                c2=8.5511;
                af=0.62;
                beta=0.45;
                mu=0.002;
                k=13.0;
                s1=c1*(u0[0]-pi/2)+u0[1];
                s2=c2*u0[2]+u0[3];
                Ueq1=-(F[0]+c1*u0[1])/B[0];
                Ueq2=-(F[1]+c2*u0[3])/B[1];
                S=af*s1+beta*s2;
                if(S>=0){
                    sgn=1;}
                else{
                            sgn=-1;
                }
                stau=(af*B[0]*Ueq1+beta*B[1]*Ueq2-mu*sgn-k*S)/(af*B[0]+beta*B[1]);

                popcount=0;




                   for (pop_member = 0; pop_member<POP_SIZE; pop_member++)
                   {
                            for (current_trait = 0;current_trait<NUM_TRAITS;current_trait++)
                            {
                              trait[current_trait][pop_member][0] =
((double)rand()/(double)RAND_MAX-0.5)*(HIGHTRAIT[current_trait]-
LOWTRAIT[current_trait])+(1/2)*(HIGHTRAIT[current_trait]+LOWTRAIT[current_trait]);
```

```
                    }
                }

                CHROM_LENGTH=SIG_FIGS[0] +SIG_FIGS[1]+NUM_TRAITS; //Length of the
chromosome is the number of sig. figs. plusthe number of sign positions Initialize

                // Determinarion of starting point of each trait

                TRAIT_START[0]=0;
                for (current_trait=0;current_trait<NUM_TRAITS; current_trait++){


TRAIT_START[current_trait+1]=TRAIT_START[current_trait]+SIG_FIGS[current_trait]+1;

                }



//*************************************************************************
***********

                while (popcount<(MAX_GENERATION-1)){




                    for( pop_member=0; pop_member<POP_SIZE; pop_member++){

                            for (current_trait = 0;current_trait<NUM_TRAITS;current_trait++){


                                    if
(trait[current_trait][pop_member][popcount]>HIGHTRAIT[current_trait]){

                trait[current_trait][pop_member][popcount]=HIGHTRAIT[current_trait];
                                    } // End of if statement

                else if (trait[current_trait][pop_member][popcount]<LOWTRAIT[current_trait]){



                trait[current_trait][pop_member][popcount]=LOWTRAIT[current_trait];

                } // End of els if statement



                // Now that we have reset the traits to be in range, we must
                // convert them to the chromosome form for use with the genetic operators.
                // First, we transfer the sign of the trait into the chromosome

                if (trait[current_trait][pop_member][popcount] < 0){
                        pop[TRAIT_START[current_trait]][pop_member]=0;
                }
                else {
                        pop[TRAIT_START[current_trait]][pop_member]=9;
                }
```

```
                //*********************************************************/

temp_trait[current_trait][pop_member]=fabs(trait[current_trait][pop_member][popcount]);
                                // temp_trait is trait without the sign of trait

                // Next, we store the numbers of the trait in the chromosome:
                // First, set up a temporary trait with at most
                // one nonzero digit to the left of the decimal point.
                // This is used to strip off the numbers to put
                // them into a chromosome.


temp_trait[current_trait][pop_member]=(temp_trait[current_trait][pop_member])/pow(10.0,DECIMAL
[current_trait]-1);


                // Encode the new trait into chromosome form


                for ( make_gene = TRAIT_START[current_trait]+1;make_gene
<TRAIT_START[current_trait+1];make_gene++){

                // For each gene on the trait make the gene the corresponding digit on
                // temp_trait (note that rem(x,y)=x-roundtowardszero(x/y)*y or
                // rem(x,1)=x-roundtowardszero(x) so that rem(x,1) is the fraction part
                // and x-rem(x,1) is the integer part so the next line makes the location in
                // pop the same as the digit to the left of the decimal point of temp_trait

                pop[make_gene][pop_member]=(int)(temp_trait[current_trait][pop_member]-
fmod(temp_trait[current_trait][pop_member],1.0));



                // Next, we take temp_trait and rotate the next digit to the left so that
                // next time around the loop it will pull that digit into the
                // chromosome.  To do this we strip off the leading digit then shift
                // in the next one.

                temp_trait[current_trait][pop_member]=(temp_trait[current_trait][pop_member]-
pop[make_gene][pop_member])*10.0;



                }//end of make gene for


                        }// end of second for
                }    // end of first for



                //*************  Evaluation of Fitness Fuction ******************************

                    sumfitness = 0;             // Re-initialize for each generation
                    bestfitness[popcount]=0;

                //  Evlaute each trait in fittnes function.
```

```
                    for (chrom_number = 0;chrom_number<POP_SIZE;chrom_number++)  {
                    /* fitness[chrom_number]=0;

                   s=lmda*(pi/2-x[0]+x[2])+x[1]+0.75*x[3];
                  tau_bar=-lmda*(x[1]+x[3]);


tau[chrom_number]=(trait[0][chrom_number][popcount]+trait[1][chrom_number][popcount]);

                fitness[chrom_number]=10000000000*exp(-fabs(tau[chrom_number]-
s));//+(0.1/(pow((x[1]-pi/2),2.0)+epsl))+(1/(pow(x[2],2.0)+epsl))+(0.1/(pow(x[3],2.0))+epsl);// +
epsl)+(1/(x[1]-pi/2)^2+ epsl)+(0.1/(x[2]^2+epsl))+(1/(x[3]^2+epsl))+(0.1/(x[4]^2+epsl));
                sumfitness =sumfitness + fitness[chrom_number];*/
                 tau[chrom_number]=(af*B[0]*Ueq1+beta*B[1]*Ueq2-mu*sgn-
trait[0][chrom_number][popcount]*S)/(af*B[0]+beta*B[1]);

                    x2_est=(F[1]+B[1]*tau[chrom_number])*Ts+u0[1];
              x4_est=(F[2]+B[2]*tau[chrom_number])*Ts+u0[3];
              x1_est=x2_est*Ts+u0[0];
              x3_est=x4_est*Ts+u0[2];

              s1_est=c1*(x1_est-pi/2)+x2_est;

              s2_est=c2*x3_est+x4_est;
              S_est=af*s1_est+beta*s2_est;

              fitness[chrom_number]=10000*exp(-fabs(S_est));

                   if((fitness[chrom_number]) > (bestfitness[popcount])){
                    bestmember=chrom_number;
                     bestfitness[popcount]=fitness[chrom_number];



                   }// End of if statemnet

                     } // End of Fittness evalaution Loop


                //*****************************************
                if(bestfitness[popcount]>best_fitness ){
                     tau_best=tau[bestmember];
                   Kt1_best=trait[0][bestmember][popcount];
                   Kt2_best=trait[1][bestmember][popcount];
                    best_fitness=bestfitness[popcount];



                   }// End of if statemnet


                //***************************** Selecting the Parents of Next Generation
                for (pop_member = 0;pop_member<POP_SIZE;pop_member++){


                   if ((ELITISM ==1) && (pop_member==bestmember)){  // If elitism on, and have

                      // the elite member
                   for (gene=0;gene<CHROM_LENGTH;gene++){
```

```
                    parent_chrom[gene][pop_member]=pop[gene][pop_member]; // Makes sure
that
                        // the elite member gets into the next generation.
            } }

                else{


                 pointer=((double)rand()/(double)RAND_MAX)*sumfitness; // This makes the
pointer for the roulette wheel.

                 member_count=0;
                 total=fitness[0];

                 while ((total < pointer) && (pop_member < POP_SIZE)){          // This spins the
wheel to the
                                                                                                //
pointer and finds the
                                                                                                //
chromosome there - which is
                                                                                                //
identified by member_count
                   member_count++;
                   total=total+fitness[member_count];
                 }//end of while

            // Next, make the parent chromosome

            for (gene=0;gene<CHROM_LENGTH;gene++){
             parent_chrom[gene][pop_member]=pop[gene][member_count];


            }// End of for parents selection
            } // end of else

            }//end of member for loop



            //********************************* Cross over
*******************************/

            for (parent_number1 = 0;parent_number1<POP_SIZE;parent_number1++){    //
Crossover (parent_number1 is the individual who gets to mate

                if (ELITISM ==1 && parent_number1==bestmember) {                  // If elitism on
and have the elite member

                        for (gene=0;gene<CHROM_LENGTH;gene++){
                        child[gene][parent_number1]=parent_chrom[gene][parent_number1];

                        }// End of For
                } // End of if

                        else{
                  parent_number2=parent_number1;  // Initialize who the mate is
                  while (parent_number2 == parent_number1){   // Iterate until find % a mate other
than self
```

```
                        parent_number2 =
(int)(((double)rand()/(double)RAND_MAX)*(double)POP_SIZE); // Choose parent number 2
randomly (a random mate)


                    } // End of While


                if ( CROSS_PROB > ((double)rand()/(double)RAND_MAX) ) {        // If true then
crossover occurs

                    site = (int)(((double)rand()/(double)RAND_MAX)*CHROM_LENGTH);  //
Choose site for crossover


                // The next two lines form the child by the swapping of genetic
                // material between the parents
                    for (gene=0;gene<site;gene++){
                            child[gene][parent_number1]=parent_chrom[gene][parent_number1];

                    } // End of for Loop

                for (gene=site;gene<CHROM_LENGTH;gene++){
                    child[gene][parent_number1]=parent_chrom[gene][parent_number2];

                    } // End of for Loop

                } // End of if staement

                else    {                   // No crossover occurs

                // Copy non-crossovered chromosomes into next generation
                // In this case we simply take one parent and make them the child.


                    for (gene=0;gene<CHROM_LENGTH;gene++){
                            child[gene][parent_number1]=parent_chrom[gene][parent_number1];
                    } // End of for Loop



                } // End of else

                    } // End the "if ELITISM..." statement
                    }// End "for parent_number1=..." loop
                //*********************************************************************
                //******************************** Mutate children.    *****************



                // Here, we mutate to a different allele
                // with a probability MUTAT_PROB



                for  (pop_member= 0;pop_member<POP_SIZE;pop_member++){

                    if (ELITISM ==1 && pop_member==bestmember ){  // If elitism on, and have the
elite member
```

```
                    for (gene=0;gene<CHROM_LENGTH;gene++){
            child[gene][pop_member]=child[gene][pop_member];

        } // End of for Loop
      } // end of if


      else {
            for (site1 = 0;site1<CHROM_LENGTH;site1++){

                    if (MUTAT_PROB > ((double)rand()/(double)RAND_MAX)){
// If true then mutate Creat a random gene


      rand_gene=(int)(((double)rand()/(double)RAND_MAX)*10);


            // If it is the same as the one already there then
           // generate another random allele in the alphabet

                while (child[site1][pop_member] == rand_gene){
              rand_gene=(int)(((double)rand()/(double)RAND_MAX)*10);
           }

            child[site1][pop_member]=rand_gene;


            // If takes a value of 10 (which it cannot
            // mutate to) then try again (this is a very low probability
                    //event (most random number generators generate numbers
                    //on the *closed* interval [0,1] and this is why this line
                    // is included).

            if (rand_gene >= 10){
                site1=site1-1;

            }
                }  //End "if MUTAT_PROB > rand ...
                }  // End for site... loop
      } // End "if ELITISM..."
    }  // End for pop_member loop


/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Create the next generation (this completes the main part of the GA)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/
    for(pop_member= 0;pop_member<POP_SIZE;pop_member++){
       for(gene=0;gene<CHROM_LENGTH;gene++){
     pop[gene][pop_member]=child[gene][pop_member];              // Create next
generation (children become parents)




    }
```

```
        }

          popcount++;                    // Increment to the next generation


/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%
        % Next, we have to convert the population (pop) to the base-10
        % representation (called trait) so that we can check if the traits
        % all still lie in the proper ranges specified by HIGHTRAIT and LOWTRAIT
        % at the beginning of the next time around the loop.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%*/

          for (pop_member = 0;pop_member<POP_SIZE;pop_member++){

              for (current_trait = 0; current_trait<NUM_TRAITS;current_trait++){

        trait[current_trait][pop_member][popcount]=0; // Initialize variables
        place_pointer=1;


        /* Change each of the coded traits on the chromosomes into base-10 traits:
        % For each gene on the current_trait past the sign digit but before the
        % next trait find its real number amount and hence after finishing
        % the next loop trait(current_trait,pop_member,popcount) will be the base-10
        % number representing the trait*/

        for
(gene=TRAIT_START[current_trait];gene<TRAIT_START[current_trait+1];gene++){
         place=DECIMAL[current_trait]-place_pointer;

trait[current_trait][pop_member][popcount]=trait[current_trait][pop_member][popcount]+(pop[gene][p
op_member])*pow(10.0,(double)place);
         place_pointer++;

              }//end of for


        /* Determine sign of the traits and fix
        % trait(current_trait,pop_member,popcount) so that it has the right sign:*/


        if (pop[TRAIT_START[current_trait]][pop_member] < 5){
          trait[current_trait][pop_member][popcount]=-
trait[current_trait][pop_member][popcount];

        }//end of if

         } // Ends "for current_trait=..." loop

          }   // Ends "for pop_member=..." loop



//*************************************************************************

        }//end of generation loop
```

```
                    y0[0]=tau_best;



/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

/*
 * Updates function
 *
 */
void gcslide3_Update_wrapper(const real_T *u0,
                      const real_T *y0,
                      real_T *xD)
{
/* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO _END */
```

A.4 TUNING PD GAINS USING GENETIC ALGORITHM

```
/*
 *
 *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 2.0 ---
 *
 *   This file is a wrapper S-function produced by the S-Function
 *   Builder which only recognizes certain fields.  Changes made
 *   outside these fields will be lost the next time the block is
 *   used to load, edit, and resave this file. This file will be overwritten
 *   by the S-function Builder block. If you want to edit this file by hand,
 *   you must change it only in the area defined as:
 *
 *       %%%-SFUNWIZ_wrapper_XXXXX_Changes_BEGIN
 *           Your Changes go here
 *       %%%-SFUNWIZ_wrapper_XXXXXX_Changes_END
 *
 *   For better compatibility with the Real-Time Workshop, the
 *   "wrapper" S-function technique is used.  This is discussed
 *   in the Real-Time Workshop User's Manual in the Chapter titled,
 *   "Wrapper S-functions".
 *
 *   Created: Sat Jan  7 16:28:36 2006
 */


/*
 * Include Files
 *
 */
#include "tmwtypes.h"
/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include "mex.h"
#include <stddef.h>
#include <_syslist.h>
#include <stdlib.h>
#include <math.h>
#include"tmwtypes.h"
#include "matrix.h"
 # define POP_SIZE 10
# define MAX_GENERATION 10
# define NUM_TRAITS 2




/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 8
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
SRC_PATH,INC_PATH,LIB_PATH;


/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */

/*
 * Output functions
 *
```

```
 */
void GFD5_Outputs_wrapper(const real_T *u0,
                  real_T *y0,
                  const real_T *xD)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
//*****************************************

                    double h1,h2,h3,h4,h5,g,den;
                                                  double pi=22.0/7;

                    double B[2]={0};
                     double F[2]={0};
                     //*********************************************

                     int DELTA,ELITISM,SELF_ENTERED,popcount;
                    int pop_member,current_trait;
                    double  MUTAT_PROB,CROSS_PROB, EPSILON;
                      double  HIGHTRAIT[NUM_TRAITS]={450,28};
                 double LOWTRAIT[NUM_TRAITS]={350,20};
                 int SIG_FIGS[NUM_TRAITS]={6,6};
                int  DECIMAL[NUM_TRAITS]={3,2};
                      int CHROM_LENGTH,make_gene;
                      int TRAIT_START[NUM_TRAITS+1];
                      double trait[NUM_TRAITS][POP_SIZE][MAX_GENERATION];
                     int pop[14][POP_SIZE];
                     int parent_chrom[14][POP_SIZE];
                     int child[14][POP_SIZE];
                     double temp_trait[NUM_TRAITS][POP_SIZE];
                     double  sumfitness;

                     double lmda=6.0;
                     double Ts;
                     double tau[POP_SIZE];
                     double fitness[POP_SIZE]={0};

                      double epsl=0.0000001;
                     int  chrom_number;
                     double bestfitness[MAX_GENERATION]={0};  // Initialze the best fitness of each
generation
                     int bestmember;                 // initialize the rank of best pop. member
                     double best_fitness=0;   // Initialze the best fitness of whole generation
                     int xx=2,j;
                      double tau_best,Kt2_best,Kt1_best;
                      double  pointer,total;
                       int   member_count=0;
                      int gene ,parent_number1,parent_number2,site,rand_gene,site1 ;
                      int place_pointer,place;




    double kd,kp,x1,x2,x3,x4;



     double D[2][2]={0};
     double C[2][2]={0};
     double G[2]={0};
```

```
 double  v1,d11_b,c11_b,G_b,c12_b,Sum_S1,S_abs;

//***********************************************
h1=6.0000e-004;
h2=3.6245e-004;
h3= 1.9200e-004;
h4= 0.0068;
h5=0.0024;
g=9.81;
Ts=0.005;

den=(h1*h2-pow(h3*cos(u0[2]),2));
F[0]=(h2*h3*sin(u0[2])*pow((u0[1]+u0[3]),2)+pow(h3,2)*cos(u0[2])*sin(u0[2])*pow(u0[1],2)-
h2*h4*g*cos(u0[0])+h3*h5*g*cos(u0[0]+u0[2]))/den;
F[1]=(-h3*(h2+h3*cos(u0[2]))*sin(u0[2])*pow((u0[1]+u0[3]),2)-
(h1+h3*cos(u0[2]))*h3*sin(u0[2])*pow(u0[3],2)+(h2+h3*cos(u0[2]))*h4*g*cos(u0[0])-
(h1+h3*cos(u0[2]))*h5*g*cos(u0[0]+u0[2]))/den;
 B[0]=h2/den;
 B[1]=-(h2+h3*cos(u0[2]))/den;

D[1][1]=h1+h2+2*h3*cos(u0[2]);
D[1][2]=h2+h3*cos(u0[2]);
D[2][1]=h2+h3*cos(u0[2]);
D[2][2]=h2;

C[1][1]=-h3*sin(u0[2])*u0[3];
C[1][2]=-h3*sin(u0[2])*u0[3]-h3*sin(u0[2])*u0[1];
C[2][1]=h3*sin(u0[2])*u0[1];
C[2][2]=0;

G[1]=h4*g*cos(u0[0])+h5*g*cos(u0[0]+u0[2]);
G[2]=h5*g*cos(u0[0]+u0[2]);



d11_b=D[1][1]-D[1][2]*D[2][1]/D[2][2];
c11_b=C[1][1]-D[1][2]*C[2][1]/D[2][2];
c12_b=C[1][2];
G_b=G[1]-D[1][2]*G[2]/D[2][2];


/*****************************************************************************/

                MUTAT_PROB=0.1;      /* Probability of mutation (typically <.1)*/
                CROSS_PROB=0.5;      /* Probability of crossover (typically near 1) */
                SELF_ENTERED=0;       /* "0": a random initial population. */
                                /* "1": a specified initial population */
                                /* If you choose "1", enter it in the program. */

                ELITISM=1;            /* Elitism ON/OFF, 1/0 */
                DELTA=100;             /* Number of generations to be counted  */
                                /* for the termination criteria. */
                EPSILON = 0.01;               /* Range that the fitness must change */
                                /* in the termination criteria. */


              //***********************************************


                        ;
```

```
popcount=0;


                for (pop_member = 0; pop_member<POP_SIZE; pop_member++)
                {
                        for (current_trait = 0;current_trait<NUM_TRAITS;current_trait++)
                        {
                          trait[current_trait][pop_member][0] =
((double)rand()/(double)RAND_MAX-0.5)*(HIGHTRAIT[current_trait]-
LOWTRAIT[current_trait])+(1/2)*(HIGHTRAIT[current_trait]+LOWTRAIT[current_trait]);



                        }
                }

                CHROM_LENGTH=SIG_FIGS[0] +SIG_FIGS[1]+NUM_TRAITS;
//Length of the chromosome is the number of sig. figs. plusthe number of sign positions Initialize

                // Determinarion of starting point of each trait

        TRAIT_START[0]=0;
        for (current_trait=0;current_trait<NUM_TRAITS; current_trait++){


TRAIT_START[current_trait+1]=TRAIT_START[current_trait]+SIG_FIGS[current_trait]+1;

        }



//*************************************************************************
***********

        while (popcount<(MAX_GENERATION-1)){




                  for( pop_member=0; pop_member<POP_SIZE; pop_member++){

                          for (current_trait =
0;current_trait<NUM_TRAITS;current_trait++){


                                  if
(trait[current_trait][pop_member][popcount]>HIGHTRAIT[current_trait]){

        trait[current_trait][pop_member][popcount]=HIGHTRAIT[current_trait];
                                  } // End of if statement

        else if (trait[current_trait][pop_member][popcount]<LOWTRAIT[current_trait]){
```

```
                    trait[current_trait][pop_member][popcount]=LOWTRAIT[current_trait];

                    } // End of els if statement



                    // Now that we have reset the traits to be in range, we must
                    // convert them to the chromosome form for use with the genetic operators.
                    // First, we transfer the sign of the trait into the chromosome

                    if (trait[current_trait][pop_member][popcount] < 0){
                              pop[TRAIT_START[current_trait]][pop_member]=0;
                    }
                    else {
                             pop[TRAIT_START[current_trait]][pop_member]=9;
                    }



                    //*********************************************************/

temp_trait[current_trait][pop_member]=fabs(trait[current_trait][pop_member][popcount]);
                                        // temp_trait is trait without the sign of trait

                    // Next, we store the numbers of the trait in the chromosome:
                    // First, set up a temporary trait with at most
                    // one nonzero digit to the left of the decimal point.
                    // This is used to strip off the numbers to put
                    // them into a chromosome.


temp_trait[current_trait][pop_member]=(temp_trait[current_trait][pop_member])/pow(10.0,DECIMAL
[current_trait]-1);


                    // Encode the new trait into chromosome form


                    for ( make_gene = TRAIT_START[current_trait]+1;make_gene
<TRAIT_START[current_trait+1];make_gene++){

                    // For each gene on the trait make the gene the corresponding digit on
                    // temp_trait (note that rem(x,y)=x-roundtowardszero(x/y)*y or
                    // rem(x,1)=x-roundtowardszero(x) so that rem(x,1) is the fraction part
                    // and x-rem(x,1) is the integer part so the next line makes the location in
                    // pop the same as the digit to the left of the decimal point of temp_trait

                    pop[make_gene][pop_member]=(int)(temp_trait[current_trait][pop_member]-
fmod(temp_trait[current_trait][pop_member],1.0));



                    // Next, we take temp_trait and rotate the next digit to the left so that
                    // next time around the loop it will pull that digit into the
                    // chromosome.  To do this we strip off the leading digit then shift
                    // in the next one.
```

```
                    temp_trait[current_trait][pop_member]=(temp_trait[current_trait][pop_member]-
pop[make_gene][pop_member])*10.0;



                    }//end of make gene for


                                }// end of second for
                            }    // end of first for



                    //*************  Evaluation of Fitness Fuction
*****************************

                                sumfitness = 0;             // Re-initialize for each generation
                            bestfitness[popcount]=0;

                    //  Evlaute each trait in fittnes function.

                                for (chrom_number = 0;chrom_number<POP_SIZE;chrom_number++)  {
                            kp=trait[0][chrom_number][popcount];
     kd=trait[1][chrom_number][popcount];

     v1=kd*(0- u0[1])+kp*(pi/2-u0[0]);
     tau[chrom_number]=d11_b*v1+c11_b*u0[1]+c12_b*u0[3]+G_b;
     Sum_S1=0;
     x1=u0[0];
     x2=u0[1];
     x3=u0[2];
     x4=u0[3];
     for (j=1;j<=10;j++){
     S_abs=fabs(x1-pi/2)+fabs(x2)+fabs(x3)+fabs(x4);
     Sum_S1=S_abs+Sum_S1;
     x2=(F[1]+B[1]*tau[chrom_number])*Ts+x2;
     x4=(F[2]+B[2]*tau[chrom_number])*Ts+x4;
     x1=x2*Ts+x1;
     x3=x4*Ts+x3;

     F[0]=(h2*h3*sin(x3)*pow((x2+x4),2)+pow(h3,2)*cos(x3)*sin(x3)*pow(x2,2)-
h2*h4*g*cos(x1)+h3*h5*g*cos(x1+x3))/den;
     F[1]=(-h3*(h2+h3*cos(x3))*sin(x3)*pow((x2+x4),2)-
(h1+h3*cos(x3))*h3*sin(x3)*pow(x4,2)+(h2+h3*cos(x3))*h4*g*cos(x1)-
(h1+h3*cos(x3))*h5*g*cos(u0[0]+x3))/den;
      B[0]=h2/den;
      B[1]=-(h2+h3*cos(x3))/den;
      den=(h1*h2-pow(h3*cos(x3),2));

     }

     fitness[chrom_number]=exp(-fabs(Sum_S1));

                                if((fitness[chrom_number]) > (bestfitness[popcount])){
                            bestmember=chrom_number;
                                bestfitness[popcount]=fitness[chrom_number];



                            }// End of if statemnet
```

```
                    } // End of Fittness evalaution Loop


                //*****************************************
                if(bestfitness[popcount]>best_fitness ){
                     tau_best=tau[bestmember];
                   Kt1_best=trait[0][bestmember][popcount];
                   Kt2_best=trait[1][bestmember][popcount];
                    best_fitness=bestfitness[popcount];



                    }// End of if statemnet


                //***************************** Selecting the Parents of Next Generation
                for (pop_member = 0;pop_member<POP_SIZE;pop_member++){


                    if ((ELITISM ==1) && (pop_member==bestmember)){  // If elitism on, and
have

                       // the elite member
                    for (gene=0;gene<CHROM_LENGTH;gene++){

                            parent_chrom[gene][pop_member]=pop[gene][pop_member]; //
Makes sure that
                                // the elite member gets into the next generation.
                } }

                    else{


                    pointer=((double)rand()/(double)RAND_MAX)*sumfitness; // This makes the
pointer for the roulette wheel.

                    member_count=0;
                    total=fitness[0];

                    while ((total < pointer) && (pop_member < POP_SIZE)){          // This spins
the wheel to the
                                                                                                //
pointer and finds the
                                                                                                //
chromosome there - which is
                                                                                                //
identified by member_count
                      member_count++;
                      total=total+fitness[member_count];
                    }//end of while

                // Next, make the parent chromosome

                for (gene=0;gene<CHROM_LENGTH;gene++){
                 parent_chrom[gene][pop_member]=pop[gene][member_count];


                }// End of for parents selection
                } // end of else
```

```
}//end of member for loop


//******************************** Cross over
*********************************/

for (parent_number1 = 0;parent_number1<POP_SIZE;parent_number1++){    //
Crossover (parent_number1 is the individual who gets to mate

if (ELITISM ==1 && parent_number1==bestmember) {                    // If
elitism on and have the elite member

for (gene=0;gene<CHROM_LENGTH;gene++){

child[gene][parent_number1]=parent_chrom[gene][parent_number1];

}// End of For
} // End of if

else{
parent_number2=parent_number1;  // Initialize who the mate is
while (parent_number2 == parent_number1){   // Iterate until find % a
mate other than self


parent_number2 =
(int)(((double)rand()/(double)RAND_MAX)*(double)POP_SIZE); // Choose parent number 2
randomly (a random mate)


} // End of While


if ( CROSS_PROB > ((double)rand()/(double)RAND_MAX) ) {         // If
true then crossover occurs

site = (int)(((double)rand()/(double)RAND_MAX)*CHROM_LENGTH);  //
Choose site for crossover


// The next two lines form the child by the swapping of genetic
// material between the parents
for (gene=0;gene<site;gene++){

child[gene][parent_number1]=parent_chrom[gene][parent_number1];

} // End of for Loop

for (gene=site;gene<CHROM_LENGTH;gene++){
child[gene][parent_number1]=parent_chrom[gene][parent_number2];

} // End of for Loop

} // End of if staement

else     {                  // No crossover occurs

// Copy non-crossovered chromosomes into next generation
```

```
                    // In this case we simply take one parent and make them the child.


                              for (gene=0;gene<CHROM_LENGTH;gene++){

          child[gene][parent_number1]=parent_chrom[gene][parent_number1];
                                } // End of for Loop



                    } // End of else

                              } // End the "if ELITISM..." statement
                              }// End "for parent_number1=..." loop

//*********************************************************************
                    //******************************** Mutate children.    ******************



                    // Here, we mutate to a different allele
                    // with a probability MUTAT_PROB



                    for  (pop_member= 0;pop_member<POP_SIZE;pop_member++){

                              if (ELITISM ==1 && pop_member==bestmember ){  // If elitism on, and
have the elite member


                                        for (gene=0;gene<CHROM_LENGTH;gene++){
                                   child[gene][pop_member]=child[gene][pop_member];

                                } // End of for Loop
                              } // end of if


                              else {
                                        for (site1 = 0;site1<CHROM_LENGTH;site1++){

                                             if (MUTAT_PROB >
((double)rand()/(double)RAND_MAX)){       // If true then mutate Creat a random gene


          rand_gene=(int)(((double)rand()/(double)RAND_MAX)*10);


                               // If it is the same as the one already there then
                              // generate another random allele in the alphabet

                                        while (child[site1][pop_member] == rand_gene){
                                   rand_gene=(int)(((double)rand()/(double)RAND_MAX)*10);
                              }

                               child[site1][pop_member]=rand_gene;


                               // If takes a value of 10 (which it cannot
                               // mutate to) then try again (this is a very low probability
```

```
                                        //event (most random number generators generate
numbers
                                        //on the *closed* interval [0,1] and this is why this line
                                        // is included).

                    if (rand_gene >= 10){
                        site1=site1-1;

                    }
                            }  //End "if MUTAT_PROB > rand ...
                            }  // End for site... loop
                     } // End "if ELITISM..."
               }  // End for pop_member loop


/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
               % Create the next generation (this completes the main part of the GA)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/
               for(pop_member= 0;pop_member<POP_SIZE;pop_member++){
                      for(gene=0;gene<CHROM_LENGTH;gene++){
                 pop[gene][pop_member]=child[gene][pop_member];           // Create next
generation (children become parents)




               }
               }

                 popcount++;                 // Increment to the next generation


/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%
               % Next, we have to convert the population (pop) to the base-10
               % representation (called trait) so that we can check if the traits
               % all still lie in the proper ranges specified by HIGHTRAIT and LOWTRAIT
               % at the beginning of the next time around the loop.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%*/

                 for (pop_member = 0;pop_member<POP_SIZE;pop_member++){

                        for (current_trait = 0; current_trait<NUM_TRAITS;current_trait++){

               trait[current_trait][pop_member][popcount]=0; // Initialize variables
               place_pointer=1;


               /* Change each of the coded traits on the chromosomes into base-10 traits:
               % For each gene on the current_trait past the sign digit but before the
               % next trait find its real number amount and hence after finishing
               % the next loop trait(current_trait,pop_member,popcount) will be the base-10
               % number representing the trait*/
```

```
                    for
(gene=TRAIT_START[current_trait];gene<TRAIT_START[current_trait+1];gene++){
                     place=DECIMAL[current_trait]-place_pointer;

trait[current_trait][pop_member][popcount]=trait[current_trait][pop_member][popcount]+(pop[gene][p
op_member])*pow(10.0,(double)place);
                     place_pointer++;

                              }//end of for


                    /* Determine sign of the traits and fix
                    % trait(current_trait,pop_member,popcount) so that it has the right sign:*/


                    if (pop[TRAIT_START[current_trait]][pop_member] < 5){
                      trait[current_trait][pop_member][popcount]=-
trait[current_trait][pop_member][popcount];

                    }//end of if

                     } // Ends "for current_trait=..." loop

                       }   // Ends "for pop_member=..." loop



//*********************************************************************

                    }//end of generation loop


                     if (u0[7]<=0.50){

                      y0[0]=-0.0425;}
                      else{
                      y0[0]=tau_best;
                      }

/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

/*
 * Updates function
 *
 */
void GFD5_Update_wrapper(const real_T *u0,
                    const real_T *y0,
                    real_T *xD)
{
/* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO _END */


xD[0] = u0[0]*u0[1]*0.05;

/* %%%-SFUNWIZ_wrapper_Update_Changes_END --- EDIT HERE TO _BEGIN */
}

/*
```

```
 *  Derivatives function
 *
 */
void GFD5_Derivatives_wrapper(const real_T *u0,
                      const real_T *y0,
                      real_T *dx )
{
/* %%%-SFUNWIZ_wrapper_Derivatives_Changes_BEGIN --- EDIT HERE TO _END */

/* %%%-SFUNWIZ_wrapper_Derivatives_Changes_END --- EDIT HERE TO _BEGIN */
}
```

APPENDIX B

S_FUNCTIONS IN M-FILE

B.1 SLIDING MODE SWING UP CONTROLLER

```
%*******************Control of   2 Link Underactuaed Manipulator   **************%
%*****  S_function  in  m File for  Sliding Mode Swing up Controller **************%
%Written By : Joudeh Yasin              Advised By: Dr. Mohd Aljarrah
%                                      Date:14.004.05
%*************************************************************% %
%

function [sys,x0,str,ts]=slideFD3(t,x,u,flag,h1,h2,h3,h4,h5)



switch flag,

  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0,
    [sys,x0,str,ts] = mdlInitializeSizes(h1,h2,h3,h4,h5);

  %%%%%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%%%%
  case 2,
    sys = mdlUpdate(t,x,u);

  %%%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%%
  case 3
    sys = mdlOutputs(t,x,u,h1,h2,h3,h4,h5);

  %%%%%%%%%%%%
  % Terminate %
  %%%%%%%%%%%%
case 9,
    sys = [];

  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end


%end simom2

%
%=============================================================================
=====
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
=====
%
function [sys,x0,str,ts] = mdlInitializeSizes(h1,h2,h3,h4,h5)

sizes = simsizes;

sizes.NumContStates  = 0;
sizes.NumDiscStates  = 9;
sizes.NumOutputs     =1;
```

```
sizes.NumInputs      = 9;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);


x0  = 0;
str = [];
ts  = [-1 0];  % inherited sample time

%
function sys = mdlUpdate(t,x,u)

sys = u;

%end mdlUpdate

%
%=============================================================================
% mdlOutputs
% Return the output vector for the S-function
%=============================================================================
%
function sys = mdlOutputs(t,x,u,h1,h2,h3,h4,h5)

% Evaluation of Dynamic Equation Parameters
g=9.81;
D=[(h1+h2+2*h3*cos(x(3)))  (h2+h3*cos(x(3))); (h2+h3*cos(x(3))) h2];
C=[(-h3*sin(x(3))*x(4)) (-h3*sin(x(3))*x(4)-h3*sin(x(3))*x(2));h3*sin(x(3))*x(2) 0];
G=[(h4*g*cos(x(1))+h5*g*cos(x(1)+x(3)));h5*g*cos(x(1)+x(3))];

d11_b=D(1,1)-D(1,2)*D(2,1)/D(2,2);
c11_b=C(1,1)-D(1,2)*C(2,1)/D(2,2);
c12_b=C(1,2);
G_b=G(1)-D(1,2)*G(2,1)/D(2,2);

 F=-(c11_b*x(2)+c12_b*x(4)+G_b)/d11_b;
 b1=1/d11_b;

% Defining the Sliding Surfaces
c1=8;
mu=30;
k=24;
Ts=0.005;
e=x(1)-x(5);
e_dot=x(2)-x(6);

s1=c1*(x(1)-x(5))+x(2)-x(6);  % Defining the first sliding Surface of first link
v1=-mu*sign(s1)-k*s1;
Ueq1=-(F+c1*x(2)-v1)/b1;
% Ueq1=-(F-v1)/b1;
tau=Ueq1;

if (x(8)<=0.5)
   sys=-0.0435;
else
sys =tau;

end                 %end mdlOutputs
```

B.2 SLIDING MODE BALANCING CONTROLLER

```
%*******************Contro of   2 Link Underactuaed Manipulator **************%
%********* S_function  m File for Sliding Mode  Balancing Controller ************%
%Written By : Joudeh Yasin              Advised By: Dr. Mohd Aljarrah
%                                      Date:14.004.05
%************************************************************% %


function [sys,x0,str,ts]=gatest30(t,x,u,flag,h1,h2,h3,h4,h5)



switch flag,

  %%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%
  case 0,
    [sys,x0,str,ts] = mdlInitializeSizes(h1,h2,h3,h4,h5);

  %%%%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%%%
  case 2,
    sys = mdlUpdate(t,x,u);

  %%%%%%%%%%
  % Outputs %
  %%%%%%%%%%
  case 3
    sys = mdlOutputs(t,x,u,h1,h2,h3,h4,h5);

  %%%%%%%%%%%
  % Terminate %
  %%%%%%%%%%%
case 9,
    sys = [];

  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end


%end simom2

%
%=============================================================================
=====
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
=====
%
function [sys,x0,str,ts] = mdlInitializeSizes(h1,h2,h3,h4,h5)

sizes = simsizes;

sizes.NumContStates  = 0;
sizes.NumDiscStates  = 4;
sizes.NumOutputs     =1;
```

```
sizes.NumInputs      = 4;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);

x0  = 0;
str = [];
ts  = [-1 0];  % inherited sample time

%
function sys = mdlUpdate(t,x,u)

sys = u;

%end mdlUpdate

%=============================================================================
% mdlOutputs
% Return the output vector for the S-function
%=============================================================================
%
function sys = mdlOutputs(t,x,u,h1,h2,h3,h4,h5)

% Evaluation of Dynamic Equation Parameters
g=9.81;
D=[(h1+h2+2*h3*cos(x(3)))  (h2+h3*cos(x(3))); (h2+h3*cos(x(3))) h2];
C=[(-h3*sin(x(3))*x(4)) (-h3*sin(x(3))*x(4)-h3*sin(x(3))*x(2));h3*sin(x(3))*x(2) 0];
G=[(h4*g*cos(x(1))+h5*g*cos(x(1)+x(3)));h5*g*cos(x(1)+x(3))];
F=-inv(D)*(C*[x(2) x(4)]'+G);
B=inv(D)*[1 0]';

% Defining the Sliding Surfaces
c1=6.327;
c2=9;
af0=0.62;
beta=0.458;
mu=0.002;
k=13;
s1=c1*(x(1)-pi/2)+x(2);  % Defining the first sliding Surface of first link
%  s1=c1*(pi/2-x(1))+x(2);  % Defining the first sliding Surface of first link
s2=c2*x(3)+x(4);         % Defining the second sliding Surface of second link


% if statement for asymmptotical stability condition
% if (s1*s2)>0
%     af=af0;
% else
%     af=-af0;
% end
af=af0;
Ueq1=-(F(1)+c1*x(2))/B(1);
Ueq2=-(F(2)+c2*x(4))/B(2);
S=af*s1+beta*s2;
% Defining the output Torque
% tau=k*S;
tau=(af*B(1)*Ueq1+beta*B(2)*Ueq2-mu*sign(S)-k*S)/(af*B(1)+beta*B(2));
sys =tau;
%end mdlOutputs
```

B.3 GENETIC SLIDING MODE BALANCING CONTROLLER

```
%*******************Contro of   2 Link Underactuaed Manipulator **************%
%******************* S_function  m Filefor Genetic Sliding Mode Algorithm  **************%
%Written By : Joudeh Yasin              Advised By: Dr. Mohd Aljarrah
%                                      Date:14.004.05
%*************************************************************% %
% This program is a _Functiom m file that develops genetic sliding mode
% Algorithm For Controlling Two Link underactuated Manipultor
% It is reprented in Matlab Simulink as block and take inputs andprduce
% output from Matlab Simulink.

% The Skelton og Genetic Algorithm is used from

%   Kevin Passino
%   Version: 7/21/98, latest change 10/22/03 with correction from
%    Phan Tran Ho Truc, Mr., B. Eng., lecturer in HCM City University of Technology
%    to lines 379-384 of old program (lines 380-381 of this program)
%
% Notes: This program has evolved (hopefully to achieve a higher fitness!)
% over time using programming ideas from several persons including
% LaMoyne Porter, Will Lennon, Jonathan Cook, and Jim Gremling.


function [sys,x0,str,ts]=gaslide(t,x,u,flag,h1,h2,h3,h4,h5)



switch flag,

  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0,
    [sys,x0,str,ts] = mdlInitializeSizes(h1,h2,h3,h4,h5);

  %%%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%%
  case 2,
    sys = mdlUpdate(t,x,u);

  %%%%%%%%%%
  % Outputs %
  %%%%%%%%%%
  case 3
    sys = mdlOutputs(t,x,u,h1,h2,h3,h4,h5);

  %%%%%%%%%%%%
  % Terminate %
  %%%%%%%%%%%%
case 9,
    sys = [];

  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end


%end simom2
```

```
%
%=============================================================
================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================
================
%
function [sys,x0,str,ts] = mdlInitializeSizes(h1,h2,h3,h4,h5)

sizes = simsizes;

sizes.NumContStates  = 0;
sizes.NumDiscStates  = 6;
sizes.NumOutputs     =3;
sizes.NumInputs      = 6;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);


x0  = 0;
str = [];
ts  = [-1 0];  % inherited sample time

%
function sys = mdlUpdate(t,x,u)

sys = u;

%end mdlUpdate

%
%=============================================================
===========
% mdlOutputs
% Return the output vector for the S-function
%=============================================================
===========
%
function sys = mdlOutputs(t,x,u,h1,h2,h3,h4,h5)

best_fittness=-10000;
g=9.81;
D=[(h1+h2+2*h3*cos(x(3)))  (h2+h3*cos(x(3))); (h2+h3*cos(x(3))) h2];
C=[(-h3*sin(x(3))*x(4)) (-h3*sin(x(3))*x(4)-h3*sin(x(3))*x(2));h3*sin(x(3))*x(2) 0];
G=[(h4*g*cos(x(1))+h5*g*cos(x(1)+x(3)));h5*g*cos(x(1)+x(3))];
F=-inv(D)*(C*[x(2) x(4)]'+G);
B=inv(D)*[1 0]';

% Defining the Sliding Surfaces
c1=6.327;
c2=9;
af0=0.62;
beta=0.458;
mu=0.002;


Ts=0.005;
```

```
s1=c1*(x(1)-pi/2)+x(2);  % Defining the first sliding Surface of first link

s2=c2*x(3)+x(4);         % Defining the second sliding Surface of second link
af=af0;
Ueq1=-(F(1)+c1*x(2))/B(1);
Ueq2=-(F(2)+c2*x(4))/B(2);
S=af*s1+beta*s2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% User: Input your own parameters below (the ones given are for the
%       optimization proble   m given above):
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  NUM_TRAITS=2;          % Number of traits in each individual
  HIGHTRAIT=[30 0.00999];          % Upper limit of a trait
   LOWTRAIT=[10 0.000];                % Lower limit of a trait
  SIG_FIGS=[6 6]';       % Number of genes in each trait
  DECIMAL=[2 -2];         % Order of magnitude the trait
  MUTAT_PROB=0.05;       % Probability of mutation (typically <.1)
  CROSS_PROB=0.5;        % Probability of crossover (typically near 1)
  SELF_ENTERED=0;        % "0": a random initial population.
                  % "1": a specified initial population
                  % If you choose "1", enter it in the program.
  POP_SIZE=30;           % Number of individuals in the population
  ELITISM=1;             % Elitism ON/OFF, 1/0
  DELTA=100;             % Number of generations to be counted
                  % for the termination criteria.
  EPSILON = 0.01;                % Range that the fitness must change
                  % in the termination criteria.
  MAX_GENERATION=50 ;  % Number of times the loop will run before

rand('state',0) % Reset the random number generator so each time you re-run
            % the program with no changes you will get the same results.



popcount=1;                  % Initialize the generation count,
                                                    % set it to one, the first population

if SELF_ENTERED == 0          % Make a random initial population for
                                                                    % base-10 operation by
specifying the
                                                                    % matrix of initial traits

 trait(1,1,popcount)= x(5);  % Inserting  the best traits of previous generation
  trait(2,1,popcount)= x(6);  % Inserting  the best traits of previous generation
for pop_member = 2:POP_SIZE
    for current_trait = 1:NUM_TRAITS,
                    trait(current_trait,pop_member,popcount)=...
                    (rand-(1/2))*(HIGHTRAIT(current_trait)-LOWTRAIT(current_trait))+...
                    (1/2)*(HIGHTRAIT(current_trait)+LOWTRAIT(current_trait));

                                % This starts the population off with numbers chosen randomly
                                        % on the allowed range of variation, for this example.
            end
end

else
```

```
for pop_member = 1:POP_SIZE
       for current_trait = 1:NUM_TRAITS,
           trait(current_trait,pop_member,popcount)=0;
                                   % To start with a guess where all the population members are zero

                        end
end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Next, we need to set some values based on the values input by
% the user.  In particular, we determine the length of the
% chromosome and start point of each trait.  This information
% will be used later in the main algorithm.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   CHROM_LENGTH=sum(SIG_FIGS)+NUM_TRAITS; % Length of the chromosome is
                                                                                   % the
number of sig. figs. plus
                                                                                   % the
number of sign positions
   TRAIT_START(1)=1;                                     % Initialize: the first trait
                                                                                   % starts at
the first digit
                                                                                   % (this is
the sign digit)

   for current_trait=1:NUM_TRAITS,           % Determine the start point of the
                                                                        % other traits - it is
the start of
                                                                        % the last trait plus
the no. of sig.
                                                                        % figs. plus one for
sign
TRAIT_START(current_trait+1)=...
   TRAIT_START(current_trait)+SIG_FIGS(current_trait)+1;
   % Yes, we compute the TRAIT_START for one extra trait - this
   % is used for convenience in the code below.
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% In this next loop the fitness is calculated, the children are
% created and it repeats until the EPSILON-DELTA termination condition
% is satisfied or MAX_GENERATION is reached.  This is the main
% loop of the GA.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while popcount <= MAX_GENERATION

% First, fix bad traits (i.e., ones that are out of the range
% specified by HIGHTRAIT and LOWTRAIT) by saturation at the extremes

for pop_member = 1:POP_SIZE
```

```
for current_trait = 1:NUM_TRAITS,


if trait(current_trait,pop_member,popcount)>HIGHTRAIT(current_trait)

                % The trait has went higher than the upper
                % bound so let the trait equal to the
                % HIGHTRAIT bound.

trait(current_trait,pop_member,popcount)=HIGHTRAIT(current_trait);

% Now consider the other case:

elseif trait(current_trait,pop_member,popcount)<LOWTRAIT(current_trait)

                % The trait has went lower than the lower
                % bound so let the trait equal to the
                % LOWTRAIT bound

trait(current_trait,pop_member,popcount)=LOWTRAIT(current_trait);

                        end

% Now that we have reset the traits to be in range, we must
% convert them to the chromosome form for use with the genetic operators.
% First, we transfer the sign of the trait into the chromosome

            if trait(current_trait,pop_member,popcount) < 0
              pop(TRAIT_START(current_trait),pop_member)=0;
            else
              pop(TRAIT_START(current_trait),pop_member)=9;
            end

% Next, strip off the sign and store the resulting value in a
% temporary variable that is used in the construction of pop

temp_trait(current_trait,pop_member)=...
        abs(trait(current_trait,pop_member,popcount));
                        % temp_trait is trait without the sign of trait

% Next, we store the numbers of the trait in the chromosome:
% First, set up a temporary trait with at most
% one nonzero digit to the left of the decimal point.
% This is used to strip off the numbers to put
% them into a chromosome.

temp_trait(current_trait,pop_member)=...
temp_trait(current_trait,pop_member)/10^(DECIMAL(current_trait)-1);

% Encode the new trait into chromosome form

for make_gene = TRAIT_START(current_trait)+1:TRAIT_START(current_trait+1)-1,

% For each gene on the trait make the gene the corresponding digit on
% temp_trait (note that rem(x,y)=x-roundtowardszero(x/y)*y or
% rem(x,1)=x-roundtowardszero(x) so that rem(x,1) is the fraction part
% and x-rem(x,1) is the integer part so the next line makes the location in
% pop the same as the digit to the left of the decimal point of temp_trait

pop(make_gene,pop_member)=temp_trait(current_trait,pop_member)-...
```

```
  rem(temp_trait(current_trait,pop_member),1);

% Next, we take temp_trait and rotate the next digit to the left so that
% next time around the loop it will pull that digit into the
% chromosome.  To do this we strip off the leading digit then shift
% in the next one.

temp_trait(current_trait,pop_member)=...
  (temp_trait(current_trait,pop_member)-pop(make_gene,pop_member))*10;

end

       end % Ends "for current_trait=..." loop

            end    % Ends "for pop_member=..." loop


sumfitness = 0;                % Re-initialize for each generation

% First, determine the values of the function to be minimized for
% each chromosome.

  for chrom_number = 1:POP_SIZE,       % Test fitness
% k=trait(1,chrom_number,popcount);
% c1=trait(1,chrom_number,popcount);
% c2=trait(2,chrom_number,popcount);
k=13;
 mu=0.002;
%  mu=trait(2,chrom_number,popcount);
tau(chrom_number)=(af*B(1)*Ueq1+beta*B(2)*Ueq2-mu*sign(S)-k*S)/(af*B(1)+beta*B(2));

x2_est=(F(1)+B(1)*tau(chrom_number))*Ts+x(2);
x4_est=(F(2)+B(2)*tau(chrom_number))*Ts+x(4);
x1_est=x2_est*Ts+x(1);
x3_est=x4_est*Ts+x(3);

s1_est=c1*(x1_est-pi/2)+x2_est;

s2_est=c2*x3_est+x4_est;
S_est=af*s1_est+beta*s2_est;


 fitness(chrom_number)=1000*exp(-abs(x1_est-pi/2))+1000*exp(-abs(x3_est));


sumfitness =sumfitness + fitness(chrom_number);

end

% Next, determine the most fit and least fit chromosome and
% the chrom_numbers (which we call bestmember and worstmember).

[bestfitness(popcount),bestmember]=max(fitness);
[worstfitness(popcount),worstmember]=min(fitness);

% Next, save these (if want to save worstindividual can too)
if (bestfitness(popcount)>=best_fittness)
   tau_best=tau(bestmember);
   Kt1_best=trait(1,bestmember,popcount);
   Kt2_best=trait(2,bestmember,popcount);
```

```
   best_fittness=bestfitness(popcount);

end

bestindividual(:,popcount)=trait(:,bestmember,popcount);
%worstindividual(:,popcount)=trait(:,worstmember,popcount);

% Compute the average fitness in case you want to plot it.

avefitness(popcount) = sumfitness / POP_SIZE;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create the next generation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% First, form the mating pool.
% To do this we select as parents the
% chromosomes that are most fit.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  for pop_member = 1:POP_SIZE,
          if ELITISM ==1 & pop_member==bestmember  % If elitism on, and have
                                                                                  %
the elite member
                  parent_chrom(:,pop_member)=pop(:,pop_member); % Makes sure that
                                  % the elite member gets into the next
                                                                  % generation.
          else
     pointer=rand*sumfitness; % This makes the pointer for the roulette
                      % wheel.
     member_count=1;       % Initialization
     total=fitness(1);

     while total < pointer,              % This spins the wheel to the
                                                                  % pointer and finds
the
                                                                  % chromosome
there - which is
                                                                  % identified by
member_count
        member_count=member_count+1;
        total=total+fitness(member_count);
     end

% Next, make the parent chromosome

    parent_chrom(:,pop_member)=pop(:,member_count);
          end
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
% Reproduce section (i.e., make off-spring - "children")
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
% In the approach below each individual gets to mate and
```

```
% they randomly pick someone else (not themselves) in the
% mating pool to mate with.  Resulting children are
% composed of a combination of genetic material of their
% parents.  If elitism is on, when the elite member gets
% a chance to mate they do not; they are simply copied
% over to the next generation.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%

for parent_number1 = 1:POP_SIZE,    % Crossover (parent_number1 is the
                                                                    % individual who gets
to mate
          if ELITISM ==1 & parent_number1==bestmember % If elitism on, and
                                                                                        %
have the elite member
                    child(:,parent_number1)=parent_chrom(:,parent_number1);
          else
            parent_number2=parent_number1;  % Initialize who the mate is
            while parent_number2 == parent_number1   % Iterate until find
                                                   % a mate other than

          % yourself
      parent_number2 = rand*POP_SIZE; % Choose parent number 2
                                                                              % randomly
(a random mate)
      parent_number2 = parent_number2-rem(parent_number2,1)+1;
            end
   if CROSS_PROB > rand          % If true then crossover occurs

      site = rand*CHROM_LENGTH;   % Choose site for crossover
      site = site-rem(site,1)+1;  % and make it a valid integer
                                                        % number for a site

% The next two lines form the child by the swapping of genetic
% material between the parents

child(1:site,parent_number1)=parent_chrom(1:site,parent_number1);

child(site+1:CHROM_LENGTH,parent_number1)=...
    parent_chrom(site+1:CHROM_LENGTH,parent_number2);

     else                        % No crossover occurs

% Copy non-crossovered chromosomes into next generation
% In this case we simply take one parent and make them
% the child.

child(:,parent_number1)=parent_chrom(:,parent_number1);

            end
   end  % End the "if ELITISM..." statement
end  % End "for parent_number1=..." loop

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
% Mutate children.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
% Here, we mutate to a different allele
% with a probability MUTAT_PROB
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  for pop_member= 1:POP_SIZE,

          if ELITISM ==1 & pop_member==bestmember  % If elitism on, and
                                                                                    %
have the elite member
                    child(:,pop_member)=child(:,pop_member); % Do not mutate
                                                                                    %
the elite member
          else
           for site = 1:CHROM_LENGTH,

      if MUTAT_PROB > rand        % If true then mutate
        rand_gene=rand*10;                        % Creat a random gene

        % If it is the same as the one already there then
        % generate another random allele in the alphabet

        while child(site,pop_member) == rand_gene-rem(rand_gene,1),
          rand_gene=rand*10;
        end;

        % If it is not the same one, then mutate

        child(site,pop_member)=rand_gene-rem(rand_gene,1);

        % If takes a value of 10 (which it cannot
        % mutate to) then try again (this is a very low probability
                              % event (most random number generators generate numbers
                              % on the *closed* interval [0,1] and this is why this line
                              % is included).

        if rand_gene == 10
           site=site-1;
        end
      end  % End "if MUTAT_PROB > rand ...
            end  % End for site... loop
          end  % End "if ELITISM..."
  end  % End for pop_member loop

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create the next generation (this completes the main part of the GA)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  pop=child;              % Create next generation (children
                                                         % become parents)

  popcount=popcount+1;               % Increment to the next generation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%
% Next, we have to convert the population (pop) to the base-10
% representation (called trait) so that we can check if the traits
% all still lie in the proper ranges specified by HIGHTRAIT and LOWTRAIT
% at the beginning of the next time around the loop.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
```

```
for pop_member = 1:POP_SIZE

        for current_trait = 1:NUM_TRAITS,

trait(current_trait,pop_member,popcount)=0; % Initialize variables
place_pointer=1;

% Change each of the coded traits on the chromosomes into base-10 traits:
% For each gene on the current_trait past the sign digit but before the
% next trait find its real number amount and hence after finishing
% the next loop trait(current_trait,pop_member,popcount) will be the base-10
% number representing the trait

for gene=TRAIT_START(current_trait)+1:TRAIT_START(current_trait+1)-1,
 place=DECIMAL(current_trait)-place_pointer;
 trait(current_trait,pop_member,popcount)=...
 trait(current_trait,pop_member,popcount)+...
   (pop(gene,pop_member))*10^place;
 place_pointer=place_pointer+1;
end

% Determine sign of the traits and fix
% trait(current_trait,pop_member,popcount) so that it has the right sign:

if pop(TRAIT_START(current_trait),pop_member) < 5
  trait(current_trait,pop_member,popcount)=...
    -trait(current_trait,pop_member,popcount);
end


        end % Ends "for current_trait=..." loop

end    % Ends "for pop_member=..." loop

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Terminate the program when the best fitness has not changed
% more than EPSILON over the last DELTA generations.  It would also
% make sense to use avefitness rather than bestfitness in this test.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  if popcount > DELTA+1 & ...
     max(abs(bestfitness(popcount-DELTA:popcount-1)-...
       bestfitness(popcount-DELTA-1:popcount-2)))<=EPSILON
     break;
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end  % End "for pop_count=..." loop - the main loop.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sys =[tau_best Kt1_best Kt2_best];

%end mdlOutputs
```

B.4 TUNING PD GAINS USING GENETIC ALGORITHM

```
%*******************Contro of   2 Link Underactuaed Manipulator **************%
%****** S_function  m File   for Tuning PD Gains using Genetic Algorithm  **************%
%Written By : Joudeh Yasin              Advised By: Dr. Mohd Aljarrah
%                                      Date:14.004.05
%************************************************************% %
%

function [sys,x0,str,ts]=gatest30(t,x,u,flag,h1,h2,h3,h4,h5)



switch flag,

  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0,
    [sys,x0,str,ts] = mdlInitializeSizes(h1,h2,h3,h4,h5);

  %%%%%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%%%%
  case 2,
    sys = mdlUpdate(t,x,u);

  %%%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%%
  case 3
    sys = mdlOutputs(t,x,u,h1,h2,h3,h4,h5);

  %%%%%%%%%%%%
  % Terminate %
  %%%%%%%%%%%%
case 9,
    sys = [];

  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end


%end simom2

%
%=============================================================================
=====
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
=====
%
function [sys,x0,str,ts] = mdlInitializeSizes(h1,h2,h3,h4,h5)

sizes = simsizes;

sizes.NumContStates  = 0;
sizes.NumDiscStates  = 9;
sizes.NumOutputs     =3;
```

```
sizes.NumInputs      = 9;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);


x0  = 0;
str = [];
ts  = [-1 0];  % inherited sample time

%
function sys = mdlUpdate(t,x,u)

sys = u;

%end mdlUpdate

%
%=============================================================================
% mdlOutputs
% Return the output vector for the S-function
%=============================================================================
%
function sys = mdlOutputs(t,x,u,h1,h2,h3,h4,h5)

% Evaluation of Dynamic Equation Parameters
g=9.81;
D=[(h1+h2+2*h3*cos(x(3)))  (h2+h3*cos(x(3))); (h2+h3*cos(x(3))) h2];
C=[(-h3*sin(x(3))*x(4)) (-h3*sin(x(3))*x(4)-h3*sin(x(3))*x(2));h3*sin(x(3))*x(2) 0];
G=[(h4*g*cos(x(1))+h5*g*cos(x(1)+x(3)));h5*g*cos(x(1)+x(3))];
F=-inv(D)*(C*[x(2) x(4)]'+G);
B=inv(D)*[1 0]';

d11_b=D(1,1)-D(1,2)*D(2,1)/D(2,2);
c11_b=C(1,1)-D(1,2)*C(2,1)/D(2,2);
c12_b=C(1,2);
G_b=G(1,1)-D(1,2)*G(2,1)/D(2,2);



% Defining the Sliding Surfaces
c1=8;
mu=30;
k=24;
Ts=0.005;
kd=28;
kp=310;


v1=x(7) +kd*(x(6)- x(2))+kp*(x(5)-x(1));
tau=d11_b*v1+c11_b*x(2)+c12_b*x(4)+G_b;

best_fittness=-1000;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% User: Input your own parameters below (the ones given are for the
%       optimization proble   m given above):
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 NUM_TRAITS=2;         % Number of traits in each individual
  HIGHTRAIT=[450 28];   % Upper limit of a trait
   LOWTRAIT=[350 20];      % Lower limit of a trait
  SIG_FIGS=[6 6]';      % Number of genes in each trait
  DECIMAL=[3 2];        % Order of magnitude the trait
  MUTAT_PROB=0.05;      % Probability of mutation (typically <.1)
  CROSS_PROB=0.6;       % Probability of crossover (typically near 1)
  SELF_ENTERED=0;       % "0": a random initial population.
                 % "1": a specified initial population
                 % If you choose "1", enter it in the program.
  POP_SIZE=50;          % Number of individuals in the population
  ELITISM=0;            % Elitism ON/OFF, 1/0
  DELTA=6;              % Number of generations to be counted
                 % for the termination criteria.
  EPSILON = 0.01;               % Range that the fitness must change
                 % in the termination criteria.
  MAX_GENERATION=20;   % Number of times the loop will run before

rand('state',0) % Reset the random number generator so each time you re-run
           % the program with no changes you will get the same results.



popcount=1;                   % Initialize the generation count,
                                                 % set it to one, the first population

if SELF_ENTERED == 0          % Make a random initial population for
                                                                  % base-10 operation by
specifying the
                                                                  % matrix of initial traits


for pop_member = 1:POP_SIZE
     for current_trait = 1:NUM_TRAITS,
                   trait(current_trait,pop_member,popcount)=...
                   (rand-(1/2))*(HIGHTRAIT(current_trait)-LOWTRAIT(current_trait))+...
                   (1/2)*(HIGHTRAIT(current_trait)+LOWTRAIT(current_trait));
                             % This starts the population off with numbers chosen randomly
                                  % on the allowed range of variation, for this example.
           end
end

else

for pop_member = 1:POP_SIZE
      for current_trait = 1:NUM_TRAITS,
         trait(current_trait,pop_member,popcount)=0;
                           % To start with a guess where all the population members are zero

                   end
end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Next, we need to set some values based on the values input by
```

```
% the user.  In particular, we determine the length of the
% chromosome and start point of each trait.  This information
% will be used later in the main algorithm.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  CHROM_LENGTH=sum(SIG_FIGS)+NUM_TRAITS; % Length of the chromosome is
                                                                                % the
number of sig. figs. plus
                                                                                % the
number of sign positions
  TRAIT_START(1)=1;                                  % Initialize: the first trait
                                                                                % starts at
the first digit
                                                                                % (this is
the sign digit)

  for current_trait=1:NUM_TRAITS,          % Determine the start point of the
                                                                        % other traits - it is
the start of
                                                                        % the last trait plus
the no. of sig.
                                                                        % figs. plus one for
sign
TRAIT_START(current_trait+1)=...
  TRAIT_START(current_trait)+SIG_FIGS(current_trait)+1;
  % Yes, we compute the TRAIT_START for one extra trait - this
  % is used for convenience in the code below.
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% In this next loop the fitness is calculated, the children are
% created and it repeats until the EPSILON-DELTA termination condition
% is satisfied or MAX_GENERATION is reached.  This is the main
% loop of the GA.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while popcount <= MAX_GENERATION

% First, fix bad traits (i.e., ones that are out of the range
% specified by HIGHTRAIT and LOWTRAIT) by saturation at the extremes

for pop_member = 1:POP_SIZE

for current_trait = 1:NUM_TRAITS,


if trait(current_trait,pop_member,popcount)>HIGHTRAIT(current_trait)

                % The trait has went higher than the upper
                % bound so let the trait equal to the
                % HIGHTRAIT bound.

trait(current_trait,pop_member,popcount)=HIGHTRAIT(current_trait);

% Now consider the other case:

elseif trait(current_trait,pop_member,popcount)<LOWTRAIT(current_trait)
```

```
                % The trait has went lower than the lower
                % bound so let the trait equal to the
                % LOWTRAIT bound

trait(current_trait,pop_member,popcount)=LOWTRAIT(current_trait);

                    end

% Now that we have reset the traits to be in range, we must
% convert them to the chromosome form for use with the genetic operators.
% First, we transfer the sign of the trait into the chromosome

            if trait(current_trait,pop_member,popcount) < 0
              pop(TRAIT_START(current_trait),pop_member)=0;
            else
              pop(TRAIT_START(current_trait),pop_member)=9;
            end

% Next, strip off the sign and store the resulting value in a
% temporary variable that is used in the construction of pop

temp_trait(current_trait,pop_member)=...
        abs(trait(current_trait,pop_member,popcount));
                    % temp_trait is trait without the sign of trait

% Next, we store the numbers of the trait in the chromosome:
% First, set up a temporary trait with at most
% one nonzero digit to the left of the decimal point.
% This is used to strip off the numbers to put
% them into a chromosome.

temp_trait(current_trait,pop_member)=...
temp_trait(current_trait,pop_member)/10^(DECIMAL(current_trait)-1);

% Encode the new trait into chromosome form

for make_gene = TRAIT_START(current_trait)+1:TRAIT_START(current_trait+1)-1,

% For each gene on the trait make the gene the corresponding digit on
% temp_trait (note that rem(x,y)=x-roundtowardszero(x/y)*y or
% rem(x,1)=x-roundtowardszero(x) so that rem(x,1) is the fraction part
% and x-rem(x,1) is the integer part so the next line makes the location in
% pop the same as the digit to the left of the decimal point of temp_trait

pop(make_gene,pop_member)=temp_trait(current_trait,pop_member)-...
  rem(temp_trait(current_trait,pop_member),1);

% Next, we take temp_trait and rotate the next digit to the left so that
% next time around the loop it will pull that digit into the
% chromosome.  To do this we strip off the leading digit then shift
% in the next one.

temp_trait(current_trait,pop_member)=...
 (temp_trait(current_trait,pop_member)-pop(make_gene,pop_member))*10;

end

      end % Ends "for current_trait=..." loop
```

```
            end    % Ends "for pop_member=..." loop


sumfitness = 0;                 % Re-initialize for each generation

% First, determine the values of the function to be minimized for
% each chromosome.

   for chrom_number = 1:POP_SIZE,       % Test fitness

 kp=trait(1,chrom_number,popcount);
 kd=trait(2,chrom_number,popcount);

v1=x(7) +kd*(x(6)- x(2))+kp*(x(5)-x(1));
tau(chrom_number)=d11_b*v1+c11_b*x(2)+c12_b*x(4)+G_b;
Sum_S1=0;
x1=x(1);
x2=x(2);
x3=x(3);
x4=x(4);
for j=1:10
S_abs=abs(x1-pi/2)+abs(x2)+abs(x3)+abs(x4);
Sum_S1=S_abs+Sum_S1;
x2=(F(1)+B(1)*tau(chrom_number))*Ts+x2;
x4=(F(2)+B(2)*tau(chrom_number))*Ts+x4;
x1=x2*Ts+x1;
x3=x4*Ts+x3;
D=[(h1+h2+2*h3*cos(x3))  (h2+h3*cos(x3)); (h2+h3*cos(x3)) h2];
C=[(-h3*sin(x3)*x4) (-h3*sin(x3)*x4-h3*sin(x3)*x2);h3*sin(x3)*x2 0];
G=[(h4*g*cos(x1)+h5*g*cos(x1+x3));h5*g*cos(x1+x3)];
F=-inv(D)*(C*[x2 x4]'+G);
B=inv(D)*[1 0]';
end

 fitness(chrom_number)=1/(Sum_S1+0.000001);


sumfitness =sumfitness + fitness(chrom_number);

end

% Next, determine the most fit and least fit chromosome and
% the chrom_numbers (which we call bestmember and worstmember).

[bestfitness(popcount),bestmember]=max(fitness);
[worstfitness(popcount),worstmember]=min(fitness);

% Next, save these (if want to save worstindividual can too)
if (fitness(bestmember)>=best_fittness)
    tau_best=tau(bestmember);
 best_fittness=fitness(bestmember);
 kp_best=trait(1,bestmember,popcount);
 kd_best=trait(2,bestmember,popcount);

end

bestindividual(:,popcount)=trait(:,bestmember,popcount);
%worstindividual(:,popcount)=trait(:,worstmember,popcount);

% Compute the average fitness in case you want to plot it.
```

```
avefitness(popcount) = sumfitness / POP_SIZE;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%
% Create the next generation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% First, form the mating pool.
% To do this we select as parents the
% chromosomes that are most fit.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 for pop_member = 1:POP_SIZE,
       if ELITISM ==1 & pop_member==bestmember  % If elitism on, and have
                                                          %
the elite member
            parent_chrom(:,pop_member)=pop(:,pop_member); % Makes sure that
                     % the elite member gets into the next
                                   % generation.
       else
   pointer=rand*sumfitness; % This makes the pointer for the roulette
              % wheel.
   member_count=1;      % Initialization
   total=fitness(1);

   while total < pointer,             % This spins the wheel to the
                                             % pointer and finds
the
                                             % chromosome
there - which is
                                             % identified by
member_count
     member_count=member_count+1;
     total=total+fitness(member_count);
   end

% Next, make the parent chromosome

  parent_chrom(:,pop_member)=pop(:,member_count);
       end
 end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
% Reproduce section (i.e., make off-spring - "children")
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
% In the approach below each individual gets to mate and
% they randomly pick someone else (not themselves) in the
% mating pool to mate with.  Resulting children are
% composed of a combination of genetic material of their
% parents.  If elitism is on, when the elite member gets
% a chance to mate they do not; they are simply copied
% over to the next generation.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
```

```
for parent_number1 = 1:POP_SIZE,    % Crossover (parent_number1 is the
                                                                          % individual who gets
to mate
        if ELITISM ==1 & parent_number1==bestmember % If elitism on, and
                                                                                             %
have the elite member
                child(:,parent_number1)=parent_chrom(:,parent_number1);
        else
          parent_number2=parent_number1;  % Initialize who the mate is
          while parent_number2 == parent_number1   % Iterate until find
                                       % a mate other than

        % yourself
      parent_number2 = rand*POP_SIZE; % Choose parent number 2
                                                                        % randomly
(a random mate)
      parent_number2 = parent_number2-rem(parent_number2,1)+1;
          end
  if CROSS_PROB > rand          % If true then crossover occurs

      site = rand*CHROM_LENGTH;   % Choose site for crossover
      site = site-rem(site,1)+1;  % and make it a valid integer
                                               % number for a site

% The next two lines form the child by the swapping of genetic
% material between the parents

child(1:site,parent_number1)=parent_chrom(1:site,parent_number1);

child(site+1:CHROM_LENGTH,parent_number1)=...
    parent_chrom(site+1:CHROM_LENGTH,parent_number2);

     else                       % No crossover occurs

% Copy non-crossovered chromosomes into next generation
% In this case we simply take one parent and make them
% the child.

child(:,parent_number1)=parent_chrom(:,parent_number1);

         end
   end  % End the "if ELITISM..." statement
end  % End "for parent_number1=..." loop

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
% Mutate children.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
% Here, we mutate to a different allele
% with a probability MUTAT_PROB
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   for pop_member= 1:POP_SIZE,

        if ELITISM ==1 & pop_member==bestmember  % If elitism on, and
                                                                                             %
have the elite member
                child(:,pop_member)=child(:,pop_member); % Do not mutate
```

```
                                                                            %
the elite member
     else
      for site = 1:CHROM_LENGTH,

   if MUTAT_PROB > rand        % If true then mutate
     rand_gene=rand*10;                 % Creat a random gene

     % If it is the same as the one already there then
     % generate another random allele in the alphabet

     while child(site,pop_member) == rand_gene-rem(rand_gene,1),
       rand_gene=rand*10;
     end;

     % If it is not the same one, then mutate

     child(site,pop_member)=rand_gene-rem(rand_gene,1);

     % If takes a value of 10 (which it cannot
     % mutate to) then try again (this is a very low probability
                  % event (most random number generators generate numbers
                  % on the *closed* interval [0,1] and this is why this line
                  % is included).

     if rand_gene == 10
        site=site-1;
     end
   end  % End "if MUTAT_PROB > rand ...
        end  % End for site... loop
       end  % End "if ELITISM..."
 end  % End for pop_member loop

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
% Create the next generation (this completes the main part of the GA)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%

 pop=child;             % Create next generation (children
                                                 % become parents)

 popcount=popcount+1;             % Increment to the next generation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%
% Next, we have to convert the population (pop) to the base-10
% representation (called trait) so that we can check if the traits
% all still lie in the proper ranges specified by HIGHTRAIT and LOWTRAIT
% at the beginning of the next time around the loop.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%

for pop_member = 1:POP_SIZE

     for current_trait = 1:NUM_TRAITS,

trait(current_trait,pop_member,popcount)=0; % Initialize variables
place_pointer=1;
```

```
% Change each of the coded traits on the chromosomes into base-10 traits:
% For each gene on the current_trait past the sign digit but before the
% next trait find its real number amount and hence after finishing
% the next loop trait(current_trait,pop_member,popcount) will be the base-10
% number representing the trait

for gene=TRAIT_START(current_trait)+1:TRAIT_START(current_trait+1)-1,
 place=DECIMAL(current_trait)-place_pointer;
 trait(current_trait,pop_member,popcount)=...
 trait(current_trait,pop_member,popcount)+...
   (pop(gene,pop_member))*10^place;
 place_pointer=place_pointer+1;
end

% Determine sign of the traits and fix
% trait(current_trait,pop_member,popcount) so that it has the right sign:

if pop(TRAIT_START(current_trait),pop_member) < 5
  trait(current_trait,pop_member,popcount)=...
    -trait(current_trait,pop_member,popcount);
end


          end % Ends "for current_trait=..." loop

end    % Ends "for pop_member=..." loop

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Terminate the program when the best fitness has not changed
% more than EPSILON over the last DELTA generations.  It would also
% make sense to use avefitness rather than bestfitness in this test.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  if popcount > DELTA+1 & ...
     max(abs(bestfitness(popcount-DELTA:popcount-1)-...
       bestfitness(popcount-DELTA-1:popcount-2)))<=EPSILON
     break;
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end  % End "for pop_count=..." loop - the main loop.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


if (x(8)<=0.5)
  tau_best =-0.0435;
  sys =[tau_best  0  0];
else
sys =[tau_best  kp_best  kd_best];

end
%end mdlOutputs
```

B.5 PARAMETER IDENTIFICATION OF MANIPULTOR AND LQR METHOD

```
%*******************Control of   2 Link Underactuaed Manipulator **************%
%***************** Evaluating Manipultor Paarametrs and LQR Gains **************%
%Written By : Joudeh Yasin              Advised By: Dr. Mohd Aljarrah
%                                       Date:14.004.05
%*************************************************************%  %
% this program Linearuze the sytem at the upright posituin and calculates
% the best gains the stabilize the manipulator using the LQR method

%******* Evaluating the state feedback gains using LQR method **%

% ****   initializing the physical  & geometric data of link#1
m1=0.06;
L1=0.08;
Lc1=0.06;
I1=0.000128;
g=9.81;
% ****   initializing the physical  & geometric data of link#2;
m2=0.04;
L2=0.12 ;
Lc2=0.06;
I2=0.00021845;

%*** Grouping Dynamic parameters into five parameters the

h1=m1*Lc1^2+m2*L1^2+I1;
h2=m2*Lc2^2+I2;
h3=m2*L1*Lc2;
h4=m1*Lc1+m2*L1;
h5=m2*Lc2;
%******* Linearizing the system about the upright position *****%
a21=(h2*h4-h3*h5)*g/(h1*h2-h3*h3);
a23=-h3*h5*g/(h1*h2-h3*h3);
a41=(h5*g*(h1+h3)-h4*g*(h2+h3))/(h1*h2-h3*h3);
a43=h5*g*(h1+h3)/(h1*h2-h3*h3);

A=[0 1 0 0;a21 0 a23 0;0 0 0 1;a41 0 a43 0];
B=[0 h2/(h1*h2-h3*h3) 0 (-h2-h3)/(h1*h2-h3*h3)]';
%******* Evauating the state feedback Gains  *****%
H1=[1/(pi/2) 0 1 0];              % H is the desired output stae
rho=1;
Q=rho*H1'*H1;                     % Evaluting First Q matrix Choice
H2=[1/(pi/2) 0 0 0];
rho=0.1;
Q2=rho*H2'*H2;
Q3=rho*[(2/pi)^2 0 0 0;0 1/10000 0 0;0 0 1 0;0 0 0 1/10000];
R2=[10000];
R=[1];
[Kqr,S,E] = LQR(A,B,Q,R); % Evaluating the Gians Kqr using LQR method
[Kqr1,S,E] = LQR(A,B,Q,R); % Evaluating the Gians Kqr using LQR method
[Kqr2,S,E] = LQR(A,B,Q2,R); % Evaluating the Gians Kqr using LQR method

%*********************************************************************
%*********** Evaluating Sliding Mode Controller ***********************
% Evaluation of Dynamic Equation Parameters
x=[(pi/2 -0.1) 0.05  0.08 0.05];
g=9.81;
a=readfis('anfsw6');
bal=readfis('anfb5');
```

APPENDIX C

EXPERIMENTAL SETUP

C. EXPERIMENTAL SETUP

C.1 Mechanical Design of the Manipulator

The detailed drawings of the manipulator are shown in Figure (1) and Figure (2). The Photography of manipulator is shown in Figure (3) .The two links are made of aluminum and have bronze coupler at each end. The length of link #1 is selected to be 8cm; the other dimensions of the link are selected to be 2 cm width and 4 mm thickness. The length of link#2 is selected to be longer than link#1, to fascinate its balancing .Thus it is selected to be 12 cm, the other dimensions of the link are same as link#1 .

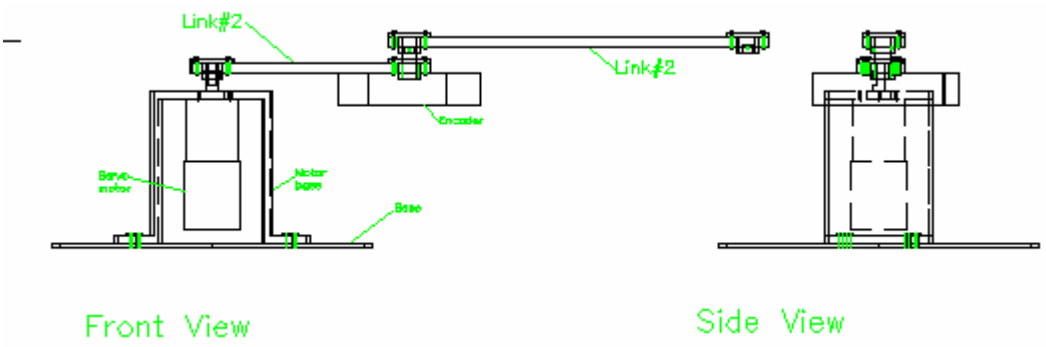


Figure 1 Front and side view of manipulator

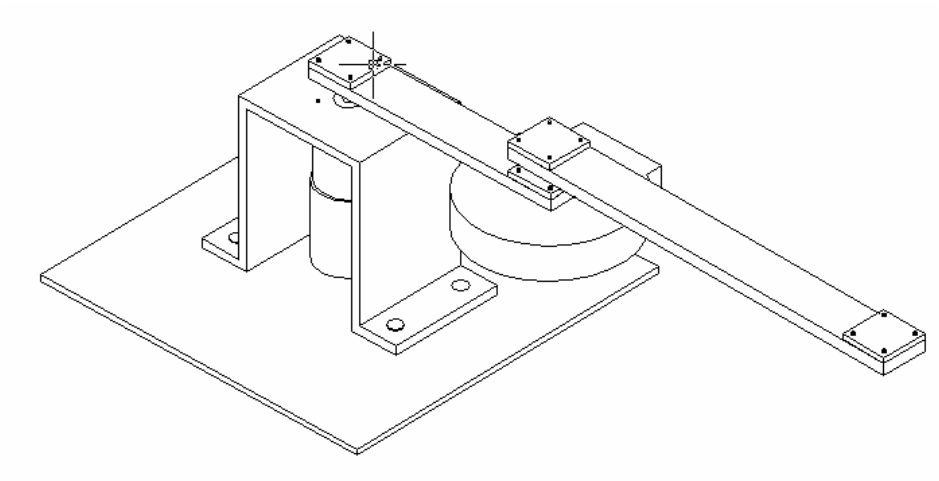


Figure 2 Isometric view of manipulator

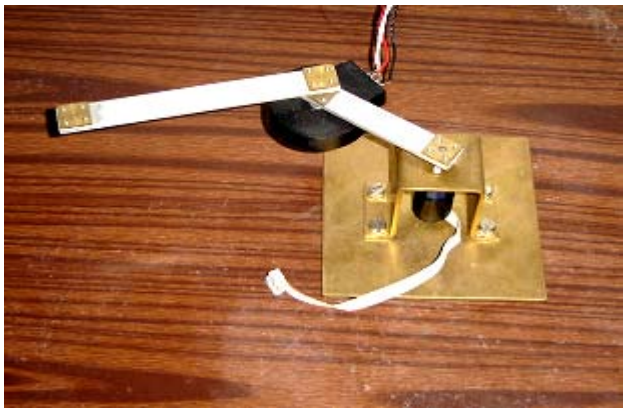



Figure 3 Photos of manipulator

Using Sim-Mechanics tool box in Matlab Simulink, the dynamic decoupling of the manipulator is verified and the maximum torque that servo motor should produce is computed and found to be 0.1 N.m. During the simulation of swinging up, the maximum instantaneous torque was found to be 0.7 N.m and the maximum relatively constant torque was found to be 0.3 N.m. In addition, the maximum angular velocity 40 rad/s (382 rpm). Considering these design requirements we have selected a servo motor of 0.5 N.m continuous torques, 0.7 N.m instantaneous torque, and speed of 804 rpm. The data sheet of servo motor and gear box is shown in Appendix II.

The Servo motor is fixed and hanged on motor base and coupled with the first end link one (Shoulder of the manipulator).The encoder on the passive joint (second joint) is hanged on link#1 and coupled with link#2.That is, the encoder shaft rotates as link#2 rotates and weight of encoder is added to link#1. The geometric and mass properties of manipulator are shown Table (1)

Table (1) *Geometric and Mass properties of Manipulator*

Parameter	Value	Parameter	Value
m_1	0.06 kg	lc_1	6 cm
m_2	0.06 kg	lc_2	8 cm
$l1$	8 cm	I_1	0.00012800 kg.m^2
$l2$	12 cm	I_2	0.00021845 kg.m^2

C.2 Hardware Interfacing Circuit:

The interfacing circuit of the experimental setup is shown in Figure (3.4).

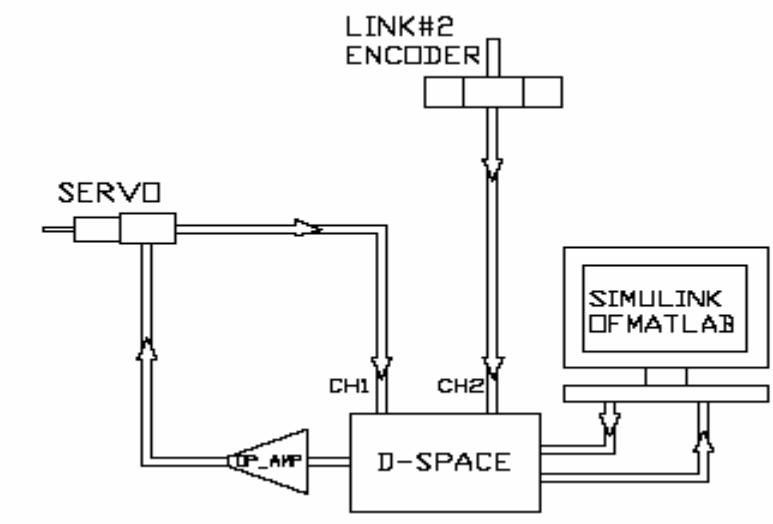


Figure 4 Interfacing Circuit of Experimental Setup

The D-space receives the control signal from Matlab Simulink and transmits it through the H_bridge to the servo motor .The servo motor rotates accordingly and sends the angular position of link#1 to D-space through channel one. The second link rotates because of dynamic coupling and sends its position by the encoder to D-space through channel two. D-space translates and transmits the positions of both links to Matlab Simulink .The controllers of manipulators are built inside Simulink .The genetic sliding mode balancing controller and adaptive genetic algorithm for PD gain-tuning are built inside S-function blocks.

C.3 Features of D-Space:

The DS1104 R& D Controller Board provides the following features:

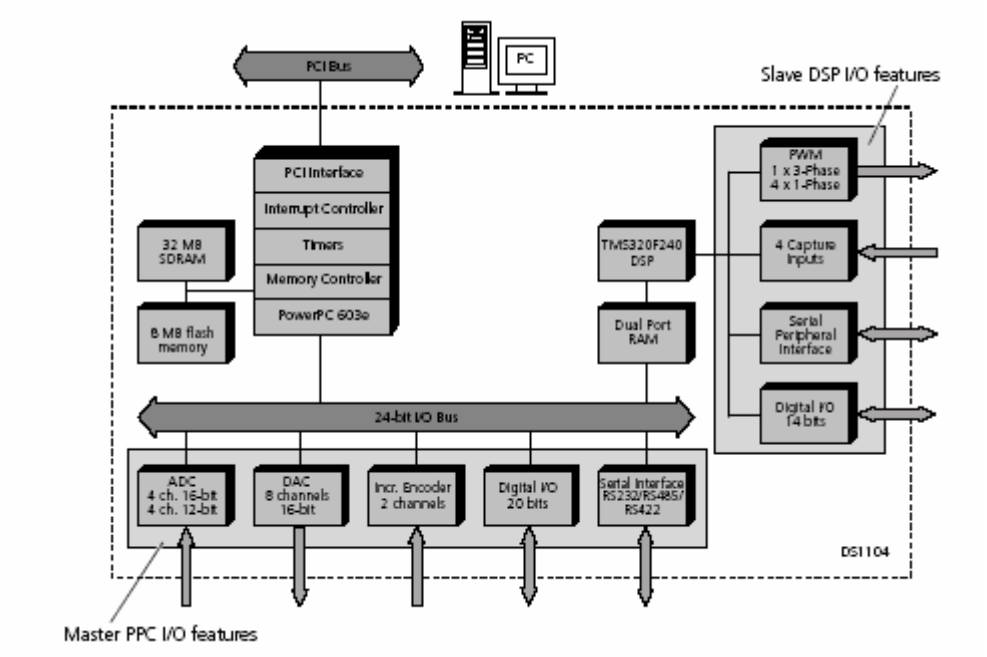


Figure 5 Features of D_Space DS1104

The detailed features of the used components in our application are demonstrated below:

Master PPC representing the computing power of the board, and featuring several I/O units. and it is running at 250 MHz (CPUclock).

DAC Unit: A/D converter has the following characteristics:

■8parallel DAC channels (signalsDACH1…DACH8)

■ 16-bit resolution

■±10V output voltage range

■± 1mV offset error, 13ppm/Koffsetdrift

■±0.1%gain error, 25ppm/Kgaindrift

■>80dBsignal-to-noise ratio (SNR)

Incremental Encoder Interface: It has the following characteristics:

■Input channels for two digital incremental encoders

■24-bit position counters

■ 1.65 MHz maximum encoder line count frequency.

Digital Encoders:

To have high precision in the reading of joint angles from digital encoders, the resolution of encoders should be at least 1440 (4 pulse/degree).Also to avoid slipping

of encoder signals, the frequency of encoder signals should be less than 1.65 MHz as per Incremental Encoder Interface of D-space. For joint one, the maximum angular velocity of servo motor is 804 rpm (13.4 rev/s), and the maximum resolution of joint one encoder is calculated to be 342 signal/degree. Consequently, we have selected the encoder of servo motor with 512 resolution and 9.7 gear ratio, which gives total resolution 4966 (13.8 pulse /degree). The maximum angular velocity of link two is 47 rad/s (7.5 rev/s) which is less than maximum servo motor velocity. As a result we selected an encoder of 2000 resolution (5.5 pulse/degree).

C.4 Power Driving Circuit:

The servo motor is driven through non inverting power amplifier as shown in Figure 3.6

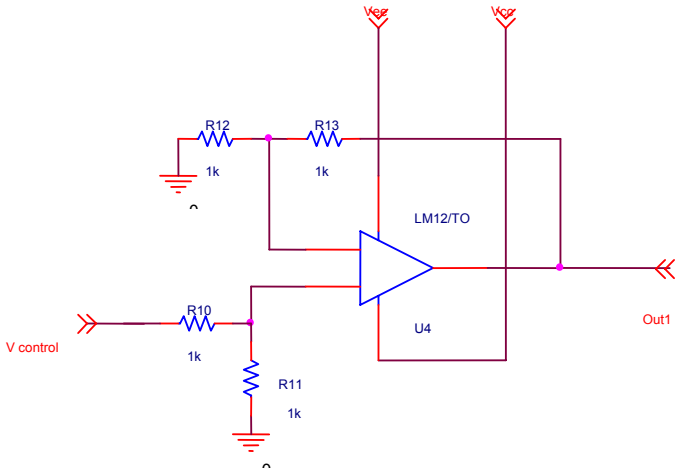


Figure 6 Non_inverting OP_AMP

The V_{cc}, V_{ee} terminals are connected to the external power supply, the power amplifier receives the$V_{control}$ from D-space.

C.5 Simulink Interfacing Block Diagram :

The Simulink Block diagram that drives the servo motor and receives encoder signals is shown in Figure 7

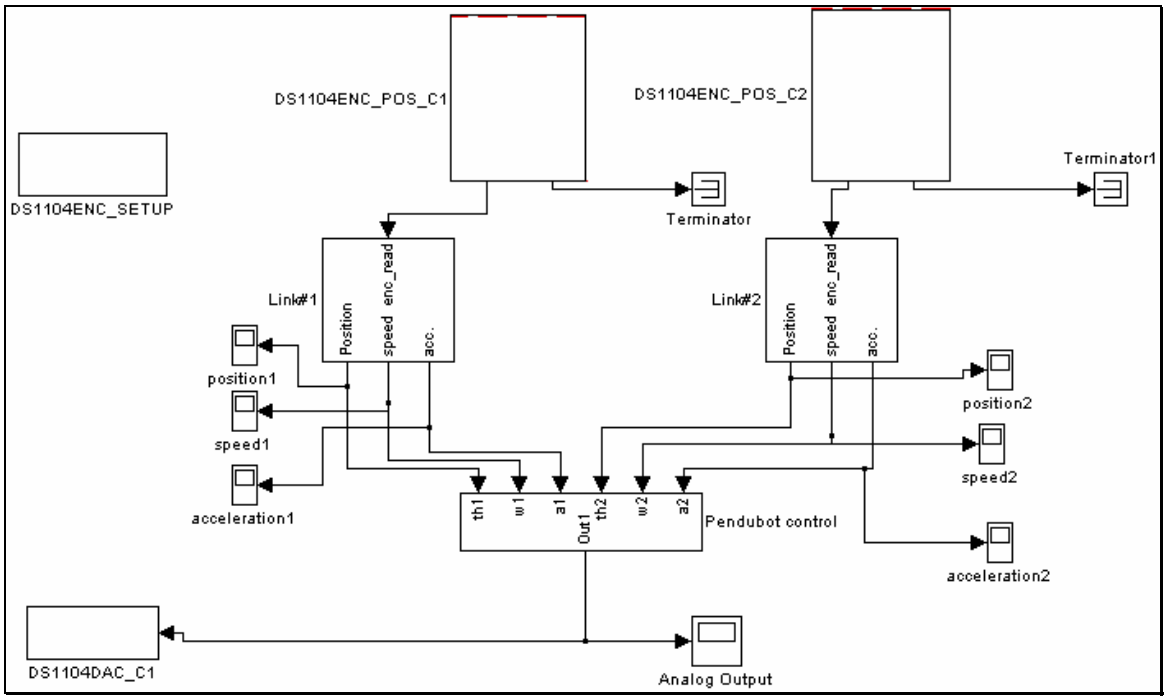


Figure 7 Simuliink Interfacing Block diagram

The blocks DS1104ENC-PS-C1, DS1104ENC-PS-C2 send the position readings of encoders to the blocks Link#1 and Link#2 respectively, where the angular velocities, acceleration are filtered and calculated. The states of manipulator are sent to the 'Pendubot control 'block. In Pendubot control the swinging up and balancing controllers are applied and the required $V_{control}$ is computed and sent to DAC channel in D_Space.

The sampling time is selected to be 0.005 sec, so that it is less than one-tenth of the natural frequency of system (continuous model can be approximated) and noise is reduced significantly.

VITA

Joudeh Yasin Abed was born on October 30, 1976, in Amman, Jordan. He was educated in local public schools and graduated from AL Hussain College Secondary School in 1994. He joined Jordan University of Science and Technology and graduated in 2000. His degree was a Bachelor of Science in Mechanical Engineering.

Mr. Joudeh Yasin moved to the United Arab Emirates in 2000 and worked as project engineer for Reliance Electromechanical Contracting Company for six years. Meanwhile Mr. Joudeh Yasin began a master's program in Mechatronics Engineering at American University of Sharjah. He was awarded the Master of Science degree in Mechatronics Engineering in 2006.

Mr. Joudeh Yasin is a member of the Jordanian Engineering Association.